TRABAJO FIN DE CARRERA

TÍTULO: Análisis de algoritmos multicast en redes overlay

AUTOR: Sergio Domínguez Segura

DIRECTOR: Javier Ozón Górriz

FECHA: 10 de Octubre de 2006

TÍTULO: Análisis de algoritmos multicast en redes overlay

AUTOR: Sergio Domínguez Segura

DIRECTOR: Javier Ozón Górriz

FECHA: 10 de Octubre de 2006

Resumen

En este trabajo se presenta y analiza un algoritmo capaz de resolver el problema de encaminamiento multicast. Aunque el algoritmo se puede aplicar en cualquier nivel de la torre de protocolos, en este documento se estudia su utilización en el nivel de aplicación. Este algoritmo permite crear tablas de encaminamiento entre los nodos de un mismo grupo multicast, con el objeto de minimizar el tiempo que tarda la información en llegar a todos los nodos del grupo.

Los algoritmos empleados limitan la cadencia de cada uno de los nodos en función de una variable b_0 que depende del tiempo de transmisión del nodo origen y de un valor *S* (que es el número máximo de veces que el nodo origen puede transmitir un mismo paquete). Para estudiar las características y el comportamiento del algoritmo, se ha realizado una serie de simulaciones sobre una red virtual que modela la red IP y sobre la cual se ha definido una segunda red formada por los nodos de usuarios. En esta nueva red, creada en el nivel de aplicación (y que por lo tanto podemos llamar *red overlay*) hemos ejecutado tres modelos del algoritmo, el modelo *cadencia fija,* el modelo *cadencia fija variable S,* y por último, el modelo *cadencia fija forzada.*

Title: Analysis of multicast algorithms for overlay networks

Author: Sergio Domínguez Segura

Director: Javier Ozón Górriz

Date: October, 10th 2006

Overview

In this work we present an algorithm for the problem of multicast routing. Though the algorithm could work at different levels, in this document we apply it at the application layer. The algorithm defines routing tables between nodes of the same multicast group with the aim of minimizing the transmission delay.

The algorithm limits the cadence of all nodes as a function of a variable called b_0, which depends on the transmission time of the root node, and also on a second variable *S* which is the maximum number of times that the root node can transmit a packet. To study the characteristics of the algorithm we perform simulations on a virtual IP network. Over this network, we define a second network of users' nodes. In this new network, which belongs to application layer (and therefore we can call it *overlay network*), we have applied three different types of the algorithm: *cadence fixed, cadence fixed variable S* and *cadence fixed force*.

ÍNDICE

INTRODUCCIÓN……1

CAPÍTULO 1. TRANSMISIÓN MULTICAST Y MODELADO DE LA RED IP….5

1.1 El problema de la transmisión multicast……5

1.2 El modelo cartero.....6

1.3 Modelado de la red IP.....7
1.3.1 Modelado de redes reales……8
1.3.2 Modelado de cualquier tipo de red……9
1.3.3 El modelo utilizado……12

CAPÍTULO 2. TEORÍA DE GRAFOS……15

2.1 Definiciones……..15

2.2 Búsqueda de caminos mínimos. El algoritmo de Dijkstra……17

CAPÍTULO 3. ALGORITMO MULTICAST……..19

3.1 El modelo inicial para un sólo paquete 19

3.2. Un modelo sencillo para enviar M paquetes 20

3.3. El retardo Stream Multicast 20

CAPÍTULO 4. APLICACIÓN DEL ALGORITMO..... 23

41. Procedimiento para simular el algoritmo..... .23

4.2. Obtención del grafo troncal..... 23

4.3. Ampliación del grafo troncal con los nodos de usuario 25

4.4. La red overlay: aplicación del método de Dijkstra..... 26

4.5. Simulación de los algoritmos..... 27

CAPÍTULO 5. RESULTADOS Y CONCLUSIONES……..29

BIBLIOGRAFIA……...37

INTRODUCCIÓN

El número de ordenadores conectados en la red de Internet crece año tras año, así como las aplicaciones que se pueden ejecutar. En muchos casos, estas aplicaciones se basan en la transmisión desde un ordenador (que podríamos llamar fuente de información) hasta uno o diversos ordenadores, que pueden estar situados en una red cercana o del mismo tipo de topología que la fuente, o bien en redes lejanas y con topologías completamente diferentes.

Desde las primeras redes de ordenadores, ya sea a nivel de enlace (como Ethernet) o a nivel de red (como la red IP) ha sido relativamente sencillo hacer transmisiones desde un punto hacia otro (transmisión unicast). En cambio, la transmisión de información de un punto hasta un grupo variable de receptores (transmisión multicast) es un problema no solucionado de manera satisfactoria. En la actualidad, muchos servicios se basan en este tipo de transmisión: una videoconferencia entre múltiples interlocutores, los participantes en una misma partida de un juego on-line, o el conjunto de usuarios que se intercambian un mismo archivo utilizando una red P2P, son tres ejemplos de transmisión multicast.

Una posible solución a este problema es “IP Multicast”, una propiedad del encaminamiento IP que permite definir una única dirección IP “de grupo” para un conjunto de ordenadores. Pero esta solución también tiene sus inconvenientes, ya que se trata de una actualización que se hizo sobre el estándar del encaminamiento IPv4. Las limitaciones son las siguientes:

- Rango de direcciones IP multicast limitado, de manera que nos encontramos con una cantidad limitada de grupos.

- Algoritmos de encaminamiento complejos, como el PIM, que se encargan de formar los grupos añadiendo los nuevos miembros y eliminando los antiguos, pasando esta información a los routers y otros elementos intermedios de la red que participan en la transmisión.

- Todos los routers, firewalls y elementos de nivel de red han de entender IP Multicast, cosa que no siempre sucede, de manera que para utilizarlo deben cambiarse físicamente los elementos de la red.

Lo que proponemos en este TFC es otra posible solución al problema de la transmisión multicast. Se trata de una familia de algoritmos de punto a multipunto capaces de funcionar tanto a nivel de enlace como en el de red o aplicación –debido al grado de abstracción con el que se han definido– y que presenta como principales características:

- Implementación sencilla.

- Bajo retardo de transmisión: se demuestra que el algoritmo es óptimo en condiciones de estabilidad.

- Alta escalabilidad: no hace falta variar el algoritmo según el tamaño de la red. Además, el algoritmo se puede adaptar fácilmente al dinamismo de los nodos.

Esta familia de algoritmos responde a un esquema simple. Dado un nodo origen (que llamaremos raíz) y un conjunto de nodos destinos (que formarán parte de un mismo grupo multicast), el algoritmo ha de ser capaz de encontrar, de la manera más sencilla posible, un conjunto de rutas óptimas que minimicen el retardo total del envío de información desde el nodo raíz hasta todos los nodos destinatarios. La información que se ha de transmitir puede ser un único paquete de datos o un número determinado de paquetes, cosa que da pie a diferentes modelos de los algoritmos.

Inicialmente, se ha definido el algoritmo para la transmisión de un sólo paquete desde el nodo raíz hasta el resto de nodos. De todas maneras, como en la práctica la información que se ha de enviar suele ocupar más de un único paquete de datos, los algoritmos se han de modificar ligeramente para poder enviar los paquetes uno tras otro. El funcionamiento básico del algoritmo requiere que los nodos que hayan recibido un paquete lo vaya enviando a diferentes destinatarios mientras este paquete no haya llegado a todos los miembros del grupo. En otras palabras, para conseguir el tiempo óptimo cuando se envía un sólo paquete, el nodo origen deberá transmitir el paquete de datos mientras éste no llegue a todos los miembros del grupo. En el caso de enviar más de un paquete, sin embargo, el nodo raíz puede empezar a enviar el segundo paquete antes de que el primero haya llegado a todos los nodos. De este modo, el primer nodo tardará más en llegar a todos los destinatarios pero se empezará a enviar antes el segundo, y más adelante el tercero y así sucesivamente, y al final se podrá recuperar el retraso en la transmisión del primer paquete.

Para llevar a cabo esta modificación, se ha añadido al algoritmo una pequeña condición de comportamiento, de manera que se limitará el número de veces que un nodo puede enviar un paquete.

Con el objeto de estudiar y comprobar las ventajas de este conjunto de algoritmos, en este proyecto hemos trabajado sobre diferentes redes simuladas. De hecho, hemos comenzado con la recreación virtual de una posible red troncal de Internet, de donde obtenemos un grafo con dos tipos diferentes de redes: las redes de tránsito, con enlaces de alta velocidad y tiempo de propagación elevado, y redes de acceso, con velocidades inferiores y tiempo de propagación bajo.

Una vez se han definido las topologías de red sobre las cuales queremos aplicar el algoritmo, añadimos al grafo el conjunto de nodos terminales (o usuarios) del grupo multicast. Estos nodos terminales se han conectado a los nodos de acceso mediante enlaces con velocidades bajas, pero prácticamente sin retardos. La idea es utilizar los nodos de acceso como si se trataran de operadores que dan acceso a Internet a un usuario que quiere entrar a formar parte de un determinado grupo multicast.

Una vez tenemos la red completa, pasaremos a crear nuestra red overlay, es decir, la red formada por los equipos de usuario del grupo multicast. Como ya se ha dicho, a pesar de que los algoritmos propuestos se pueden utilizar en cualquier nivel de la torre de protocolos, en este trabajo se ha estudiado el nivel de aplicación. La nueva red de usuarios se obtiene después de aplicar el algoritmo de Dijkstra al grafo inicial, resultando un grafo completo donde todos los nodos, que representan a todos los usuarios, están conectados entre sí. Además, cada uno de los nodos conocerá su distancia al resto de los nodos, medida en tiempo de transmisión más tiempo de propagación.

Estos retardos servirán para aplicar, sobre este grafo completo, los algoritmos de encaminamiento multicast. Conociendo, para cada miembro del grupo, los miembros más cercanos, podremos generar una tabla de encaminamiento para cada nodo, y lo que es más importante, saber el tiempo que tardaremos en enviar un paquete desde un nodo del grupo a todos los otros.

El trabajo se ha dividido en tres partes:

- La generación del grafo troncal para el modelo de Internet, siguiendo los modelos propuestos por Zegura, Calvert i Bhattacharjee [2]. Se han utilizado un conjunto de herramientas creadas para esta finalidad: el GT-ITM [7], del Georgia Institute of Technology, que genera grafos modelados siguiendo diferentes parámetros para conseguir una representación fiel de una red como Internet.

- Programación en lenguaje java de pequeñas aplicaciones para el tratamiento de los grafos. Por un lado, se ha escrito una aplicación que permite añadir aleatoriamente un número de usuarios (todos miembros de un mismo grupo multicast) a un grafo troncal. También se ha programado una aplicación que, dado un grafo que modela Internet con un conjunto multicast de terminales, aplica el algoritmo de Dijkstra y obtiene el grafo de la red overlay, donde podremos utilizar la última aplicación. Ésta ejecuta el algoritmo descrito en el presente TFC (del cual se han definido tres versiones, *cadencia fija, cadencia fija variable S* y *cadencia fija forzada,* que explicaremos en capítulos posteriores) sobre el grafo completo, dando como resultado las tablas de encaminamiento y los retardos de la transmisión multicast.

- Por último, se han hecho simulaciones con los diferentes modelos de algoritmos aplicados a distintas redes overlay, y se han obtenido los retardos finales de transmisión.

El primer capítulo de la memoria presenta el problema de encaminamiento multicast –explicando las limitaciones de los actuales métodos y presentando otras posibles soluciones– y el modelo de red que hemos utilizado para crear un grafo de simulación. El segundo es una breve introducción a la teoría de grafos, que describe además el algoritmo de Dijkstra que hemos utilizado para crear la red overlay de los miembros del grupo multicast. El tercer capítulo presenta de forma general la familia de algoritmos que hemos definido para el encaminamiento multicast, comenzando por los casos más sencillos (valores

constantes de los retardos y posibilidad de retransmitir indefinidamente un mismo paquete). En el cuarto, se describe la aplicación propiamente dicha de los algoritmos, así como las herramientas que se han utilizado. Y en el quinto capítulo se exponen los resultados obtenidos de la simulación, con las correspondientes conclusiones sobre la utilización de un modelo u otro.

1. TRANSMISIÓN MULTICAST Y MODELADO DE LA RED IP

1.1 El problema de la transmisión multicast

En todas las redes de comunicación existen diferentes modos de transmisión. Los más habituales suelen ser los de punto a punto, como el de la RTC (Red Telefónica Conmutada), y la transmisión broadcast, que consiste en enviar la información desde un sólo punto al resto, sin concretar los receptores, tal y como sucede en la comunicación de una emisora de radio, por ejemplo. Pero si lo que queremos es una comunicación desde un punto hasta un conjunto determinado de receptores, la cosa se complica considerablemente. Este tipo de comunicación recibe el nombre de transmisión multicast.

Por lo que se refiere a la red de Internet, que es lo que nos interesa, hace tiempo que existe el encaminamiento IP multicast. Además, cada vez hay más aplicaciones que aprovechan las características de una transmisión de este tipo: videoconferencia, comunicaciones corporativas, enseñanza a distancia, televisión por Internet, etc. Se ha de decir que el diseño inicial del direccionamiento IP no tuvo en cuenta la comunicación multicast, y se podría decir que este tipo de direccionamiento es un “añadido” al protocolo IPv4. De hecho, encontramos inconvenientes importantes para utilizar este encaminamiento. Por ejemplo, el rango de direcciones de que disponemos para hacer grupos multicast es muy limitado, y la mayoría ya están reservados. Igualmente necesitamos utilizar algoritmos complejos (como el PIM) para realizar el direccionamiento multicast a la red IP, y lo que es más importante, necesitamos que los elementos de la red (como por ejemplo los routers) entiendan estos protocolos.

Por todo esto, la comunicación multicast a nivel de red, es bastante complicada y difícilmente acabará de extenderse antes de que aparezca el protocolo IPv6. También podemos ver, en este caso, que el grupo multicast se crea justamente en el nivel 3 de la torre de protocolos (el nivel de red), de forma que una aplicación que necesite definir y gestionar un grupo multicast (como por ejemplo una videoconferencia entre varios usuarios) encontrará graves dificultades para tener el control absoluto del grupo multicast.

Por este motivo, en el presente proyecto se describen nuevos algoritmos para el direccionamiento multicast que pueden ser aplicados en el nivel de aplicación. Aunque, estrictamente, no será una comunicación multicast (ya que si no utilizamos las propiedades multicast del protocolo de red, tendrán que enviarse tantos paquetes como miembros formen el grupo), podemos intentar encontrar una manera inteligente de hacer el reenvío de la información hacia los nodos terminales.

1.2 El modelo del cartero

Una manera posible de hacer estos reenvíos “inteligentes” de la información consiste en aprovechar el retardo que sufre la información para llegar hasta su destino. Un símil lo podríamos encontrar en el siguiente ejemplo.

Imaginemos que una persona quiere dar una noticia a otras dos. El emisor puede coger el teléfono y llamar a la primera persona, explicarle lo que le ha de explicar, y colgar. A continuación, debería repetir el mismo proceso para la otra persona. Así, el emisor envía la información al primer destinatario, y cuando éste la ha recibido completamente envía la información al segundo, de manera que mientras está enviando la información el emisor no puede hacer nada más (es decir, ha de hacer las dos llamadas, pero no puede iniciar la segunda hasta haber concluido la primera). Si el emisor quiere dar la noticia a las mismas personas, pero ahora por carta, se encontrará con el hecho de que también tendrá que escribir dos, pero con la diferencia de que ahora puede escribir la primera, echarla al buzón, y mientras el cartero se encarga de repartirla puede escribir la carta del segundo destinatario. Igual que en el caso anterior, el emisor tendrá que enviar dos paquetes, pero en este segundo caso, se aprovecha el tiempo que tarda la información en llegar a un destinatario para preparar el mensaje hacia otro. En un hipotético caso en que el servicio de correos fuera muy eficiente, de manera que una carta echada al buzón se pudiera dar a su destinatario en tiempo muy corto, y al mismo tiempo, el servicio de llamadas telefónicas fuera muy lento, se podría conseguir que la información llegara a la totalidad de los receptores mas rápidamente por carta que por llamada telefónica.

Esta es la idea básica del modelo propuesto por Amor Bar-Noy y Shiomo Klipnis [1], de manera que la comunicación entre dos elementos (nodos en nuestro caso) no implique que los dos hayan de estar ocupados, sino que se aprovecha el retardo de la transmisión para ir haciendo más envíos y para permitir a aquellos elementos que ya han recibido la información retransmitirla a otros nodos. El modelo presentado utiliza un parámetro λ de latencia, mediante el cual se representa el tiempo que tarda el mensaje en llegar desde un nodo hasta otro, expresado en términos de “tiempos de transmisión”; es decir, tomando como unidad el tiempo de transmisión de la información, que sería el tiempo que tarda un nodo en poner la información en la red (en este modelo simplificado se supone que todos los nodos tienen el mismo tiempo de transmisión y todos los enlaces el mismo retardo de propagación). Este modelo propuesto busca árboles de encaminamiento óptimos basándose en árboles de Fibonacci, y ofrece una manera óptima de transmitir un mensaje hacia un conjunto de nodos. Fundamentalmente, cada vez que un nodo recibe el paquete empieza a retransmitirlo a los siguientes nodos hasta que el paquete ha llegado a todos los destinatarios. A pesar de que en el comentado artículo el algoritmo se define para una transmisión broadcast, se puede entender fácilmente el caso de multicast, escogiendo un grupo de nodos y haciendo una transmisión broadcast hacia todos ellos.

En el siguiente diagrama, vemos un ejemplo de transmisión de un mensaje desde el nodo P_o hacia 7 nodos, siguiendo el modelo propuesto de árbol de

Fibonacci. En este caso tenemos que un nodo puede transmitir el paquete cada unidad de tiempo (que ya hemos visto que es la unidad básica de la latencia), y el valor de λ es 2 (de forma que necesitamos dos unidades de t para ir de un nodo a otro).

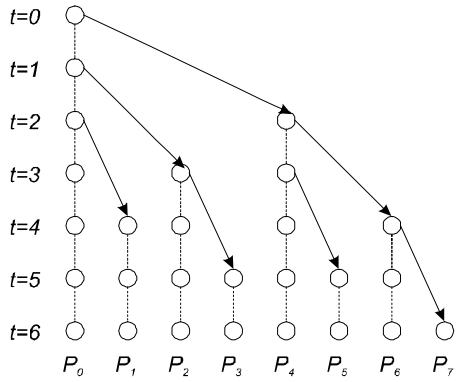


Fig. 1.1 Árbol de transmisión

Como se ha propuesto, el modelo es aplicable a cualquier tipo de red, ya sea entre ordenadores en una red IP o entre diferentes procesadores conectados entre sí. De todas maneras, el modelo tiene el inconveniente que siempre se basa en el valor de la latencia λ, medido en unidades de tiempo de transmisión. Si lo que queremos hacer es una transmisión sobre una red como Internet, muy heterogénea, los retardos de los enlaces y el tiempo de transmisión pueden variar mucho de un nodo a otro nodo. Además, una de las principales aplicaciones donde se puede utilizar el encaminamiento multicast es la transmisión de audio o video por Internet, o streaming. En un caso como este, no sólo hemos de tratar de minimizar el retardo de transmisión, sino que además hemos de tener en cuenta parámetros como la cadencia de la información o el riesgo de congestión de la red. Por estas razones, en el presente trabajo hemos diseñado un algoritmo que, si bien se basa en la mismas premisas (aprovechar que se puede seguir enviando información mientras un paquete viaja por la red, y que los nodos que ya lo tienen pueden retransmitirlo a usuarios que aun no les haya llegado), presenta ciertas características que lo hacen útil sobre una red real como Internet.

1.3 Modelado de la red IP

El crecimiento que han sufrido las redes de ordenadores, y en particular la red IP, han aumentado los problemas relacionados con el encaminamiento, la congestión y la administración de estas redes. El estudio de algoritmos de encaminamiento, como en el caso que tratamos en este trabajo, no puede realizarse sobre redes reales por dos razones principales:

- Las redes suficientemente grandes para dar resultados significativos son caras y difíciles de controlar.
- Dichas redes, además, no acostumbran a estar disponibles para experimentos.

Por esta razón, es habitual probar las posibles soluciones mediante modelos que de algún modo sean una “buena abstracción” de las redes reales.

Si echamos un vistazo a los diferentes modelos que se han utilizado para modelar redes de ordenadores, encontraremos que normalmente son:

- Topologías regulares, como anillos, árboles y topologías en estrella.
- Topologías “bien conocidas”, como ARPAnet o la troncal NSFnet.
- Topologías generadas aleatoriamente.

Las limitaciones de estas redes son obvias. Las topologías regulares o las topologías “bien conocidas” sólo representan una parte de las redes actuales. Igualmente, las topologías generadas aleatoriamente no suelen representar de manera real ninguna red, ni antigua, ni presente ni futura. Esto es un problema, ya que el resultado de un algoritmo aplicado sobre un tipo u otro de topología puede ser completamente diferente.

A continuación se presenta un posible modelo de Internet en forma de grafo. Para hacerlo, se ha seguido el modelo propuesto por Ellen W. Zegura, Kenneth L. Calvert y Samrat Bhattacharjee [2].

1.3.1 Modelado de redes reales

El objetivo del trabajo de Zegura, Calvert y Bhattacharjee consiste en representar de una manera suficientemente real los caminos (es decir, las secuencias de nodos) por los cuales viaja la información en una transmisión entre dos nodos cualesquiera de una red IP. Los nodos representan routers y las aristas los caminos entre los elementos de interconexión. Su modelo no tiene en cuenta los equipos terminales, sino que se limita a la estructura troncal de la red. Además, se pueden aplicar diferentes criterios para crear el modelo de la red. Por ejemplo, en algunos casos puede resultar interesante estudiar el rendimiento de un algoritmo aplicado en una red de grandes dimensiones, donde únicamente conocemos una parte de la estructura. En este caso, el modelo reflejará las características de esta parte de la red. También nos será útil para modelar una red corporativa o universitaria, ya que se trata de modelos estáticos.

Por lo tanto, como la mejor manera de modelar la topología de una red (o subred) IP depende del uso que se le vaya a dar, la creación de estos modelos se basa en diferentes atributos topológicos. El modelo contiene tanto métricas hop-based, en las que los enlaces tienen un peso de una unidad, como métricas length-based, en donde el peso de una arista depende de su longitud real. Para un grafo con n nodos y m aristas, consideramos los siguientes atributos:

- Grado medio de los nodos: determina el valor medio del grado de los nodos, es decir, el número medio de conexiones de cada nodo.
- Distribución de la profundidad de salto (hop-depth) o diámetro del grafo, es decir, el número de saltos máximos necesarios para viajar entre los dos nodos más alejados del grafo.

- Distribución de la profundidad de distancia (length-depth): equivalente al anterior, pero utilizando la métrica de la distancia euclidiana.

Se han seleccionado estas características por dos razones:

- son abstracciones cuantitativas de ciertas características de la red.
- son relativamente sencillas de calcular.

Recalcamos que estos parámetros no son independientes, de manera que incrementar el grado medio del nodo puede comportar, por ejemplo, una reducción del diámetro de la red.

En la tabla 1.1 podemos ver los atributos de alguna red real de Internet. Es difícil conseguir información completa sobre la topología de la red IP entera, o incluso de ciertos fragmentos, ya que no hay ningún tipo de administración centralizada. Pero sí que hay alguna red real de la cual se ha podido obtener la información, algunas de las cuales estarán representadas en la tabla. ARPAnet corresponde a la red troncal de la antigua ARPAnet, la topología de la cual se ha utilizado en muchas simulaciones. También tenemos CERFnet, una red troncal regional donde también se tienen en cuenta los enlaces hacia otros dominios de encaminamiento. La PGE es una gran red privada, con un considerable número de nodos "hoja" (o nodos terminales de grado uno). Por ultimo, la red Campus corresponde a la red del instituto Tecnológico de Georgia, lugar de origen de este modelo.

Tabla 1.1. Atributos de redes reales

Red	Nodos	Grado Medio	Diámetro	Bicomponentes
ARPAnet	47	2.9	9	5
CERFnet	63	2.1	5	59
PGE	137	2.2	10	103
Campus	30	3.9	3	16

1.3.2 Modelación de cualquier tipo de red

1.3.2.1 Grafos planos aleatorios

Todos los tipos de generación de grafos son variaciones del modelo estándar de grafo aleatorio, que distribuye los vértices en diferentes posiciones por el plano de manera aleatoria, y después, añade una arista entre un par de nodos con probabilidad α. A este tipo de generación de grafos le llamaremos el modelo *aleatorio puro*. Como se ha comentado, estos grafos no representan la estructura de las redes IP reales, pero su simplicidad hace que se utilice en el estudio de problemas de redes.

El resto de modelos aleatorios también distribuyen los vértices de manera aleatoria por el plano, pero alteran la función de probabilidad de las aristas, con el objeto de obtener una mejor aproximación de la estructura real de la red. Después del modelo aleatorio puro, el más utilizado es el modelo de Waxman.

En este caso, la probabilidad que haya una arista entre *u* y *v* viene dada por la expresión:

$$P(u,v) = \alpha e^{-d/(\beta L)}$$

donde *0<α, β≤1*, *d* es la distancia euclidiana entre *u* y *v*, y *L* es la distancia máxima entre dos nodos cualesquiera de la red. Se han definido pequeñas variaciones de este modelo de Waxman, como el de sustituir el valor de *d* por un número aleatorio entre *0* y *L*, o permitir que *α>1*.

Otro modelo que también relaciona la probabilidad de existencia de una arista con la distancia entre vértices, es el modelo exponencial. En este caso, la probabilidad de que haya una arista entre *u* y *v* es:

$$P(u,v) = \alpha e^{-d/(L-d)}$$

que también es inversamente proporcional a la distancia entre los dos nodos. Existe, por último, el modelo de localización, que divide los nodos en diferentes categorías, asignando a cada una la probabilidad de enlace. Por ejemplo, para dos categorías, la probabilidad de una arista entre *u* y *v*, viene dada por:

$$P(u,v) = \begin{cases} \alpha & si\ d < r \\ \beta & si\ d \geq r \end{cases}$$

donde *r* es un parámetro utilizado para definir los límites de la categoría. En la tabla 1.2 podemos ver un resumen de estos modelos de grafos aleatorios.

Tabla 1.2 Modelos de grafos aleatorios

Modelo	Probabilidad de arista
Aleatorio puro	α
Waxman	$P(u,v) = \alpha e^{-d/(\beta L)}$
Exponencial	$P(u,v) = \alpha e^{-d/(L-d)}$
Localización	$P(u,v) = \begin{cases} \alpha & si\ d < r \\ \beta & si\ d \geq r \end{cases}$

1.3.2.2 Modelos jerárquicos

Los modelos aleatorios y regulares que hemos visto hasta ahora representan extremos de las topologías, en el sentido de que los modelos aleatorios permiten poco control sobre la topología resultante, mientras que los modelos regulares crean topologías muy rígidas. Ninguno de estos modelos copia la estructura jerárquica que podemos encontrar en las redes reales de unas dimensiones considerables, a pesar de que ambos son buenas representaciones de ciertos tipos de redes que podríamos llamar “locales”.

Ahora veremos dos métodos para crear topologías jerárquicas, conectando entre sí diferentes componentes siguiendo una estructura determinada.

Nivel N

El modelo jerárquico de nivel N construye una topología de manera recursiva. Partiendo de un grafo conexo, en cada paso de la recursividad los nodos son sustituidos por un nuevo grafo conexo. El problema de la conectividad de las aristas del grafo original se soluciona con diferentes procedimientos, siendo el más sencillo escoger aleatoriamente un nodo del nuevo grafo y hacer que la arista original incida en él.

Este modelo se genera dividiendo el plano donde se encuentra el grafo en una cuadrícula, y donde hay un nodo se genera un nuevo grafo conexo que esté dentro de las dimensiones del cuadrado que lo contiene. Entonces, se subdivide este cuadrado en una nueva cuadrícula, y se va reiterando el proceso. Esto permite tener una estructura de niveles jerárquicos (tantas como iteraciones hagamos), donde podemos definir atributos como el coste de los enlaces de los diferentes niveles (por ejemplo, haciendo que las aristas del nivel más alto sean enlaces troncales, con un ancho de banda superior al resto), y también permite definir dominios. La figura 1.2 es un ejemplo del modelo jerárquico de nivel N, con $N = 3$, donde podemos ver los nodos y aristas que forman el nivel superior, dos grafos que se situarían en el segundo nivel y, finalmente, cuatro nodos que formarían un grafo de tercer nivel.

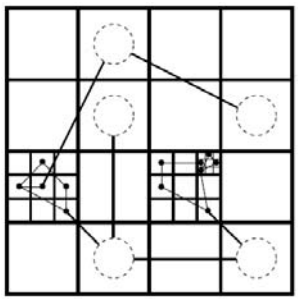


Fig.1.2 Modelo de jerarquía de nivel N

Transit-Stub

Existe otra posibilidad de producir modelos jerárquicos: el modelo Transit-Stub (“Transit-Access”). Primero construimos un grafo conexo aleatorio, donde cada nodo representa un dominio de tránsito completo. Entonces, cada nodo es sustituido por un nuevo grafo conexo, que representa la topología troncal de este dominio. Seguidamente, para cada nodo en cada dominio de tránsito, generamos un número de grafos que representan los dominios de stub, llamados también de *acceso*, añadidos a aquel nodo. Por ultimo, se añaden algunas aristas entre nodos de un dominio de tránsito y de un dominio stub, o entre dos dominios stub diferentes. Si todos los grafos que se han generado son conexos, el grafo que resulta será un grafo conexo.

Los parámetros que se utilizan para crear estos modelos son los siguientes:

- T: indica el número de dominios de tránsito.
- N_i: número medio de nodos de un dominio de tránsito
- K: número medio de dominios stub por nodo de tránsito
- N_s: número medio de nodos por dominio stub

Estos parámetros son valores medios, de manera el número de nodos que encontramos en cada dominio, ya sea de tránsito o stub, o el número de dominios stub que hay en dominio de tránsito, es un valor aleatorio. En la figura 1.3 tenemos un ejemplo de la estructura de un modelo Transit-Stub.

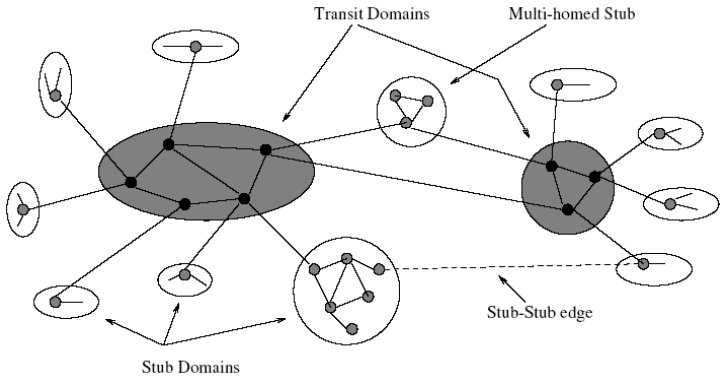


Fig.1.3 Estructura del modelo Transit-Stub

1.3.3 El modelo utilizado

Para poder hacer nuestra simulación, hemos utilizado el modelo Transit-Stub. Hemos escogido este modelo porque siendo relativamente simple, se parece a la topología de Internet que necesitamos. El grafo troncal que hemos utilizado está formado por 100 nodos y 4 dominios de tránsito de uno o dos nodos que actúan como los grandes routers que comunican con el resto de redes. El resto de nodos forman diferentes redes stub, que representan redes de acceso a Internet (como por ejemplo las redes corporativas, la red de un campus universitario, o las redes de los operadores que actúan de ISP's).

De todas maneras, el modelo que podemos obtener no es suficiente para nuestro estudio ya que, como hemos visto anteriormente, no se tienen en cuenta los usuarios individuales, o dicho de otra manera, lo que serían los equipos de usuario. Por tanto, además de tener la red troncal (con nodos de tránsito, *T*, y con nodos de acceso, *S*) necesitaremos los nodos terminales o de usuario que llamamos *U* y que actuarán como miembros del grupo multicast entre los cuales se quiere enviar cierta información. Estos nodos de usuario se añadirán mediante una aplicación en Java, que conectará de manera aleatoria los nodos de usuario del grupo multicast a uno de los nodos stub que tendremos repartidos por el grafo. De hecho, la interpretación que hacemos de estos nodos *U* es la de un ordenador que, mediante una línea de acceso

(ADSL, PPP, Ethernet, etc.), se conecta punto a punto con otro ordenador que le da acceso a Internet (un ISP, que viene representado por un nodo *S*), todo ello para integrarse en la *red overlay* formada por los miembros de su mismo grupo.

2. TEORIA DE GRAFOS

En este capitulo se dan algunas definiciones básicas de la teoría de grafos y se presenta el algoritmo de Dijkstra. Con estas herramientas podremos crear el modelo de Internet y obtener los caminos más cortos entre los nodos terminales que forman parte de un mismo grupo multicast.

2.1 Definiciones

Un grafo simple *G* es un par *(V(G), E(G))* de manera que *V(G)* es un conjunto finito de elementos llamados nodos o vértices, y *E(G)* es un conjunto finito de pares no ordenados de nodos, llamados aristas. El orden *n* de un grafo es el número de nodos. Definimos el tamaño *E* de un grafo como el número de aristas. Normalmente, un grafo se representa gráficamente dibujando los nodos como puntos, y las aristas como una línea que une los nodos.

Tanto los nodos como las aristas pueden tener aplicaciones $\Phi: V(G) \rightarrow Z$ y $\Phi': E(G) \rightarrow Z$ de modo que cada nodo y/o arista tenga asociado un número o etiqueta. Estas etiquetas pueden servir para identificar los elementos del grafo, o (en nuestro caso) indicar la velocidad de un enlace o el tipo de nodo.

Según la definición, un grafo simple no puede tener aristas repetidas, es decir, pares de nodos unidos por más de una arista. Diremos que dos nodos *u* y *v* son adyacentes cuando están unidos por una arista *uv*. En este caso, diremos que los nodos *u* y *v* inciden sobre la arista *uv*, o bien, que la arista *uv* incide sobre los nodos *u* y *v*. Dos aristas son adyacentes cuando tienen un nodo en común. El grado $\delta(v)$ de un nodo *v* es el número de aristas incidentes en *v*.

Dado un grafo simple de orden *n*, el número máximo de aristas que puede tener es igual a $n(n-1)/2$. La densidad $\rho(G)$ de un grafo *G* de orden *n* se define como el cociente entre el número de aristas del grafo *G* y el máximo número de aristas que puede tener el grafo de orden *n*, obteniendo la expresión:

$$\rho(G) = \frac{E}{n(n-1)/2} = \frac{2E}{n(n-1)}$$

Una secuencia de aristas es una sucesión de aristas consecutivas v_0v_1, $v_1v_2, v_2v_3, .., v_{m-1}v_m$. Esta secuencia dibuja un camino continuo sobre el grafo. Una secuencia donde no se repitan aristas se llama *cola*, y si tampoco se repite ningún nodo, *trayecto*. Un trayecto cerrado, de forma que el nodo inicial y el final coincidan se llama *circuito*.

Decimos que un grafo *G* es conexo cuando podemos trazar, como mínimo, un trayecto entre dos nodos cualesquiera de *V(G)*. Definimos la distancia *d(u,v)* entre dos nodos *u* y *v* como el cardinal mínimo de todos los trayectos entre *u* y *v*, es decir, el menor número de aristas por las cuales hemos de pasar para llegar de un nodo a otro. Si un grafo no es conexo, y dos nodos se encuentran

en componentes conexos diferentes, se considera que su distancia es infinita. Finalmente, se define el diámetro de un grafo *G* como la distancia máxima entre dos nodos cualesquiera de *V(G)*.

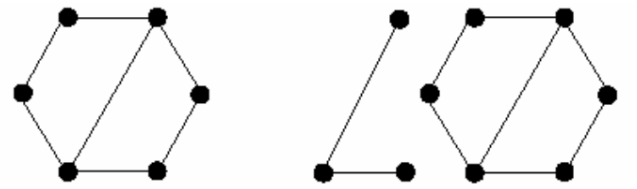


Fig.2.1 Ejemplos de grafo conexo y grafo no conexo

Un grafo conexo sin ningún circuito es un *árbol*. Un grafo completo K_n es un grafo simple de tal forma que todos sus pares de nodos están unidos por una arista, de manera que su tamaño es *n(n-1)/2*. Por otro lado, si todos los nodos de un grafo tienen el mismo grado diremos que se trata de una grafo regular. En concreto, si todos los nodos del grafo tienen grado *r,* diremos que el grafo es regular de grado r.

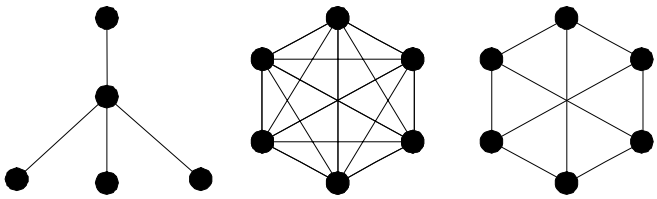


Fig.2.2 Ejemplo de árbol, grafo completo y grafo regular

Un digrafo *G* se define como un par *(V(G), A(G)),* donde *V(G)* es un conjunto finito no vacío de elementos, llamados nodos, y *A(G)* es una familia finita de pares ordenados de elementos de *V(G)*, llamados arcos o aristas dirigidas (nosotros las llamamos aristas). Un arco de *v* a *w*, es un arco el primer elemento del cual es *v* y el segundo, w. Haremos notar que el arco *vw* y el *wv* son dos arcos diferentes. Si *G* no tiene ningún arco de un nodo a sí mismo y todos los arcos de *G* son diferentes, diremos que *G* es un digrafo simple. Si *G* es un digrafo, el grafo que obtenemos de eliminar “las flechas” de los arcos se llama grafo base de *G*.

Todas las definiciones dadas anteriormente para un grafo simple se pueden extender a los digrafos. Así, podemos definir aplicaciones de etiquetaje $\Phi:V(G) \rightarrow Z$ y $\Phi':A(G) \rightarrow Z$, definir términos de adyacencia entre nodos e incidencias de arcos a nodos, o también secuencias finitas de arcos $v_0v_1, v_1v_2, .., v_{m-1}v_m$ donde no se repiten ni arcos ni nodos.

Un digrafo *G* es conexo si el grafo base de *G* es un grafo conexo. Si, además, existe un trayecto de *v* a *w* para cualquier par de nodos $v,w \in V(G)$ diremos

que el grafo *G* es fuertemente conexo. Si bien todo digrafo fuertemente conexo es conexo, no todo digrafo conexo es fuertemente conexo.

2.2 Búsqueda de caminos mínimos. El algoritmo de Dijkstra.

En este proyecto se estudia la aplicación de ciertos algoritmos para la transmisión de información. Más adelante veremos con detalle cómo funcionan estos algoritmos, pero ahora podemos avanzar que su funcionamiento parte del hecho de que los ordenadores de un mismo grupo se han de reenviar la información de manera "inteligente" entre ellos.

Para esto, es necesario conocer los miembros del grupo que están más cerca de un determinado usuario, así como el coste de la transmisión. Este problema se puede plantear como un simple caso de búsqueda de caminos mínimos, y se puede resolver óptimamente utilizando el algoritmo de Dijkstra.

En este algoritmo, cada nodo *v* del grafo *G(V, E)* tiene una etiqueta asociada *L(v)*. Esta etiqueta indica la menor distancia conocida para ir desde un nodo fijado *u* hasta este nodo. Inicialmente, el valor de *L(v)* corresponde al peso *w(u,v)* de la arista que une los nodos *u* y v, en el caso de que esta arista exista, siendo $L(v) = \infty$ en caso contrario. El algoritmo funciona creando un conjunto de nodos $T \subset V$, para los cuales se han obtenido hasta ese momento el camino más corto desde *u* hasta ellos. Al final del algoritmo, *L(v)* contiene el coste del camino mas corto para ir desde *u* hasta v.

En cada iteración, el algoritmo añade un nuevo nodo en la lista *T*. Esto se consigue escogiendo un nodo *v'* que todavía no pertenezca a *T* y que tenga una etiqueta *L(v´)* mínima. En otras palabras, escogemos el nodo *v'* más cercano a *u* y externo a la lista *T*. Una vez hecho esto, se actualizan las etiquetas de los nodos sobre los que incide *v'*, de manera que hacemos un nuevo cálculo de las distancias de *u* a estos nodos y añadimos este nodo *v'* a *T*. El proceso se repite hasta que todos los nodos del grafo se encuentran en la lista.

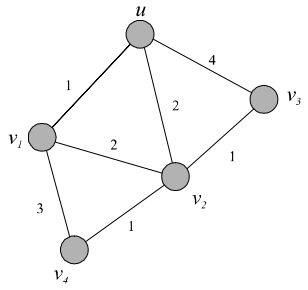
1. **para todo** $v \neq u$ $L(v) = w(u,v)$
2. $L(u) = 0$
3. $T = \{u\}$
4. **mientras** $T \neq V$
   Inicio
5. encuentra $v' \notin T$ de forma que $\forall\ v \notin T$ $L(v') \leq L(v)$
6. $T = T \cup \{v'\}$
7. **para todo** $v \notin T$ de forma que *v'* es adyacente a *v*
   si $L(v) > L(v') + w(v',v)$
   entonces $L(v) = L(v') + w(v',v)$ **fin si**
   fin para a
   fin mientras

Fig. 2.3 Algoritmo de Dijkstra

Se puede demostrar que este algoritmo es óptimo. Para hacerlo, probaremos que cada vez que un nodo *v'* se añade a *T*, la etiqueta *L(v')* indica la distancia mínima de *u* a *v'*. En efecto, supongamos que esto no es así, y que existe un camino mas corto de *u* a *v'*. Sea w_2 el primer nodo adyacente a *u* por el cual pasa este nuevo camino de *u* a *v'* de distancia menor a *L(v')*. Este nodo w_2, por construcción, deberá pertenecer a *T*, ya que la distancia de *u* a w_2, que el algoritmo conoce desde la primera iteración, ha de ser menor que *L(v´)*. El mismo argumento se puede repetir para el siguiente nodo del camino, w_3, que tendrá una distancia desde *u* mas pequeña que *L(v')* calculada al añadirse w_2 a *T* y que por tanto se habrá añadido igualmente a *T*, y así sucesivamente hasta llegar a *v'*. De aquí se deduce que, si existiera un camino más corto de *u* a *v'*, el algoritmo debería ir añadiendo en *T* los nodos w_2, w_3, w_4…que forman este camino más corto, y por tanto lo debería encontrar. Por último, como al acabar el algoritmo todos los nodos se encuentran en la lista *T*, el algoritmo de Dijkstra encuentra el camino más corto desde un nodo *u* hasta cualquier otro nodo del grafo.

También es fácil probar que este algoritmo requiere un tiempo de cálculo de orden $O(n^2)$, cosa que en la práctica significa que se pueden encontrar los caminos óptimos en tiempos de computación reducidos. Para determinar el mínimo *L(v')* (línea 5 del pseudocódigo del algoritmo) necesitamos *O(n)* comparaciones, y la línea 7 no necesita mas de *n* asignaciones. Estas dos líneas se encuentran en la sentencia *mientras,* de la línea 4, que se ejecuta *(n-1)* veces. Por tanto, el algoritmo completo se puede ejecutar en un tiempo de orden $O(n^2)$.

Para acabar, hemos de añadir que este algoritmo, además de calcular el camino más corto entre dos nodos, determina la ruta que forma el camino óptimo. Esto se consigue añadiendo una nueva etiqueta en cada nodo, de manera que cada vez que se actualiza el valor de *L(v')* de un determinado nodo *v'* se añadirá el nodo *v* a partir del cual se ha calculado su valor *L(v')*. De esta manera, el algoritmo de Dijkstra no dará únicamente el coste de la ruta mínima para ir entre dos nodos, sino que también obtendremos la sucesión de nodos que conducirán de manera óptima desde *u* hasta cualquier nodo del grafo.



Iteración	*v'*	*L(u)*	$L(v_1)$	$L(v_2)$	$L(v_3)$	$L(v_4)$	*T*
0	-	0	1	2	4	∞	*{u}*
1	v_1	0	1	2	4	4	*{u, v_1}*
2	v_2	0	1	2	3	3	*{u, v_1, v_2}*
3	v_3	0	1	2	3	3	*{u, v_1, v_2, v_3}*
4	v_4	0	1	2	3	3	*{u, v_1, v_2, v_3, v_4}*

Fig.2.4 Ejemplo de aplicación del algoritmo de Dijkstra desde el nodo u

3. ALGORITMO MULTICAST

En los capítulos anteriores se ha hablado de los problemas de una transmisión multicast, y de una posible solución del problema. En este capítulo se presenta una familia de algoritmos de punto a multipunto capaces de funcionar en diferentes niveles de la torre de protocolos, fáciles de implementar, escalables y con unos retrasos de transmisión, como se verá más adelante, pequeños.

3.1 El modelo inicial para un sólo paquete

El planteamiento básico del algoritmo es el siguiente: dado un nodo origen (también llamado raíz) y un conjunto de nodos destinos (que actúan como miembros del grupo multicast a los cuales se pretende enviar la misma información), el algoritmo ha de encontrar las rutas óptimas que minimicen el retardo total de la transmisión de un paquete. En el caso de un único paquete, el nodo raíz envía el paquete al nodo más cercano, a continuación lo envía al segundo nodo más cercano y así sucesivamente. Cuando un nodo recibe el paquete repite el mismo procedimiento: retransmite el paquete al nodo más cercano, después al segundo más cercano, etc., siempre y cuando estos nodos no hayan recibido el paquete por otro camino.

Para saber si un nodo ya ha recibido el paquete no hace falta hacer ninguna consulta, ya que todos los nodos saben lo que hacen los demás –suponemos que primero ha habido una inundación y todos los nodos conocen el estado de toda la red– y por lo tanto sabrán si un nodo al que quieren enviar el paquete ya lo ha recibido por otro lado. Como el algoritmo se define de manera constructiva, se puede demostrar que es óptimo repitiendo los argumentos de la prueba de Dijkstra que hemos visto en el capítulo 2.

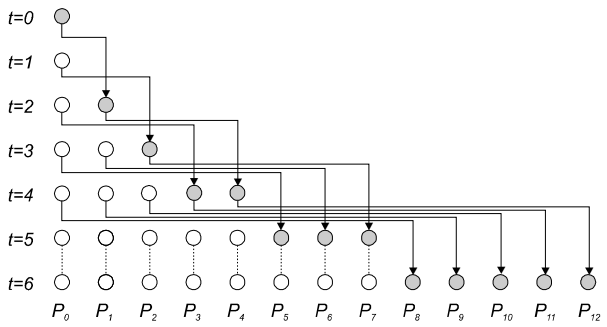


Fig. 3.1 Esquema de funcionamiento del algoritmo

En este caso, además, se podrían calcular analíticamente los tiempos de retardo total cuando todos los enlaces de la red tienen el mismo retardo, suponiendo que tenemos además un tiempo de transmisión constante. Debido a la complejidad del modelo matemático descrito, sólo se podría determinar una solución cerrada para el caso particular en el que el retardo total de un enlace fuera el doble del tiempo de transmisión de un paquete. Para el resto de casos se podría repetir el mismo procedimiento, pero sin darse una solución

cerrada –ya que los cálculos serian muy extensos– sino una fórmula matricial sencilla que se puede programar sin dificultades.

En el caso más realista de una red heterogénea, en la cual las velocidades de transmisión y los retardos de los enlaces puedan ser diferentes, podemos obtener valores del retardo a partir de las expresiones del caso anterior, suponiendo que todos los enlaces son idénticos y que su tiempo de propagación y transmisión son los mayores de toda la red.

3.2 Un modelo sencillo para enviar M paquetes

El algoritmo que hemos definido para el caso de un único paquete puede ser muy útil para transmitir poco volumen de información entre todos los miembros del grupo. Pero para las transmisiones multicast más habituales, como por ejemplo el *streaming*, este volumen de información es normalmente muy superior a un único paquete de datos.

La primera solución planteada para enviar más de un paquete es repetir sucesivamente la tabla de encaminamiento obtenida para enviar un sólo paquete, como si cada paquete de información fuera único e independiente. En otras palabras, se cogería el primer paquete de la información y se enviaría a todos los miembros del grupo, y una vez hubiese llegado a todos se comenzaría a transmitir el segundo, y así sucesivamente. En este caso, el retardo total de la transmisión sería el retardo de un sólo paquete multiplicado por el número total de paquetes que conforman la información. Pero este procedimiento, que es muy sencillo, tiene el inconveniente de que el nodo raíz debería ir transmitiendo el primer paquete hasta que éste llegase a todos los nodos, con tal de minimizar el retardo total de la transmisión. Con esto, tenemos que el nodo origen no podría comenzar a enviar el segundo paquete hasta que todos los miembros del grupo hubieran recibido el primero, e igualmente con el resto.

Es por esto que se ha planteado una nueva posibilidad: que el nodo origen deje de transmitir el primer paquete cuando tenga preparado el segundo, aunque el primero no haya llegado todavía a todos los nodos. Con esto, es evidente que un paquete tardará más en llegar a todos los nodos, pero comenzaremos antes a enviar el siguiente paquete. Este ahorro de tiempo en el intervalo que separa el inicio de transmisión entre dos paquetes consecutivos se irá acumulando progresivamente, de manera que, si el número de paquetes es suficientemente grande, puede llegar a compensar el aumento del retardo de la transmisión de un paquete.

3.3 El retardo Stream Multicast

Como hemos dicho en el apartado anterior, lo que haremos es permitir que el nodo raíz (también conocido como *root*) pueda comenzar a transmitir el segundo paquete, aunque el primero no haya llegado a todos los nodos. Como dijimos en capítulos anteriores, lo que pretendemos es limitar la cadencia del

root a través del parámetro *S* (número de paquetes que envía cada nodo). Consideraremos *M* como el número de paquetes del stream y μ_r el tiempo de la transmisión de la fuente, y $t_s[0]$ como el retado de un paquete. Asumimos que nuestra red tiene conectividad total, y que *S* es lo suficientemente grande como para que los paquetes lleguen a todos los nodos de la red. En este caso, el retardo del streaming (T_{Ms}) es:

$$T_{Ms} = (M - 1) \cdot S \cdot \mu_r + t_s[0] \qquad (1)$$

Es decir, la raíz envía el primer paquete *S* veces, y el segundo paquete comenzará a transmitirse después de $S \cdot \mu_r$ segundos y así sucesivamente. En el momento $(M - 1) \cdot S \cdot \mu_r$ el root terminará de enviar el $(M - 1)$-ésimo paquete y empezará con el último paquete que llegará al último nodo después de $t_s[0]$. Puede demostrarse que si la cadencia de todos los nodos es superior a la del nodo raíz, no habrá que guardar paquetes en las colas de espera y el retardo $t_s[0]$ para el último paquete será igual al retardo para cualquier otro paquete. De este modo, consideraremos $t_s[0]$ como el retardo de tiempo del primer paquete.

La ecuación anterior sólo es válida cuando la raíz envía cada paquete *S* veces. Cuando *S* es grande el mensaje puede ser recibido por todos los nodos antes de que el root haya enviado el paquete *S* veces. Aunque en este caso el nodo podría permanecer parado y se podría esperar hasta $S \cdot \mu_r$ y entonces empezar a enviar el segundo mensaje, esto significaría una pérdida de eficiencia. Por lo tanto, cuando el mensaje se recibe en todos los nodos antes que el nodo raíz lo haya enviado *S* veces, permitiremos al nodo raíz enviar el segundo mensaje inmediatamente, sin un intervalo de silencio. En este caso particular el parámetro *S* de la ecuación anterior debe reemplazarse por el número $S_r(s)$ de veces que el nodo raíz ha enviado realmente el paquete, en donde $S_r(s) \leq S$

$$T_{Ms} = (M - 1) \cdot Sr(s) \cdot \mu_r + t_s[0]$$

4. APLICACIÓN DEL ALGORITMO

4.1 Procedimiento para simular el algoritmo

Para probar el comportamiento del algoritmo se ha realizado una serie de simulaciones sobre el modelo de red de Internet descrito en el primer capítulo. Con este objetivo, se han creado una serie de programas en Java que, partiendo de un grafo troncal siguiendo el modelo de dominios *Transit-Stub*, genera la red overlay correspondiente y aplica el algoritmo con diferentes parámetros.

4.2 Obtención del grafo troncal

Ya se ha comentado que una posible aplicación del algoritmo que estamos estudiando se encontraría en la transmisión multimedia por Internet, o en las redes P2P que miles de usuarios utilizan diariamente. Por lo tanto, la simulación deberá realizarse sobre un modelo de red que se aproxime, en la medida de lo posible, a la complejidad de la red IP.

En el primer capítulo de esta memoria hemos visto diversas propuestas para representar de forma realista la red de Internet. El modelo que hemos utilizado es el de dominios Transit-Stub, que describe grafos con nodos de tránsito y nodos de acceso con distribuciones aleatorias. Estos grafos troncales los hemos conseguido mediante las herramientas del Instituto Tecnológico de Georgia, un conjunto de aplicaciones que permite generar grafos de diferentes topologías siguiendo diferentes leyes probabilísticas para la interconexión de los nodos.

Este generador lleva el nombre de GT-ITM [7], correspondiente a las iniciales de Georgia Tech Internetwork Topology Models. Su funcionamiento se basa en una serie de scripts que indican al generador las características topológicas del modelo creado. Por ejemplo, podemos generar un grafo siguiendo alguno de los modelos aleatorios descritos, indicando el número de nodos y los diferentes parámetros probabilísticos. O si queremos generar el grafo con un modelo jerárquico, ya sea utilizando el modelo de nivel N o el Transit-Stub, podemos indicar al generador la distribución aleatoria de cada nivel jerárquico y sus parámetros.

La salida del GT-ITM es un fichero de texto con la información sobre los nodos y las aristas que forman el grafo base. Aunque se trata de un fichero simple, su comprensión es complicada debido al modo en que se definen los atributos de los nodos y aristas. Por ello se ha utilizado otra herramienta del generador para hacer más “comprensible” la estructura del grafo. De esto se encarga un programa que transforma la salida obtenida anteriormente a un nuevo formato, más sencillo de interpretar, ya que los atributos de los nodos y las aristas se escriben de una manera más ordenada. Además, este tipo de fichero es utilizable por otras aplicaciones, como por ejemplo el NED (Network Editor) [8],

un programa de código abierto escrito en Java que permite ver representados gráficamente los grafos generados, así como especificar ciertas propiedades de los enlaces o de los nodos.

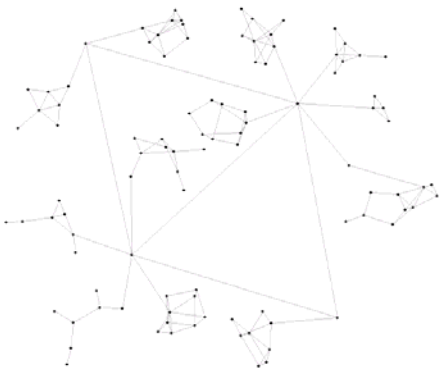


Fig.4.1 Representación con NED de uno de los grafos troncales utilizados

GT-ITM [7] ofrece asimismo un traductor que transforma el formato original en un formato interpretable por NS-2 (Network Simulator) [10], la herramienta más utilizada para hacer simulaciones de redes. El paquete también incluye herramientas para hacer estadísticas y diagnósticos sobre los grafos y comprobar que cumplen los parámetros impuestos.

Como nuestro objetivo es hacer una simulación del comportamiento del algoritmo, hemos generado 3 grafos troncales siguiendo un modelo Transit-Stub. Los parámetros utilizados para la construcción de los diferentes grafos los mostramos en la tabla 4.1. Obsérvese que no se permiten *enlaces extras* entre dos stubs, es decir, dos stubs deben estar conectados a través de un dominio de tránsito. Tampoco permitimos *enlaces extras* entre un stub y un dominio de tránsito, es decir, un stub debe estar conectado únicamente a un dominio de tránsito a través de un único enlace. La utilización del modelo de dominios de tránsito y acceso nos parece el más adecuado para este experimento, ya que interpretamos que los miembros del grupo multicast son ordenadores de usuarios conectados a Internet desde su lugar de trabajo o estudio, o desde su casa.

La topología definida consta de un conjunto de enlaces troncales (o de tránsito) que conectan los diferentes dominios y a los cuales se conectan las subredes independientes, de forma que haya muy pocos enlaces entre los nodos de la red y los nodos de tránsito y que las conexiones directas entre nodos de diferentes subredes sean muy poco probables. Esta topología se parece bastante a la situación que queremos representar: una colección de nodos que hacen de routers, con enlaces entre ellos de alta velocidad y tiempo de retardo elevado, que actúan como puertas de enlace de las subredes, que a su vez representan las redes de proveedores de Internet, o una LAN corporativa. Entonces, lo que hace falta es añadir los usuarios, es decir, los nuevos nodos terminales conectados al resto de la red a través de alguna red de acceso.

TABLA 4.1. PARÁMETROS GT-ITM

Dominios Stub / Nodos de Tránsito	3
Enlaces extras Tránsito-Stub	0
Enlaces extras Stub-Stub	0
Dominios de Tránsito	1
Nodos de Tránsito / Dominio de Tránsito	4
Prob. Enlace (entre nodos en el mismo Dominio de Tránsito)	0.6
Nodos Stub / Dominios Stub	8
Prob. Enlace (entre nodos en el mismo Dominio Stub)	0.42

Ancho de Banda Tránsito - Tránsito	1 Gbit / s
Retardo de propagación Tránsito - Tránsito	100 ms
Ancho de Banda Tránsito - Stub	100 Mbits / s
Retardo de propagación Tránsito - Stub	10 ms
Ancho de Banda Stub - Stub	100 Mbits / s
Retardo de propagación Stub - Stub	10 ms
Ancho de Banda de acceso para el Usuario	Aleatorio
Retardo de propagación de acceso	1 ms

4.3 Ampliación del grafo troncal con los nodos de usuario

A partir de ahora, todos los procedimientos de transformación de grafos necesarios para conseguir el modelo adecuado para la aplicación del algoritmo, así como su simulación, se ha de realizar mediante aplicaciones escritas con el lenguaje Java. Hemos escogido este lenguaje de programación porque, aunque puede ser un poco más lento que el lenguaje C (ya que el Java es un lenguaje interpretado), es fácil de utilizar (hay muchas librerías con muchas funciones, muy documentado…), y tiene una gran versatilidad (al ser interpretado se puede ejecutar un programa en Java sobre cualquier sistema operativo que tenga instalada la máquina virtual).

La primera aplicación se encarga de generar la red troncal y de añadirle un número *n* de nodos de usuarios *U*, que actuarán como los nodos terminales del grupo multicast. En el momento de crear el grafo troncal, igual que cuando hemos escogido los nodos de usuario, añadimos a las aristas unas etiquetas de coste. Cada enlace, además de un identificador, contiene dos etiquetas; el valor de *BW*, que indica el ancho de banda disponible en este enlace, en unidades de kbits/s; y el valor de *delay*, que indica el tiempo de propagación en unidades de milisegundos. En los modelos que hemos creado, estos parámetros tratan de representar los valores reales que tendrían en una red real. Así, los enlaces entre nodos *T* (que corresponderían a enlaces directos entre routers de dominios de Internet) tienen un ancho de banda de 1Gb/s, mientras que su retardo es de 100ms. Los enlaces entre nodos *S* y entre un nodo *S* y un nodo *T* (es decir, los enlaces dentro de la misma red de acceso, así como los existentes entre un nodo de la red de acceso y el router que actúa de puerta de enlace) tienen un ancho de banda de 100 Mb/s, mientras que su

retardo es de 10 ms. Por último, los enlaces entre un nodo *S* y un nodo *U* pueden ser muy variables.

Para que la red definida sea lo más realista posible, la asignación del ancho de banda de los enlaces ha considerado una encuesta realizada en Febrero de 2006 por AIMC (Asociación para la investigación de medios de comunicación) [11]. De este modo, el ancho de banda de los enlaces de acceso ha sido escogido conforme la siguiente distribución de probabilidades:

	Absolutos	**%**
BASE (accede desde casa)	52.398	100,0
No lo sé	3.725	7,1
56 Kbps	4.951	9,4
64 Kbps	445	0,8
128 Kbps	914	1,7
256 Kbps	2.235	4,3
512 Kbps	5.278	10,1
1 Megas (1 Mbps)	21.811	41,6
2 Megas (2 Mbps)	6.342	12,1
4 Megas (4 Mbps)	5.767	11,0
8 Megas (8 Mbps)	765	1,5
NS/NC	165	0,3

El retardo que hemos supuesto para enlaces entre *S* y *U* es de 1 ms, ya que suele tratarse de enlaces próximos. Debemos comentar también que en nuestro grafo las conexiones son simétricas, y que la velocidad de recepción y de transmisión es idéntica, lo que nos aleja de lo que podríamos considerar "real" en una red de usuarios conectados mediante ADSL. De todas maneras, el aumento de las velocidades "de subida" de las conexiones (en la actualidad, la gran mayoría de proveedores ofrecen 300kb/s) hace que los valores utilizados en la simulación no sean demasiado lejanos de la realidad.

4.4 La red overlay: aplicación del método de Dijkstra

Una vez se ha obtenido el grafo con los nodos troncales (de tipo *T* y *S*) y con los equipos terminales que forman el grupo multicast (nodos de tipo *U*), debe determinarse la red overlay. Esta red corresponde a la red que formarían los miembros del grupo, vista desde el nivel de aplicación. De hecho, ya hemos comentado que el algoritmo es aplicable a cualquier nivel de la torre OSI, pero dado que queremos estudiar su comportamiento en transmisión multimedia, sobre una determinada aplicación, nos limitaremos al nivel más elevado de la torre de protocolos.

Así, lo que necesitamos es un nuevo grafo, donde tengamos representados los nodos que forman el grupo, y donde se vean las conexiones entre sí. Dado

que, por la definición que hemos hecho del modelo, el grafo es conexo, todos los nodos de usuario pueden conectarse mediante un camino dentro del grafo a cualquier otro nodo *U*. Por lo tanto, el grafo resultante que representará la red overlay, será un grafo completo, ya que si suponemos que los miembros conocen al resto de los miembros del grupo, cada nodo del nuevo grafo estará conectado al resto de nodos. Además, cada arista tendrá una etiqueta con el coste de la transmisión. Este coste, según el modelo del cartero, sumará el tiempo de transmisión del nodo origen y el retardo total de la propagación. Es decir, despreciaremos el tiempo de procesado de los nodos *S* y *T* intermedios. Para ello se aplicará el algoritmo de Dijkstra, presentado en el capítulo 2.

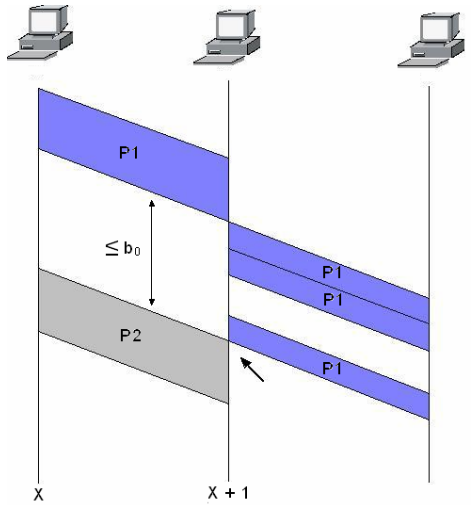
El procedimiento que sigue la aplicación programada es sencillo: para cada nodo *U* del grafo se aplica Dijkstra para llegar al resto de nodos. Al finalizar una iteración (es decir, cuando se ha ejecutado el algoritmo para un nodo *U*), tenemos el coste del camino mínimo desde el nodo origen hasta cualquiera de los otros nodos *U*. Este coste vendrá dado por el tiempo total que tarda un mensaje en ir de un lugar a otro. Como la arista que une estos dos nodos de tipo *U* representa la conexión que tendremos de manera directa entre los dos nodos, el ancho de banda correspondiente será el mínimo ancho de banda de todas las aristas del grafo original por las que hemos tenido que pasar para llegar al destino. Es decir, lo que haremos es calcular el mínimo ancho de banda entre el nodo origen del paquete, y el nodo destino. Las aristas intermedias no las calculamos porque sabemos que los nodos de tipo *U* pueden tener un ancho de banda comprendidos entre 56 kbps y 8 Mbps, y en cambio, las aristas que unen nodos de tipo *T* tienen ancho de banda de 1Gb/s y las aristas que unen nodos de tipo *S* o uno de tipo *S* con otro de tipo *T* tienen un ancho de banda de 100 Mb/s. Por lo tanto, de antemano ya sabemos que el mínimo ancho de banda será el de un nodo de tipo *U*, ya sea el de origen o el de destino. Por otro lado, el retardo de propagación será la suma de los retardos de los distintos enlaces.

El resultado, como hemos visto, es un grafo completo, donde para cada nodo de tipo *U* conocemos el retardo de transmisión hacia cualquier otro nodo del grupo.

4.5 Simulación de los algoritmos

Con el objeto de evitar la congestión de los nodos, de un lado, y de asegurar la conectividad de la transmisión, de otro, se han definido las siguientes tres versiones del algoritmo.

En la primera de ellas, llamada de *cadencia fija*, todos los nodos deben tener en cuenta la cadencia del nodo raíz. Dicho de otra forma: ningún nodo podrá transmitir un mismo paquete durante un intervalo mayor a b_0, en donde b_0 es el tiempo durante el cual el nodo raíz puede enviar a distintos nodos un mismo paquete (b_0 es, entonces, el número máximo de veces *S* que el nodo raíz puede transmitir un paquete por el tiempo medio de transmisión de todos los enlaces del nodo raíz). Esto se hace para evitar problemas de congestión, como se explica a continuación.



Supóngase que el nodo x+1 transmite el paquete 1 durante un cierto tiempo. Pero en el momento en que recibe el paquete 2 del nodo x, el nodo x+1 no debe de estar transmitiendo el paquete 1. Por eso limitamos a b_0 el tiempo máximo durante el cual un nodo puede transmitir un mismo paquete a otros nodos. Si el nodo x+1 transmitiera el paquete 1 y sobrepasara el instante en que comienza a recibir el paquete 2, éste debería almacenarse en la cola de entrada para ser reenviado más tarde. Por eso, lo que hacemos es parar de transmitir si vemos que superamos el intervalo marcado por b_0, y dejamos el canal libre de x+1 para poder retransmitir el paquete 2 en cuanto se reciba completamente. En este caso, sin embargo, podría suceder que el paquete no llegara a todos los nodos, como veremos más adelante.

A la segunda versión la hemos llamado *cadencia fija variable S*. En este caso lo que hacemos es reutilizar los árboles ya creados. Imaginemos que creamos una red y para S=1 el paquete se transmite por los nodos pero no llega a todos. Recuérdese que *S* es el parámetro que se emplea para calcular b_0, que es igual a *S* por el tiempo medio de transmisión del nodo raíz. Y recuérdese que la cadencia de todos los nodos no ha de superar la que marca b_0. En este caso, el algoritmo no finalizará su ejecución, sino que aumentará el valor de *S* a 2, y volverá a comenzar sin que los nodos que ya han recibido el paquete vuelvan a recibirlo. Es decir, para S=2 se vuelve a recorrer el camino que se ha creado para S=1 pero pudiendo llegar a más nodos. Pero esto no quiere decir que lleguemos a todos los nodos de la red, puede ser que tampoco lleguemos, en cuyo caso el algoritmo volverá a aumentar el valor de la *S* y volverá a recorrer el árbol creado para S=1 y posteriormente para S=2, llegando a más nodos. Esta operación se repetirá tantas veces como sea necesario hasta que se encuentre el valor mínimo de *S* que nos permita llegar a toda la red.

La tercera versión es la de *cadencia fija forzada*. En este caso lo que hacemos es forzar a que todos los nodos envíen el paquete por lo menos una vez. Con esta aplicación del algoritmo, conseguimos llegar a todos los nodos. El inconveniente, lógicamente, es que no cumpliremos la restricción de la cadencia marcada por b_0 y podremos tener, por lo tanto, problemas de congestión. Pero por lo menos nos aseguramos que los paquetes llegan a todos los nodos.

5. RESULTADOS Y CONCLUSIONES

Llegados a este punto, tenemos suficiente información para evaluar el comportamiento de las distintas versiones del algoritmo. Las pruebas se han realizado sobre redes simuladas, tal y como se ha explicado en capítulos anteriores, empleando el algoritmo en el nivel de aplicación. Esto tiene sentido dado que una de las primeras aplicaciones prácticas del algoritmo podría ser la transmisión de video en tiempo real sobre redes P2P. En este caso, además, todos los nodos verían a todos los miembros del grupo multicast –utilizando comunicación TCP-IP– de forma que el grafo resultante sería un grafo completo. El hecho de tener un grafo completo no debería ser un obstáculo para obtener tiempos de procesado bajos, ya que la complejidad de los algoritmos es polinómica de grado dos en relación al número de nodos.

Como se ha visto en el capitulo anterior, las pruebas se han realizado después de definir un conjunto variable de nodos de usuario (que representan los miembros del grupo multicast) sobre una red troncal que sigue el modelo de Zegura, Calvert y Bhattacharjee explicado en el primer capítulo. Para cada una de las redes troncales, se ha generado una colección de grafos con el mismo número de nodos terminales manteniendo la estructura del grafo troncal, y para cada uno de ellos se ha calculado el tiempo que un paquete tarda en llegar a todos los miembros del grupo.

Los resultados de la simulación se recogen en las siguientes tablas. En cada caso se muestra el retardo de transmisión de un paquete, expresado en milisegundos y promediado sobre diez grafos diferentes, definidos todos sobre el mismo grafo troncal. El parámetro *S* es el que define b_0, calculado como *S* por el tiempo de transmisión medio del nodo raíz.

En caso del algoritmo de *cadencia fija*, es posible que según la red los paquetes no lleguen a todos los nodos. En ese caso, hemos marcado en rojo el número de nodos a los que llegaría el paquete.

En el caso del algoritmo de *cadencia fija variable S,* se ha marcado en rojo la *S* mínima que necesitaremos en el caso en que la *S* inicial no haya permitido llegar a todos los nodos, y el tiempo que necesitará en ese caso para enviar el paquete a todos los nodos.

En el caso de *cadencia fija forzada,* hemos marcado en rojo los tiempos de retardo en el caso en que se haya tenido que forzar el envío de un paquete para todos los nodos.

TABLA 5.1. TRONCAL 1. CADENCIA FIJA

	S=1	S=2	S=3	S=4	S=5	S=6	S=7	S=8	S=9	S=10	S=11
10	6	8	415,6	332,2	332,2	332,2	332,2	332,2	332,2	332,2	332,2
25	20	21	22	22	23	428,1	428,1	428,1	428,1	426,2	426,9
50	36	47	47	48	48	48	489,4	489,4	489,4	486,3	486,3
100	89	95	95	95	95	98	491,2	491,2	491,2	491,2	491,2
200	174	185	185	185	185	185	515,1	515,1	505,5	505,5	505,5
500	429	431	431	445	461	461	535,7	527,3	525,6	525,6	525,6
1000	812	823	872	872	895	895	560,4	560,4	560,4	560,4	560,4

TABLA 5.2. TRONCAL 1. CADENCIA FIJA VARIABLE S

	S=1	S=2	S=3	S=4	S=5	S=6	S=7	S=8	S=9	S=10	S=11
10	S =3 861,3	S =3 747,8	415,6	332,2	332,2	332,2	332,2	332,2	332,2	332,2	332,2
25	S =6 1594,4	S =6 1229,1	S =6 1084,7	S =6 964,1	S =6 844,6	428,1	428,1	428,1	428,1	426,2	426,9
50	S =7 1684,5	S =7 1769,8	S =7 1419,8	S =7 1216,3	S =7 1043,2	S =7 870,3	489,4	489,4	489,4	486,3	486,3
100	S =7 1843,7	S =7 1651,2	S =7 1462,2	S =7 1260,5	S =7 1055,5	S =7 849,8	491,2	491,2	491,2	491,2	491,2
200	S =7 2180,8	S =7 1924,4	S =7 1721,3	S =7 1545,4	S =7 1365,9	S =7 1180,5	515,1	515,1	505,5	505,5	505,5
500	S =7 2564,1	S =7 2103,1	S =7 1987,8	S =7 1642,5	S =7 1425,7	S =7 1213,6	535,7	527,3	525,6	525,6	525,6
1000	S =7 2887,3	S =7 2446,1	S =7 2259,9	S =7 1772,9	S =7 1519,4	S =7 1376,7	560,4	560,4	560,4	560,4	560,4

TABLA 5.3. TRONCAL 1. CADENCIA FIJA FORZADA

	S=1	S=2	S=3	S=4	S=5	S=6	S=7	S=8	S=9	S=10	S=11
10	484,4	435,2	415,6	332,2	332,2	332,2	332,2	332,2	332,2	332,2	332,2
25	503,7	435,3	428,1	428,1	428,1	428,1	428,1	428,1	428,1	426,2	426,9
50	510,5	495,6	492,2	490,3	490,3	490,3	489,4	489,4	489,4	486,3	486,3
100	518,4	499,2	493,1	493,1	493,1	491,2	491,2	491,2	491,2	491,2	491,2
200	554,2	521,5	520,7	520,7	520,7	520,7	515,1	515,1	505,5	505,5	505,5
500	683,3	637,9	554,8	541,2	537,3	535,7	535,7	527,3	525,6	525,6	525,6
1000	732,4	671,7	566,8	564,8	560,4	560,4	560,4	560,4	560,4	560,4	560,4

TABLA 5.4. TRONCAL 2. CADENCIA FIJA

	S=1	S=2	S=3	S=4	S=5	S=6	S=7	S=8	S=9	S=10	S=11
10	9	9	9	446,4	446,2	446,7	446,8	446,2	446,3	446,9	446,5
25	19	20	20	548,0	537,8	537,9	537,2	537,6	537,1	537,1	537,4
50	44	46	46	46	46	46	551,4	547,2	543,8	543,3	543,9
100	85	89	92	92	92	92	573,3	571,7	571,2	571,6	571,5
200	155	177	179	179	182	187	609,7	607,2	607,2	607,2	607,2
500	422	430	443	444	444	465	615,3	615,5	615,4	615,4	615,7
1000	721	826	857	859	894	894	622,2	622,9	622,7	622,4	622,3

TABLA 5.5. TRONCAL 2. CADENCIA FIJA VARIABLE S

	S=1	S=2	S=3	S=4	S=5	S=6	S=7	S=8	S=9	S=10	S=11
10	S =5 1475,4	S =5 1214,2	S =5 964,7	446,4	446,2	446,7	446,8	446,2	446,3	446,9	446,5
25	S =4 1491,3	S =4 1223,5	S =4 903,5	548,0	537,8	537,9	537,2	537,6	537,1	537,1	537,4
50	S =7 1986,7	S =7 1746,7	S =7 1506,4	S =7 1266,6	S =7 1026,9	S =7 802,2	551,4	547,2	543,8	543,3	543,9
100	S =7 2275,3	S =7 1898,1	S =7 1615,1	S =7 1362,9	S =7 1116,6	S =7 854,7	573,3	571,7	571,2	571,6	571,5
200	S =7 2361,6	S =7 1921,2	S =7 2617,9	S =7 1421,4	S =7 1154,1	S =7 888,3	609,7	607,2	607,2	607,2	607,2
500	S =7 2461,1	S =7 2093,6	S =7 1779,3	S =7 1427,6	S =7 1275,7	S =7 894,4	615,3	615,5	615,4	615,4	615,7
1000	S =7 2613,8	S =7 2211,9	S =7 1852,2	S =7 1496,3	S =7 1163,1	S =7 923,9	622,2	622,9	622,7	622,4	622,3

TABLA 5.6. TRONCAL 2. CADENCIA FIJA FORZADA

	S=1	S=2	S=3	S=4	S=5	S=6	S=7	S=8	S=9	S=10	S=11
10	457,5	446,5	446,2	446,4	446,2	446,7	446,8	446,2	446,3	446,9	446,5
25	571,3	550,7	550,7	548,0	537,8	537,9	537,2	537,6	537,1	537,1	537,4
50	576,7	556,1	551,7	551,7	551,7	551,7	551,4	547,2	543,8	543,3	543,9
100	619,4	588,1	571,7	573,7	574,2	574,9	573,3	571,7	571,2	571,6	571,5
200	643,3	619,1	615,7	616,2	616,2	615,8	609,7	607,2	607,2	607,2	607,2
500	654,1	626,4	626,4	625,7	625,7	618,3	615,3	615,5	615,4	615,4	615,7
1000	726,6	718,4	711,1	711,1	708,5	708,5	622,2	622,9	622,7	622,4	622,3

TABLA 5.7. TRONCAL 3. CADENCIA FIJA

	S=1	S=2	S=3	S=4	S=5	S=6	S=7	S=8	S=9	S=10	S=11
10	8	9	9	9	9	409,5	409,5	409,5	409,5	409,5	409,5
25	21	23	23	23	23	519,1	515,2	515,2	515,2	515,2	515,2
50	47	49	49	49	49	530,8	525,2	525,2	525,2	525,2	525,2
100	87	91	91	94	94	94	547,7	541,7	534,9	534,9	534,9
200	163	171	171	175	177	177	540,6	539,6	539,6	539,6	539,6
500	406	438	443	444	467	467	557,3	557,8	557,8	557,8	557,8
1000	800	889	902	902	905	905	917	601,4	600,8	600,8	600,8

TABLA 5.8. TRONCAL 3. CADENCIA FIJA VARIABLE S

	S=1	S=2	S=3	S=4	S=5	S=6	S=7	S=8	S=9	S=10	S=11
10	S =6 1105,5	S =6 993,1	S =6 881,7	S =6 720,7	S =6 520,5	409,5	409,5	409,5	409,5	409,5	409,5
25	S =6 1751,8	S =6 1574,7	S =6 1340,1	S =6 1110,3	S =6 873,4	519,1	515,2	515,2	515,2	515,2	515,2
50	S =6 1626,6	S =6 1382,9	S =6 1137,3	S =6 886,2	S =6 764,2	530,8	525,2	525,2	525,2	525,2	525,2
100	S =7 1875,9	S =7 1625,2	S =7 1372,2	S =7 1138,8	S =7 1068,7	S =7 836,1	547,7	541,7	534,9	534,9	534,9
200	S =7 2203,3	S =7 1926,5	S =7 1655,4	S =7 1393,1	S =7 1131,8	S =7 869,8	540,6	539,6	539,6	539,6	539,6
500	S =7 2493,1	S =7 2245,7	S =7 2016,6	S =7 1727,5	S =7 1512,5	S =7 1078,7	557,3	557,8	557,8	557,8	557,8
1000	S =8 2569,2	S =8 2261,8	S =8 2074,3	S =8 1853,9	S =8 1697,4	S =8 1273,4	S =8 1024,5	601,4	600,8	600,8	600,8

TABLA 5.9. TRONCAL 3. CADENCIA FIJA FORZADA

	S=1	S=2	S=3	S=4	S=5	S=6	S=7	S=8	S=9	S=10	S=11
10	446,7	422,2	412,1	412,2	410,5	409,5	409,5	409,5	409,5	409,5	409,5
25	521,1	518,8	518,8	517,6	517,6	519,1	515,2	515,2	515,2	515,2	515,2
50	541,4	538,3	538,3	537,5	537,5	530,8	525,2	525,2	525,2	525,2	525,2
100	571,8	539,2	539,2	538,1	538,1	538,1	547,7	541,7	534,9	534,9	534,9
200	622,6	589,7	589,7	572,1	565,2	565,2	540,6	539,6	539,6	539,6	539,6
500	667,2	630,5	594,1	581,6	572,3	572,3	557,3	557,8	557,8	557,8	557,8
1000	730,9	701,2	686,4	652,5	623,8	617,1	617,1	601,4	600,8	600,8	600,8

Puede observarse, en primer lugar, que en aquellos casos en que *S* es suficientemente grande para llegar a todos los nodos sin tener que hacer modificaciones, los tres algoritmos son idénticos y por lo tanto los retardos son los mismos (en las tablas estos retardos aparecen en negro, en contraposición a los retardos marcados en rojo). Se observa asimismo que el tiempo de transmisión de un paquete varía poco según la topología de la red. Si nos fijamos en las tablas veremos como lógicamente el tiempo de transmisión de un paquete es mayor cuanto mayor es el número de nodos a los que tiene que llegar, aunque el aumento del retardo es de carácter logarítmico ya que, por la construcción de los algoritmos, el número de nodos que han recibido el paquete aumenta en una proporción geométrica a lo largo del tiempo.

La diferencia más significativa la encontramos en el valor de *S* (número máximo de veces que el nodo raíz puede transmitir un paquete), ya que podemos ver que el retardo va disminuyendo conforme aumenta el valor de *S*, en todos los casos (ya sea *cadencia fija, cadencia fija variable S* o *cadencia fija forzada*). En todo caso, llega un momento en que valor del tiempo de transmisión se mantiene constante. Esto es debido a que hemos llegado al "valor óptimo" de *S*, y no es necesario que los nodos de la red realicen un mayor número de retransmisiones. Por ejemplo, en el caso de la tabla 5.1 para el algoritmo de *cadencia fija*, puede verse que para n=10 el tiempo de transmisión no varía a partir de S=4. En muchas ocasiones el valor óptimo de *S* aparece para valores elevados como S=10, S=11, pero podemos observar que, en general, a partir de S=6 la variación del retardo es de pocos milisegundos.

De las tablas anteriores se deduce que el modelo de *cadencia fija* no siempre llega a todos los nodos. Los otros dos algoritmos llegan siempre ya sea aumentando el valor de *S* u obligando a todos los nodos a retransmitir un paquete como mínimo una vez. En este caso los tiempos de retardo son mejores en el último modelo, el de *cadencia fija forzada* a pesar de que será más probable encontrarse con problemas de congestión.

En el caso de los algoritmos de *cadencia fija variable S*, se comprueba lo siguiente. Cuando se parte de S=3, por ejemplo, y no se llega a todos los nodos y se ha de forzar la S a llegar a 6, el retardo total es superior al retardo obtenido de aplicar directamente S=6. Esto es así porque cuando se pasa de S=3 a S=4 y después de S=4 a S=5 y así sucesivamente, no se renuevan los árboles sino que se completan los árboles iniciales, los cuales son deficientes puesto que inicialmente se han construido sin aprovechar todas las ramas rápidas que permitiría un valor de S=6, lo cual obligará a otros nodos peores con menor ocupación a emplear ramas más lentas. Todo ello explica igualmente que aun en el caso en que deba aumentarse *S* hasta un valor de S=6, por ejemplo, el retardo será mayor si se empieza con una *S* menor. En todas las tablas del algoritmo de *cadencia fija variable S*, se ve que con S=2, por ejemplo, se obtienen retardos menores que con S=1 aunque finalmente en los dos casos se tenga que aumentar progresivamente el valor de *S* hasta llegar a 6.

De lo dicho en el anterior párrafo –así como de los resultados de las tablas– se deduce que en la práctica no tiene sentido aplicar el algoritmo de *cadencia fija*

variable S, sino en su lugar el modelo de *cadencia fija* aumentando la *S* hasta conseguir conectividad (y renovando completamente los árboles cada vez que se empiezan los cálculos con una nueva *S*). De esta forma, no iremos aprovechando árboles anteriores –como haría *cadencia fija variable S* al ir aumentado el valor de *S*– y por lo tanto tardaremos un poco más en completar el cálculo de los árboles de encaminamiento cada vez que incrementamos el valor de *S*. Pero por otra parte, tal y como se observa en las tablas anteriores y se explica en el anterior párrafo, los retrasos de transmisión serán mucho menores. En los casos extremos, por ejemplo, puede observarse que el retardo puede disminuir de 2500 hasta 500 milisegundos si en vez del algoritmo de *cadencia fija variable S*, aplicamos el modelo de *cadencia fija* con el valor de *S* mínimo que asegure conectividad. Hay que añadir que el aumento del tiempo de computación al renovar completamente los árboles tampoco es muy importante dada la escasa complejidad de los algoritmos planteados. En futuros trabajos podría hacerse una comparación de dicho diferencia en el tiempo de computación de las soluciones planteadas.

En los casos de c*adencia fija forzada* se observa que para valores pequeños de *S* difícilmente llegamos a todos los nodos. Por eso hacemos que los nodos puedan enviar cada paquete por lo menos una vez, aunque sabemos que estaremos incumpliendo una de nuestras premisas, que es no superar el tiempo de cadencia b_0. En la tabla siguiente se indica en cada caso el número de nodos que superan, de un lado, el valor de b_0 y, de otro, el intervalo de transmisión del root, que no tienen porqué ser iguales ya que dependiendo de la topología de la red, el nodo raíz podría enviar un paquete menos de *S* veces y tener un intervalo de transmisión de un paquete menor que b_0.

TABLA 5.10. TRONCAL 1. CADENCIAS SUPERADAS

	S=1	S=2	S=3	S=4	S=5	S=6	S=7	S=8	S=9	S=10	S=11
10	b0:2 root:2	b0:1 root:1	b0:0 root:1	b0:0 root:1	b0:0 root:1	b0:0 root:1	b0:0 root:1	b0:0 root:1	b0:0 root:1	b0:0 root:1	b0:0 root:1
25	b0:3 root:5	b0:2 root:2	b0:2 root:2	b0:2 root:2	b0:2 root:2	b0:0 root:2	b0:0 root:2	b0:0 root:2	b0:0 root:2	b0:0 root:2	b0:0 root:2
50	b0:10 root:12	b0:6 root:9	b0:5 root:5	b0:5 root:5	b0:5 root:5	b0:4 root:5	b0:0 root:5	b0:0 root:5	b0:0 root:5	b0:0 root:5	b0:0 root:5
100	b0:18 root: 19	b0:16 root: 18	b0:14 root: 16	b0:14 root: 16	b0:14 root: 16	b0:14 root: 16	b0:0 root: 16	b0:0 root: 16	b0:0 root: 16	b0:0 root: 16	b0:0 root: 16
200	b0:31 root: 44	b0:26 root: 26	b0:21 root: 23	b0:21 root: 22	b0:21 root: 22	b0:21 root: 22	b0:0 root: 22	b0:0 root: 22	b0:0 root: 22	b0:0 root: 22	b0:0 root: 22
500	b0:68 root: 172	b0:45 root: 72	b0:38 root: 46	b0:38 root: 38	b0:38 root: 38	b0:38 root: 38	b0:0 root: 38	b0:0 root: 38	b0:0 root: 38	b0:0 root: 38	b0:0 root: 38
1000	b0:187 root: 343	b0:145 root: 143	b0:109 root: 105	b0:104 root: 100	b0:102 root: 87	b0:100 root: 87	b0:0 root: 87	b0:0 root: 87	b0:0 root: 87	b0:0 root: 87	b0:0 root: 87

Los valores marcados en rojo indican el número de nodos que han sobrepasado el valor de b_0, debido a que se ha tenido que forzar para todos los nodos el envío de un paquete para asegurar la conectividad del algoritmo. Podemos ver que este conflicto se produce para valores bajos de *S*, y a medida

que el valor de *S* aumenta, el número de nodos que sobrepasan b_0 disminuye hasta llegar a 0, dado que no hace falta forzar ningún envío de paquetes para llegar a todos los nodos.

Lo que puede suceder, sin embargo, es que incluso para valores altos de *S* los nodos sobrepasen el tiempo total de transmisión del root. Supóngase, por ejemplo, que el tiempo de transmisión del nodo raíz es de 5 ms para cada paquete y el valor de *S* es igual a cinco, en cuyo caso b_0 valdría 25 ms. En este caso, sin embargo, podría suceder que el root sólo tuviera que transmitir 4 veces un paquete, en cuyo caso transmitiría cada paquete durante 20 ms. En este caso, además, podríamos tener nodos que transmitieran cada paquete durante 25 ms o 24 ms, por ejemplo, lo que sería “correcto” aun sin forzar la transmisión dado que no transmitirán durante un intervalo mayor a b_0, pero que podría provocar problemas de congestión dado que dichos nodos recibirán el segundo paquete antes de terminar la retransmisión del primero y tendrán por tanto que guardarlo en cola.

Ya se ha dicho que en aquellos casos en que *S* es suficientemente grande para llegar a todos los nodos, los tres algoritmos son idénticos y los retardos los mismos. Por otra parte, de los resultados de la tabla 5.10 para valores de *S* elevados, se desprende que en todos los casos podríamos tener nodos con una cadencia más lenta a la del nodo raíz, lo que podría provocar problemas de congestión. En un futuro trabajo, por ello, se aplicará el algoritmo limitando la cadencia de todos los nodos no conforme un valor fijo b_0 sino conforme la cadencia real del nodo raíz durante la evolución del algoritmo. De esta forma se evitarán posibles problemas de congestión ya que ningún nodo retransmitirá un mismo paquete durante un intervalo superior al del nodo raíz, lo que significa que cuando reciban el segundo paquete todos los nodos habrán terminado de retransmitir el primer paquete y que no habrá por tanto congestión. Bajo estas circunstancias se compararán los retrasos del algoritmo de *cadencia fija* y el de *cadencia fija forzada*, teniendo en cuenta que en este último caso los retardos serán probablemente menores pero tendremos riesgo de congestión si forzamos a los nodos a enviar cada paquete por los menos una vez.

Igualmente, las pruebas en el presente trabajo se han realizado para la transmisión de un único paquete. En este caso siempre convendrá aumentar el valor de *S* para llegar antes a todos los nodos. En el caso habitual en que se quisiera mandar más de un paquete, no obstante, habría que estudiar la posibilidad de reducir *S* para aumentar la cadencia del nodo raíz y disminuir por tanto el retardo final de la transmisión. Este estudio se deja para futuros proyectos en los que, además, está previsto realizar simulaciones en un entorno real de pruebas como Planet Lab.

BIBLIOGRAFIA

[1] Bar-Noy, A. and Kipnis, S., “Designing Broadcasting Algorithms in the Postal Model for Message-Passing Systems” de *(ACM) Symposium on Parallel Algorithms and Architectures,* pp. 13-22, 1992.

[2] Zegura, E. W. and Calvert, K. L. and Bhattacharjee, S., “How to Model an Internetwork” de *IEEE Infocom*, pp. 594-602, IEEE, San Francisco, 1996.

[3] Gibbons, *Algorithmic Graph Theory,* Cambridge University Press, Cambridge, 1985.

[4] Wilson, R. J., *Introducción a la Teoría de Grafos,* Alianza Universidad, Alianza Editorial, Madrid, 1972.

[5] Ozón, J., *Contribución al coloreado de grafos y las redes pequeño mundo*, Tesis Doctoral, Universitat Politècnica de Catalunya, 2001.

[6] Huarte, B., *Optimización de rutas para el transporte urbano de mercancías,* Projecte Final de Carrera ETSETB, Barcelona, 2004.

[7] GT-ITM, Modeling Topology of Internetworks, “http://www.cc.gatech.edu/projects/gtitm”.

[8] NED Network EDitor, “http://www.cs.nyu.edu/pdsg/projects/partitionable-services/ned/ned.htm”.

[9] Java Technology, “http://java.sun.com”.

[10] The Network Simulator NS-2, “http://www.isi.edu/nsnam/ns”.

[11] AIMC, Asociación para la investigación de medios de comunicación