Simulación de una Red P2P de Distribución de Video en Vivo

Facultad de Ingeniería

Universidad de la República

Proyecto de Grado

Carrera de Ingeniería en Computación

8 de abril de 2008

Autores:
Marcelo Martínez
Alexis Morón

Supervisores:
Ing. Pablo Rodríguez-Bocca
Dr. Franco Robledo

Agradecimientos

Queremos agradecer a nuestros tutores Pablo Rodríguez-Bocca y Franco Robledo por todo el apoyo que nos han brindado durante la realización de este trabajo.

Agradecemos a nuestras familias y amigos por comprender nuestra ausencia y por nunca dejar de motivarnos durante el desarrollo de nuestra tesis de fin de carrera.

Especialmente queremos agradecer a Nayely, no sólo por su gran tolerancia sino también por los innumerables consejos que fueron muy útiles para la realización de un proyecto de estas características.

Resumen

En este trabajo se considera una red de distribución de video en vivo basada en un modelo *Peer-to-Peer* o P2P, como alternativa a las arquitecturas cliente-servidor clásicas de distribución de contenido (*Content Delivery Networks*). En primer lugar, se hace un análisis de lo que puede aportar un modelo P2P a la distribución de video en vivo. Luego, se plantea un modelo matemático formal para una red de este tipo, teniendo en cuenta los dos principales problemas que se presentan en este caso: las restricciones de ancho de banda de los clientes de la red y la dinámica de los nodos. Para resolver el problema de la dinámica (frecuentes conexiones y desconexiones de nodos), se definen algoritmos basados en metodologías de optimización como *GRASP* y *Random Neural Networks*. Con el objetivo de probar los algoritmos, se desarrolla una aplicación que simula el comportamiento de una red como la que se describe en el modelo matemático. Por último, para evaluar el rendimiento de los algoritmos en situaciones cercanas a la realidad, se realizan simulaciones con casos de prueba generados a partir de datos obtenidos de un servicio de distribución de video en vivo existente.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Como introducción a este trabajo, se explica primero cuál es la motivación para realizarlo, así como su relevancia en el área correspondiente. Luego, se da un panorama más amplio al describir el contexto en el cual está enmarcado este trabajo, y cómo encaja con el resto de los trabajos relacionados que se desarrollan paralelos a éste. Por último, se presenta la manera de la cual está estructurado este documento, con una breve descripción de los temas tratados en cada capítulo.

1.1. Motivación

El método actual más común de distribución de contenido a través de Internet (y, en particular, video en vivo) está basado en soluciones del tipo *cliente-servidor*, que utilizan principalmente servidores, organizados en clusters y/o mirrors, para distribuir el contenido. Con la aparición de las redes *Peer-to-Peer*, o P2P, se introdujo la posibilidad de aprovechar recursos de los clientes, por ejemplo el ancho de banda disponible, para distribuir contenido (originalmente archivos) sin necesidad de servidores dedicados, ya que los mismos clientes actuarían también como servidores. Surge entonces intuitivamente la idea de aplicar estas redes a la distribución de video en vivo por Internet; en un principio, esta idea es alentadora, pues al utilizar a los propios clientes para transmitir a otros el contenido, se evitaría la necesidad de tener la costosa infraestructura de servidores con grandes cantidades de anchos de banda disponible, como es el caso de los métodos tradicionales de distribución. Ahora bien, este enfoque también puede presentar sus problemas, y por eso es necesario estudiarlo en detalle. En particular, la dinámica (frecuentes conexiones y desconexiones de nodos) en las redes P2P es un problema que gran parte del trabajo está dedicada a atacar.

1.2. Contexto del trabajo

Este trabajo forma parte del estudio de doctorado [1], el cual lleva adelante un proyecto más amplio, llamado *Gol!P2P*[1], entre las instituciones IRISA[2],

[1] http://p2ptv.gforge.inria.fr
[2] http://www.irisa.fr

ubicada en Rennes, Francia, y la Facultad de Ingeniería de la Universidad de la República[3], ubicada en Montevideo, Uruguay. El objetivo de dicho proyecto es la construcción de un prototipo de red P2P de distribución de video en vivo. El rol de este trabajo en dicho proyecto consistió en plantear un modelo formal para una red de ese tipo, y el desarrollo de algoritmos dedicados a minimizar el impacto causado por la dinámica de los nodos en la red, así como la construcción de un marco de trabajo sobre el cual realizar pruebas de estos algoritmos. Otros trabajos que atacan otras partes de este problema consisten en: definir un método de distribución del video en la red [2] (de hecho, se utilizará la técnica de Streaming Multifuente, explicada más adelante), auditar la distribución de video que se realiza [3], etc.

1.3. Organización del trabajo

Este trabajo está organizado de la siguiente manera:

En el **Capítulo 2**, se mencionan en primer lugar las características principales de los métodos actuales de distribución de contenido multimedia a través de Internet (CDN). Luego, se introduce el concepto de *Redes P2P*, describiendo a grandes rasgos cómo están estructuradas y en qué ideas se basa su funcionamiento. Se presenta entonces un relevamiento realizado sobre las redes P2P en general, analizando las características que distinguen a cada una de ellas, y los métodos que utilizan para resolver los problemas que se les presentan, todo esto con el objetivo de identificar técnicas y estructuras comunes en este tipo de redes que podamos aplicar a este trabajo. Como resumen de este relevamiento, se realiza una clasificación de las redes estudiadas según sus características. Una vez que se dispone del background suficiente acerca de la distribución de contenido en Internet y las redes P2P, se plantea el uso de este tipo de redes para la distribución de video en vivo a través de Internet como alternativa a los métodos tradicionales, evaluando ventajas y desventajas de cada uno de estos dos enfoques, para determinar qué soluciones puede brindar la alternativa basada en P2P, y a su vez cuáles son los principales problemas y limitaciones de ésta. Finalmente, se presenta la técnica de *Streaming Multifuente* como método utilizado para solucionar el problema causado por las restricciones de ancho de banda de los clientes en una red P2P de distribución de video.

En el **Capítulo 3**, se desarrolla un modelo matemático para una red P2P de distribución de video en vivo. Esto se hace para tener una definición rigurosa de todos los conceptos que se manejan para una red de este tipo, y para presentar de una manera formal los problemas y restricciones (dinámica de los nodos, ancho de banda disponible, etc.) que surgen a la hora de implementar este método de distribución de video. Primero se definen todos los elementos que conforman la red, como substreams, nodos y asignaciones, y cómo estos elementos están organizados dentro de la red. Con estos conceptos básicos ya definidos, se pasa a especificar primero un modelo *dinámico* de la red, es decir, como evoluciona ésta a través del tiempo. Luego, se introduce un modelo *estático* de la red, que describe la red en un instante dado, y se determinan las restricciones que deben cumplirse para que el estado global de la red en ese

[3] http://www.fing.edu.uy

instante sea óptimo según la medida de calidad que utilizaremos para el video distribuido por streaming, llamada PSQA . Posteriormente, se le añaden condiciones iniciales a este modelo estático, para considerar la posibilidad de que existan subárboles desconectados y nuevos nodos en el instante considerado, de modo tal de que pueda aplicarse este modelo en cualquier momento dado. Por último, se modifican las restricciones del modelo estático para hacerlo *robusto*, es decir, que la calidad global percibida en la red sea óptima no en el instante considerado, sino en el *próximo* instante, cuando se hayan producido conexiones y desconexiones de nodos; el objetivo de tener un modelo robusto es minimizar el impacto en la calidad producido por los nodos que se desconectarán en el siguiente intervalo de tiempo. Con el modelo ya definido, se discute la dificultad de encontrar una solución matemática para las restricciones planteadas, y cómo esto lleva al uso de heurísticas para resolver los problemas de dinámica en la red.

En el **Capítulo 4**, se parte del objetivo de definir algoritmos de asignación cuya función sea llevar la red de distribución de video a un estado que sea lo más cercano posible al modelo robusto que se describe en el Capítulo 3. La idea es ejecutar estos algoritmos de asignación para minimizar la pérdida de calidad producida por la dinámica de los nodos. El primer paso en este sentido es construir un marco de trabajo sobre el cual se implementarán y probarán estos algoritmos. Para ello, se describe la construcción de un simulador de la red P2P de distribución de video considerada, sobre el cual se pueden probar los algoritmos de asignación que se definan. Luego, se define un algoritmo de asignación implementado basándose en la heurística GRASP (para la cual se ofrece una introducción teórica). Posteriormente, se comenta la dificultad de implementar una fase de búsqueda local para mejorar la efectividad del algoritmo basado en GRASP, y se introduce entonces el concepto de *Random Neural Networks* (denotado *RNN*) con el objetivo de hacer una mejor exploración del espacio de soluciones en el algoritmo. En la última sección, se comentan otros algoritmos de asignación desarrollados, el de fuerza bruta y el algoritmo Proba, que fue desarrollado en Francia.

En el **Capítulo 5**, se presentan las pruebas realizadas sobre el simulador y los algoritmos de asignación implementados, los objetivos de éstas y los resultados obtenidos. La primera parte de este capítulo se enfoca a las pruebas llevadas a cabo para comprobar la *correctitud* del simulador y los algoritmos. Para ello se describen casos de prueba relativamente pequeños que permitan seguir detalladamente la simulación. Luego, se describen pruebas realizadas con el objetivo de evaluar el rendimiento del simulador en una situación más cercana a la realidad, para lo cual se utilizan datos obtenidos de un servicio existente de distribución de video en vivo. Estos datos se usan entonces para comparar la performance de los algoritmos implementados de modo tal de determinar fortalezas y debilidades de cada uno. Finalmente, se hace un resumen de las pruebas realizadas.

En el **Capítulo 6**, se hace un balance del trabajo realizado, detallando los aportes realizados al contexto más amplio en el cual nos encontramos. Además, se evalúan las posibilidades de futuros trabajos que se abren a partir de este.

En el **Apéndice A**, se describen en detalle los archivos de entrada y configuración que maneja el simulador y los diferentes algoritmos implementados,

explicando en cada caso los diferentes parámetros y los posibles valores de éstos.

En el **Apéndice B**, se proporciona una guía de uso para poder instalar, configurar y ejecutar el simulador a partir de los fuentes que se distribuyen.

En el **Apéndice C**, se presentan los casos de prueba utilizados para probar la correctitud del simulador y los algoritmos de asignación implementados, así como los parámetros a partir de los cuales se realizó la generación automática de estos casos.

En el **Apéndice D**, se describe el método utilizado para generar casos de prueba para el simulador a partir de datos reales obtenidos de logs que fueron proporcionados por un servicio de distribución de video en vivo existente.

Capítulo 2

Contexto Teórico

En este capítulo se presentará el marco teórico sobre el cual está basado este trabajo. En primer lugar, se describe el método tradicional de distribución de contenido a través de internet (CDN), así como algunas arquitecturas típicas usadas en estas redes a modo de ejemplo. Luego, se introducen las redes P2P como alternativa a las CDN para la distribución de contenido, tras lo cual se comparan las ventajas y desventajas de ambos enfoques. También se citan algunos ejemplos de redes P2P para familiarizarse con este tipo de arquitecturas. Finalmente, se presenta el concepto de Streaming Multifuente con el objetivo de ser usado en una red P2P de streaming de video para lograr superar la restricción de ancho de banda que tienen los clientes de la red. Con los conceptos introducidos en este capítulo será posible detallar un modelo teórico formal de una red P2P de streaming de video en vivo.

2.1. Redes de Distribución de Contenido (CDN)

2.1.1. Introducción a las CDN

A medida que aumenta el ancho de banda disponible para los usuarios de Internet, se hace cada vez más común la posibilidad de recibir grandes volúmenes de información que hace algunos años hubiera sido difícil de imaginar. Un ejemplo claro de esto es el contenido multimedia, en particular el video, que se caracteriza por tener tamaño muy superior a la mayoría de los otros contenidos (como texto, audio, etc.). Es por esta razón que surgen servicios destinados específicamente a la distribución de contenido, y que evolucionaron más allá de un simple servidor que posee el contenido y lo distribuye a los clientes del servicio.

Las *Redes de Distribución de Contenido*, o *CDN* [4] (*Content Delivery Networks*) están constituidas por un conjunto de computadoras en red que se encargan de distribuir distintos tipos de contenido a los clientes del servicio (ver Figura 2.1). La idea general de estas redes es que sus componentes cooperen entre sí para optimizar la distribución del contenido, aprovechando el ancho de banda y la capacidad de procesamiento del que disponen. Sin embargo, esto debe hacerse de un modo transparente para los clientes, quienes ven a la CDN como un único objeto. En la Figura 2.2 se puede apreciar una arquitectura genérica para una CDN distribuida en varios países distintos.

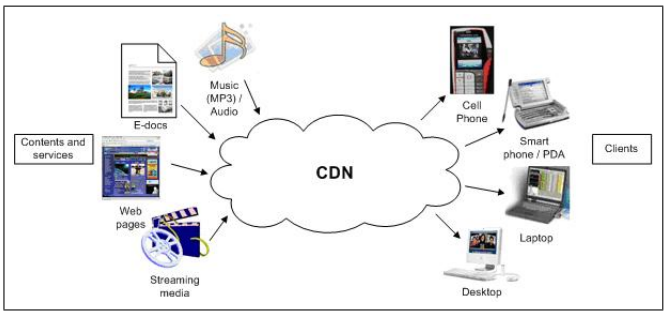


Figura 2.1: Contenidos distribuidos por una CDN

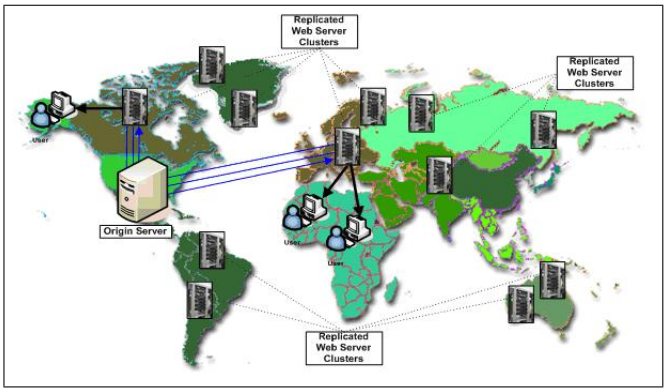


Figura 2.2: Arquitectura de una CDN genérica

Optimización de la distribución del contenido

Las CDN emplean diversas técnicas para optimizar la distribución del contenido, a continuación se mencionarán algunas de estas:

- El uso de *Web Caches* permite almacenar los contenidos a los que se accede con más frecuencia lo más cerca posible del usuario final, para no sólo mejorar los tiempos de respuesta al transmitir este contenido sino también reducir la carga en los servidores de la CDN.
- La distribución de servidores en *cluster* significa un aumento en la capacidad total (de procesamiento y ancho de banda) de la CDN, posibilita la realización de *load balancing* (equilibrio de carga) entre los servidores que componen el cluster, y también permite reasignar toda la carga de un servidor a los demás cuando éste falla.
- También existe la posibilidad de colocar *mirrors* de estos clusters de servidores en diferentes lugares, para que cada cliente reciba el contenido del mirror más cercano a él, reduciendo el tiempo de respuesta.

2.1.2. Arquitecturas de CDN

A continuación se describe la arquitectura de algunas CDN para ejemplificar los conceptos introducidos en la sección anterior.

Akamai

Akamai[1] es una compañía creada en 1998 con el objetivo de proporcionar un servicio de distribución de contenido (generalmente multimedia, como audio, video, imágenes). Akamai almacena en sus servidores los contenidos que sus clientes[2] desean distribuir. Cuando un usuario solicita contenido a los servidores de estas empresas, éste es obtenido desde los servidores de Akamai de forma transparente.

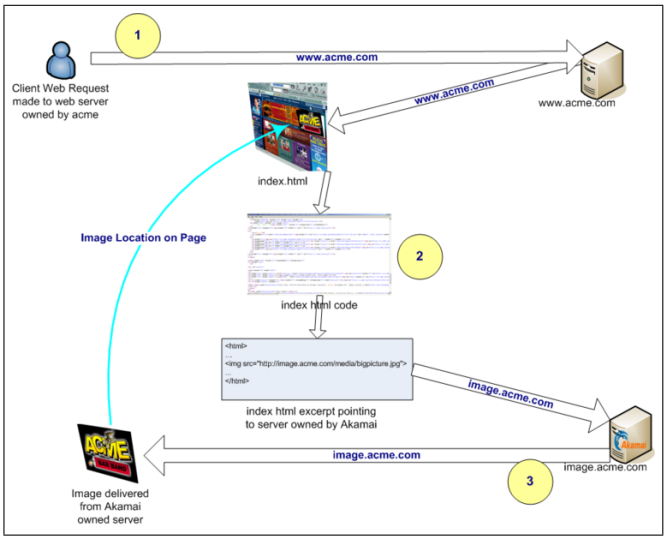


Figura 2.3: Manejo de una solicitud de contenido en Akamai

En la Figura 2.3 se puede ver como Akamai procesa las interacciones y distribuye de modo transparente el contenido a los usuarios finales, sin que éstos sepan que el servidor al que se conectaron redirigió la solicitud a un servidor de Akamai.

Akamai es un ejemplo de CDN exitosa que se usa como infraestructura de distribución de contenidos de una enorme cantidad de empresas que disponen de una gran variedad de contenidos.

Arquitectura Ninja

La arquitectura *Ninja* [5] es un ejemplo en el cual se ataca lo que es generalmente un punto débil de las CDN, esto es, la escalabilidad. La arquitectura Ninja describe una red compuesta por cuatro tipos de elementos: *bases*, que son clusters de workstations en los cuales se ejecuta una capa de software que simplifica la construcción de los servicios que provee la red; *unidades*, que corresponden a diversos tipos de clientes (PCs, celulares, Palms, etc.) que consumen el contenido; *active proxies*, nodos encargados de procesar la información de modo de transformarla y adaptarla dependiendo del tipo de cliente y del servicio que se provee. La plataforma de software utilizada en las bases para facilitar la

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

[2] Algunos de los clientes de Akamai son: MySpace, Adobe, AUDI, Verizon entre otros.

creación de servicios es vSpace, la cual, sumada al hecho de que cada base es un cluster de workstations, provee la escalabilidad y robustez mencionadas.

TV-Anytime

Otro modelo de CDN es una red de servidores *TV-Anytime* [6], se trata de un servicio que permite almacenar contenido multimedia en servidores para que los clientes puedan acceder a los streams en cualquier momento. El objetivo principal de este servicio es encontrar una manera de mapear el contenido multimedia en los servidores de modo tal de proveer el contenido a los clientes con la mayor calidad posible. En este caso se obtiene la mayor calidad teniendo en cuenta las restricciones de capacidad de almacenamiento de los servidores, ancho de banda, etc. Para la resolución de este problema se utilizan métodos heurísticos dado que se trata de un NP-completo. Se han desarrollado heurísticas para resolver este problema, por ejemplo una red jerárquica de servidores con Parallel Simulated Annealing.

2.2. Redes P2P

2.2.1. Introducción a las redes P2P

Una red P2P está constituida un conjunto de nodos que se comportan simultáneamente como clientes y servidores para los demás nodos de la red (ver Figura 2.4).

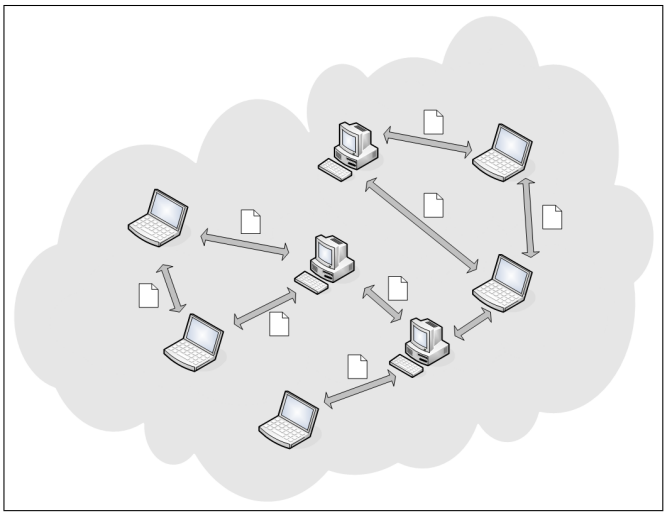


Figura 2.4: En las redes P2P los clientes también actúan como servidores

Actualmente existen diferentes arquitecturas para redes P2P:

- Centralizada: los nodos de la red solicitan a un servidor central para encontrar otros nodos que contenga el contenido deseado. Este tipo de enfoque no es muy escalable.

- Descentralizada pero estructurada: aquí no se cuenta con un servidor central de directorio pero se tiene una estructura bastante rígida, es decir, que la topología de red está muy controlada y el contenido no está distribuido de forma aleatoria.
- Descentralizada pero no estructurada: redes donde no se tiene mucho control de la topología así como no se tiene ningún servidor central de conocimiento. La topología de este tipo de redes tienen ciertas características pero el "placement" del contenido no está basado en ningún conocimiento de la topología de la red, por lo tanto, para buscar cierto contenido, un nodo se comunica con sus vecinos. El método más común es el *flooding*.

Los elementos básicos que se pueden identificar en una red P2P en la cual se realizará streaming son los siguientes: *peers*, *streams* y archivos multimedia. También se reconoce la necesidad de que algunos peers sean *seeds*, nodos particulares pertenecientes al proveedor de servicios a los que se conectan los clientes para iniciar el streaming. La idea principal de esto es que cada peer contribuya con una porción del ancho de banda usado en el streaming.

Actualmente existen implementaciones de streaming sobre una red P2P: un ejemplo de esto es GnuStream [7], un prototipo que está construido sobre la popular plataforma P2P Gnutella. Entre las principales características de GnuStream tenemos que los peers pueden ser ubicados dinámicamente, se consigue una buena distribución de la carga y una mayor velocidad de reacción a la capacidad disponible y al estado (online/offline) de los peers.

Una de las características a tener en cuenta en un servicio de streaming de video es que cada usuario pueda elegir la calidad (bitrate) del video dependiendo de sus limitaciones (ancho de banda, capacidad de procesamiento, etc.). Esto se provee por ejemplo en MTcast [8], un servicio de streaming de video sobre P2P que permite esto, usando a los peers para decodificar y redirigir el stream de video a otros usuarios. MTcast es un buen ejemplo de cómo las redes P2P atacan el problema de la escalabilidad, visto en las CDN.

2.2.2. Ejemplos de redes P2P

A continuación se presenta un relevamiento realizado sobre redes P2P, tanto propuestas, prototipos, o implementaciones que se encuentran actualmente en funcionamiento.

CoopNet

Cooperative Networking [9] (*CoopNet*) busca operar en una situación de alta dinámica donde clientes individuales pueden sólo participar por algunos minutos. Asimismo se busca resolver el problema del streaming en vivo que se refiere a la distribución sincronizada de contenido de streaming a uno o más clientes. Un árbol de distribución con el servidor como raíz es formado con los clientes como sus miembros. Cada nodo del árbol transmite el stream recibido a cada uno de sus hijos usando unicast. El servidor codifica la señal AV en M descripciones (según MDC) y transmite las descripciones a través de M diferentes árboles de distribución, cada uno con el servidor como raíz.

Aparición de un nuevo nodo: en primer lugar, el nuevo cliente contacta al servidor informándole su ancho de banda. Luego, el servidor selecciona los

padres, uno por árbol de distribución, como se describe a continuación: empezando por el servidor, se comienza a bajar por el árbol hasta encontrar un nivel donde haya uno o más nodos que permitan servir al nuevo nodo, es decir, que contengan la capacidad necesaria o ancho de banda. El servidor finalmente elige uno de los posibles nodos de forma aleatoria.

Desaparición de un nodo: hay dos posibles situaciones, una es la que voluntariamente un nodo decide irse por lo tanto avisa al servidor de tal hecho y el servidor realiza el procedimiento de *aparición de un nuevo nodo* para cada uno de los hijos. La otra situación es que haya una falla de algún tipo. Cuando esto sucede, los hijos del nodo que tuvo la falla, comenzarán a experimentar una tasa muy grande de pérdidas de paquetes por lo tanto tratará de contactar a su padre para saber si él también experimenta este problema. Si no es posible contactarlo, se sabe que estamos en presencia de falla por lo tanto este nodo, contacta al servidor como si fuera un nuevo nodo que intenta formar parte del sistema.

El corto tiempo de vida de los nodos individuales implica un desafío en la búsqueda distribuida de esquemas como CAN, Chord, Pastry y Tapestry. Trabajos en ALM como ALMI, End System Multicast y Scattercast están directamente relacionados al aspecto del streaming en vivo. Sistemas como SpreadIt, Allcast y vTrails comparten el mismo espíritu que CoopNet, es decir, entregar contenido mediante streaming utilizando un enfoque peer-to-peer.

CoolStreaming / Donet

Otro ejemplo de streaming en vivo se presenta en *CoolStreaming/Donet* [10] (Data-driven Overlay Network). Las operaciones principales son muy simples: cada nodo periódicamente intercambia información con un conjunto de vecinos y obtiene información no disponible de uno o más vecinos o suministra datos disponibles a sus vecinos.

Se enfatizan tres características importantes:

1. Fácil implementación, no se mantienen estructuras globales complejas.

2. Eficiencia, el *forwarding* de datos es dinámicamente determinado según la disponibilidad de los datos y no restringida por direcciones específicas.

3. Robustez y resistencia, la cooperación permite el intercambio adaptable y rápido entre múltiples suministradores.

SplitStream

El application-level multicast (ALM) se ha convertido en una interesante alternativa del IP multicast. En el caso de redes P2P o ambientes cooperativos el convencional multicast basado en árboles no es del todo adecuado. La razón es que la tarea de duplicar y compartir el trafico "multicast" es llevada a cabo por un conjunto pequeño de nodos que son interiores al árbol. La mayoría de los nodos son hojas y no contribuyen con sus recursos. El problema es todavía mayor en aplicaciones de alto ancho de banda como distribución de video. *SplitStream* [11] intenta resolver estos problemas.

La idea clave es dividir (*split*) el contenido en k partes y realizar el multicast de cada parte usando un árbol separado. Los nodos se unen a tantos árboles como partes deseen recibir y especifican un límite superior en la cantidad de partes que están dispuestos a transmitir. El desafío es construir un bosque de árboles "multicast" de forma que un nodo interior en un árbol sea hoja en los árboles restantes y que las restricciones de ancho de banda se satisfagan. Dicho de otra forma, lo que se quiere es distribuir la carga del pasaje de la información sujeto a las restricciones de ancho de banda de los nodos participantes en forma descentralizada, escalable, eficiente y auto-organizada. Se considera la implementación de SplitStream usando Scribe y Pastry. Scribe es un sistema de comunicación de grupos a nivel de aplicación y Pastry es una red P2P estructurada, auto-organizada y escalable.

Chainsaw

Chainsaw [12] es otro sistema "multicast" sobre P2P similar al SplitStream que intenta eliminar los árboles. Los nodos son notificados de nuevos paquetes por sus vecinos y deben explícitamente pedir un paquete a un vecino para recibirlo. De esta forma datos duplicados pueden ser eliminados y un nodo puede asegurar que recibe todos los paquetes. Se afirma que mediante simulaciones se pudo demostrar que el sistema es capaz de diseminar información con alta tasa a un gran número de nodos sin pérdida de paquetes y extremadamente baja duplicación de datos. Asimismo un nuevo nodo puede comenzar a bajar información y reproducirla en una fracción de segundo luego de unirse a la red, haciéndolo muy adecuado para aplicaciones de streaming bajo demanda. Finalmente se afirma que el sistema es robusto a fallas catastróficas, por ejemplo, la mayoría de los nodos fueron capaces de obtener información a pesar de que la mitad de los nodos de la red fallaran simultáneamente.

Avalanche

En *Avalanche* [13] se propone un esquema para la distribución de grandes archivos basado en codificación de red (*network coding*). En este esquema, cada nodo de la red de distribución es capaz de generar y transmitir bloques codificados de información. Este enfoque no requiere conocimiento de la topología de la red y además los nodos eligen como propagar paquetes basados en la información local solamente. Se muestra que el sistema es muy robusto a situaciones extremas con partidas de nodos. Utilizando codificación de red, los nodos son capaces de hacer progresos y terminar una bajada aun si el servidor se va poco después de subir sólo una copia del archivo y los nodos dejan la red inmediatamente después de terminar sus bajadas.

CollectCast (PROMISE)

Se presenta *PROMISE* [14], un sistema P2P de streaming de video que se basa en un servicio P2P llamado *CollectCast* [15] que opera en la capa de aplicación pero infiere y explota propiedades de la red. Este servicio realiza funciones de selección de emisores, monitoreo e intercambio. La arquitectura de PROMISE consiste en un conjunto de nodos interconectados a través de un sustrato P2P. Este sustrato mantiene la conectividad entre los nodos, administra

los nodos ("membership") y realiza la búsqueda o "lookup" de los objetos. Las operaciones de PROMISE son independientes del sustrato P2P bajo la cual se encuentra. Por lo tanto, PROMISE puede ser implementado arriba de sustratos P2P como Pastry, Chord y CAN (en el prototipo se utilizó Pastry).

Nemo

Nemo [16] es un protocolo multicast para redes P2P que intenta optimizar el manejo del alto grado de transitoriedad de los nodos. Se basa en dos técnicas:

1. co-líderes
2. enviar acuses negativos (NACKs)

Este enfoque esta pensado para aplicaciones de streaming de audio y video en tiempo real que se pueden beneficiar de altas proporciones de entrega dentro de límites de latencia específicos, sin requerir de perfecta confiabilidad; este modelo suele ser llamado multicast resistente. Los nodos participantes se organizan en *clusters* basados en la proximidad de la red, con cada nodo siendo miembro de algún cluster en el nivel más bajo. Los clusters varían de tamaño entre d y $3d$–1, donde d es una constante llamada "grado" (*degree*). Cada cluster selecciona un líder que se convierte en miembro de la capa inmediatamente superior. Para evitar la dependencia de un solo nodo, cada líder de los clusters recluta un número de co-líderes para formar su equipo. Este proceso se repite, es decir, todos los nodos de una capa se agrupan en clusters, se forman equipos y los líderes son promovidos para participar en la próxima capa superior.

La dinámica del comportamiento de esta red basada en árbol como la entrada y partida de nodos se describe como sigue: Un nuevo nodo se une al grupo "multicast" consultando a un nodo conocido por los identificadores de los miembros de la capa superior. Comenzando desde ahí y de forma iterativa, el nodo entrante continúa primero solicitando una lista de los miembros de la red actual al líder del cluster y luego seleccionando entre ellos a quien contactar luego basado en la proximidad de la red y por último moverse a la siguiente capa. Cuando el nuevo nodo encuentra la más cercano líder en la capa inferior, se une a ese cluster. Por otro lado, los miembros pueden dejar Nemo de forma anunciada o no. Un miembro común solamente informa a su líder de su partida. Pero un líder en cambio, debe primero elegir un reemplazo para todos los clusters que tiene y luego informar a su líder antes de irse. Para detectar partidas no anunciadas, Nemo monitorea la comunicación entre los nodos de los clusters. Después de un período de gracia, miembros que no se reportaron son considerados muertos y un algoritmo de reparación es iniciado. Si el nodo que falló es un líder, el árbol mismo debe ser reparado: los miembros del cluster víctima deben seleccionar entre ellos un nuevo líder. Para tratar con la dinámica de la red, cada nodo periódicamente verifica los líderes de las capas superiores e intercambia clusters si algún líder se encuentra más cercano que el actual. Como se mencionaba antes, la cardinalidad de los clusters tiene ciertos límites definidos por la constante de grado (d), por lo tanto debido a cambios en la red, los clusters pueden crecer o encogerse más allá de estos límites. Esto converge a las operaciones de combinación ("merge") y división ("split") de los clusters. La aplicación de estas operaciones sigue un enfoque probabilístico periódico de forma de no saturar la

red.

En caso de pérdida de paquetes, Nemo tiene un mecanismo de retransmisión de datos. Este se basa en números de secuencia y envío de NACKs.

BitTorrent

BitTorrent [17] es un método de distribución de archivos P2P con la particularidad importante de que en lugar de utilizar una red P2P en la que se congrega un gran número de usuarios (cada uno de los cuales comparte diferentes archivos), BitTorrent crea una sesión de transferencia (un *torrent*) por cada contenido. Si bien con esto se pierde la posibilidad de localizar contenido en la red, por otro lado mejora la capacidad de transferencia. Las principales características de BitTorrent son las siguientes:

- Cada archivo está dividido en *piezas*, y cada pieza está a su vez dividida en *bloques*. El bloque es la unidad de transmisión de la red (generalmente 256KB), pero el protocolo solamente toma en cuenta las piezas transferidas (un nodo no puede servir una pieza a otros hasta que ésta no esté completa).
- Cada nodo mantiene una lista de nodos "vecinos" (*peer set*), es decir nodos a los cuales potencialmente puede enviarles piezas. El subconjunto del peer set con el cual el nodo está efectivamente transfiriendo piezas, se denomina *active set*.
- El único componente centralizado de BitTorrent es el *tracker* de cada torrent, el cual lleva un registro de todos los nodos involucrados en la distribución del archivo (los nodos reportan su estado al tracker cada 30 segundos o al desconectarse, mientras que cuando un nodo se conecta utiliza el tracker para encontrar otros nodos que participan en la distribución del archivo).

BitTorrent es extremadamente popular actualmente en lo que se refiere a distribución de archivos vía P2P, existiendo una gran cantidad de clientes[3], así como un número importante de sitios para localización de torrents.

Bullet

Bullet [18] es presentado como un algoritmo distribuido para organizar los nodos de una red en una estructura del tipo *mesh* de modo tal de maximizar el ancho de banda distribuido a todos los nodos que participan de la red. Se pretende argumentar por medio de Bullet que con una estructura del tipo mesh se pueden lograr mejores resultados en cuanto a la distribución del ancho de banda a través de la red que con un árbol.

ESM

ESM [19] (*End System Multicast*) es un sistema para realizar broadcast de audio y video que trabaja sobre una red del tipo P2P. ESM está constituido por tres componentes: broadcaster (cuya tarea es distribuir el contenido), viewer

[3] BitComet, Azureus, uTorrent son los más populares

(para reproducir el audio/video) y la red sobre la cual funcionan estos componentes. El tipo de contenido que puede ser distribuido por medio de ESM es muy variado, tanto VoD, como broadcast en tiempo real o teleconferencias. Los nodos en ESM están organizados en forma de árbol, con la particularidad de que cada nodo monitorea su rendimiento (latencia y ancho de banda) con respecto a otros hosts, y ajusta su posición en el árbol cuando determina que eso mejorará la distribución. Para tener en cuenta la heterogeneidad de los recursos de los que disponen los clientes, ESM está diseñado para que se transmitan varias versiones del stream con diferentes calidades (actualmente hay 2, de 100kbps y 300kbps respectivamente). Todos los nodos comienzan recibiendo el stream de más alta calidad, pero cuando se empiezan a detectar pérdidas importantes, se cambia a un stream de menor calidad para adaptarse a los recursos de los que dispone el cliente.

IceCast / IceShare

IceShare [4] es una *biblioteca* utilizada para distribuir streams del tipo *Ogg* (tanto audio como video) en una red "pseudo P2P". Está basado en BitTorrent, y permite la distribución del contenido tanto en forma de archivos como streams continuos. En el caso de IceShare no hablamos de un P2P "puro", sino que se dispone de servidores *IceCast* dedicados, pero para aligerar la tarea de éstos los reproductores de video de los clientes pueden (por medio de la biblioteca IceShare) distribuir contenido a otros. Al igual que BitTorrent, para cada stream que se distribuye hay un tracker (*IceTracker*), que mantiene la información de todos los nodos que están participando en la distribución de un cierto contenido. Una característica interesante de IceShare es que brinda la posibilidad de streams alternativos de distintas calidades/bitrates, de ese modo los clientes con más ancho de banda pueden recibir los streams con mayor bitrate, aunque no está explicitado si esto se logra mediante la distribución de varios streams o un solo stream que es recodificado según sea necesario en algún nodo intermedio.

MTcast

El objetivo de *MTcast* [8] (*Multiple Transcode based video multicast*) es brindar un servicio de streaming de video a usuarios *heterogéneos* en lo que respecta a recursos y limitaciones (por ejemplo, algunos usuarios tendrán mas ancho de banda o capacidad de procesamiento que otros). La idea es que cada usuario seleccione un cierto *nivel de calidad* y MTcast se encargue de que el usuario reciba el stream de video en una calidad igual o cercana a la elegida. En esta propuesta, cada nodo se encarga de hacer el transcoding necesario sobre el stream para enviarlo a otros nodos con la calidad que éstos hayan elegido. MTcast consiste en un árbol de distribución (*transcode tree*), con el nodo que distribuye el stream en la raíz, y los nodos con requerimientos de calidad más elevados se ubican más próximos a la raíz. La ubicación de los nodos se determina a partir de su capacidad de procesamiento y ancho de banda (uplink y downlink). La idea es que los nodos hagan el transcoding para obtener un video con menor bitrate que el que reciben y lo transmitan a los nodos de niveles inferiores. Para esto los nodos con requerimientos de calidad similares se agrupan en *capas*, con el fin de que todos los nodos pertenecientes a una misma capa reciban el stream con la

[4] http://wiki.xiph.org/index.php/IceShare

misma calidad. De este modo, un nodo entrante puede encontrar una capa con un nivel de calidad cercano al que solicitó y ubicarse allí para recibir el stream.

Overcast

Overcast [20] propone *multicast* a nivel de aplicación, implementado como una red de tipo *overlay* (al igual que en el caso de Bullet). Es decir, los nodos se ubican de cierto modo en una red física de manera de crear una abstracción de la red. La estructura de la red de Overcast es del tipo de *árbol de distribución*, y en ese sentido Overcast también tiene en cuenta la dinámica de los nodos ya que define un protocolo para manejar las entradas de nuevos nodos a la red. La red está compuesta por: una fuente (la cual puede estar replicada), *nodos internos* de Overcast y *clientes HTTP*. La particularidad de Overcast es que de acuerdo al protocolo que utiliza esta red, el árbol de distribución se adapta a los cambios en la red subyacente sobre la cual está implementada.

PbCast

PbCast [21] es un protocolo multicast que significa "difusión probabilística". Este enfoque satisface las siguientes propiedades:

Atomicidad: el protocolo provee garantía de entrega bimodal, bajo la cual existe alta probabilidad que cada multicast alcanzará casi todos los procesos, a baja probabilidad que un multicast alcance solo un pequeño conjunto de procesos y una despreciable probabilidad que alcance un número intermedio de procesos. La tradicional garantía "todo o nada" se convierte en "casi todo o casi nada".

Estabilidad de rendimiento: la variación esperada en el rendimiento puede ser caracterizada y para las configuraciones que interesan en el protocolo, es bajo comparado a tasas típicas de multicast.

Ordenamiento: los mensajes son entregados en modalidad FIFO.

Estabilidad de multicast: el protocolo detecta estabilidad de mensaje, significando que la garantía de entrega bimodal ha sido lograda. Un mensaje estable puede ser recolectado para su eliminación de forma segura y si es deseado, la capa de aplicación también será informada.

Detección de mensajes perdidos: la entrega bimodal admite una pequeña posibilidad de que algunos multicasts no alcancen algunos procesos y cuando pasan estas pérdidas de mensajes, los procesos que no recibieron un mensaje son informados.

Escabilidad: los costos son constantes y crecen lentamente en función del tamaño de la red.

2.2.3. Clasificación de las redes P2P analizadas

A modo de síntesis del relevamiento realizado acerca de las redes P2P, se presenta en la Tabla 2.1 una clasificación de éstas según ciertos parámetros que las caracterizan. Los criterios de clasificación escogidos fueron los siguientes:

Modelo de descentralización

Las redes P2P presentan una característica que las hace distintivas de otros tipos de redes: la descentralización. La idea de compartir para lograr un mismo objetivo es algo que todas estas redes presentan. Sin embargo, hay distintos grados de descentralización, dependiendo de la cantidad de nodos que se comporten o realicen las mismas tareas. Estos grados se pueden clasificar en tres modelos:

Modelo Puro: este modelo es completamente distribuido y presenta la pureza en el sentido de que todos los nodos se comportan igual, y que tienen las mismas responsabilidades y roles.

Modelo Híbrido: este es el modelo menos descentralizado, es similar al clásico modelo cliente-servidor. En nuestro contexto, se corresponde a tener un nodo maestro, conocedor de toda la información de la red y que decide como distribuir el contenido.

Modelo Jerárquico: este modelo presenta varios nodos servidores, como podría verse en algún híbrido P2P-CDN. La idea básica es combinar las fortalezas de los dos modelos anteriores, es decir, la eficiencia de obtener la información de un nodo maestro con un buen balance de distribución y robustez de los sistemas descentralizados puros.

Topología

Otra clasificación importante de la redes P2P está relacionada a su topología u forma de organizar la red. Básicamente se encuentran dos:

Estructurada: se mantiene la red organizada, manteniendo un tipo de estructura de grafo que básicamente se tratan de árboles. De esta forma se sabe con certeza a qué nodos se debe distribuir qué contenido. A pesar de diseñarse este tipo de redes para mejorar la eficiencia de la distribución de contenido, muchas veces es costoso mantener este tipo de estructuras en redes de gran tamaño y con mucho dinamismo.

No estructurada: los nodos se organizan de forma más bien aleatoria, no se mantiene ninguna estructura global como puede ser un árbol o un grafo estructurado. Se utiliza básicamente inundación de contenido. Un ejemplo podría ser que luego de un cierto tiempo, cada nodo envía a sus vecinos información sobre que paquetes tiene, de forma que cada nodo solicite a sus vecinos, según su preferencia, los paquetes que necesita.

Tipo de codificación

En el contexto de las redes de distribución de video suele también identificarse el tipo de codificación que se utiliza para el contenido a distribución. Se distinguen los dos tipos siguientes:

Unistream: el contenido se envía completo.

Multistream: el contenido se divide en partes o substreams de forma de balancear la carga y mitigar pérdidas de paquetes así como retrasos que en tiempo real son importantes atacar.

	Modelo			Topología		Codificación	
	P	H	J	E	NE	U	M
CoopNet		x		x			x
CoolStreaming	x				x	x	
SplitStream				x			x
Chainsaw	x			x		x	
Avalanche		x			x		x
CollectCast	x			x		x	
Nemo			x	x		x	
BitTorrent	x				x		x
Bullet	x				x	x	
ESM	x			x			x
IceCast			x		x		
MTCast	x			x		x	
Overcast			x	x		x	

Cuadro 2.1: Clasificación de redes P2P

2.3. Redes P2P como alternativa a las CDN

2.3.1. Desventajas de las CDN

Si bien el modelo de distribución de contenido utilizado por las CDN ha sido exitoso en muchos casos, existen problemas intrínsecos a ese tipo de arquitecturas que nos llevan a plantear nuevas alternativas para la distribución. A continuación listamos las desventajas más importantes que se encuentran en las CDN tradicionales:

- A medida que un servicio de distribución de contenido capta más y más clientes, se va haciendo progresivamente difícil la distribución, ya que al haber más clientes, se cuenta con menos recursos disponibles (ancho de banda, servidores, etc.). Las soluciones en este caso serían: cambiar la arquitectura subyacente de la red para optimizar el método de distribución, lo cual sea seguramente engorroso y difícil, o de lo contrario comprar más ancho de banda o más servidores para aumentar la cantidad de recursos disponibles, pero esto tiene su costo. Por estas razones, las CDN no son naturalmente escalables, en el sentido que el incremento de clientes siempre va a ocasionar gastos en la empresa que maneja el servicio. En relación a esto, siempre que en una CDN se requiera aumentar el ancho de banda disponible para los clientes, o la capacidad de procesamiento de los servidores, éstos van a tener que adquirirse, ya sea el hardware o el ancho de banda. Por ejemplo, si los nuevos videos que va a transmitir un servicio tienen un bitrate mayor que los anteriores, deberá asegurarse que la CDN tenga el ancho de banda suficiente para transmitirlos, lo cual tiene su correspondiente costo. Por lo tanto, los costos son otro problema intrínseco de este tipo de redes.

- Otro problema a mencionar son los puntos de falla: si un servidor falla, los clientes que estaban recibiendo el contenido de ese servidor dejan de recibirlo hasta que la situación se arregle. Si bien muchas empresas atacan este problema por medio de mirrors y clusters de servidores, pero incluso esto puede no funcionar en caso de fallas masivas.

- Por último, la distribución de los servidores de la CDN puede afectar la distribución en el sentido que algunos clientes pueden estar ubicados

muy lejos de los servidores de la CDN, lo cual provoca que dichos clientes tengan peor latencia.

2.3.2. Uso de redes P2P para distribuir contenido

Vistas las desventajas de las CDN mencionadas anteriormente, se plantea la posibilidad de utilizar las redes P2P para distribuir el contenido (en el caso particular de este trabajo, el contenido son streams de video en vivo). A continuación veremos qué características de las redes P2P ayudan a combatir estos problemas de las CDN, de modo tal de darnos cuenta por qué las P2P constituyen una alternativa interesante a las CDN:

- Al actuar los clientes como servidores, el ancho de banda disponible en el servicio de distribución de contenido está ligado directamente al ancho de banda disponible en los clientes. En consecuencia, al aumentar la cantidad de clientes de la red, aumenta el ancho de banda disponible en la red, por lo cual una solución basada en P2P debería tener buena escalabilidad. En cuanto a los costos, la situación es similar: no es necesario contar con grandes cantidades de ancho de banda ni clusters enormes de servidores, simplemente con unas pocas fuentes que se encarguen de la distribución inicial del contenido a algunos clientes, y éstos a su vez lo transmitirán a los demás. Además, recordemos que al aumentar la cantidad de clientes aumenta el ancho de banda disponible, por lo cual no existe la misma urgencia de contratar más ancho de banda que en las CDN. Por lo tanto, los costos de mantener una red P2P de distribución de contenido son significativamente menores que para una CDN.

- Cuando un nodo falla o se desconecta, los clientes que estaban recibiendo contenido de dicho nodo pueden recibirlo de cualquier otro nodo de la red que disponga de ese contenido y tenga ancho de banda disponible para transmitirlo.

- Finalmente, al actuar los clientes también como servidores, se incrementa la posibilidad de recibir el contenido de fuentes más cercanas que en el caso de una CDN, lo cual ayudaría a combatir el problema de la latencia mencionado anteriormente.

Hemos visto entonces que una solución basada en P2P es atractiva, ya que ataca las principales debilidades de una CDN. Por eso, es viable presentar un modelo de una red de distribución de video en vivo basado en este tipo de arquitectura. Ahora bien, las redes P2P tienen a su vez una serie de problemas que deben ser tenidos en cuenta si se tienen intenciones serias de construir un prototipo funcional para una red de ese tipo.

2.3.3. Problemas y restricciones de las redes P2P

- En primer lugar, hay que notar que en las redes P2P la mayor limitante está en el ancho de banda de subida (*upstream bandwidth*) de los nodos: en efecto, si bien vimos que puede resultar ventajoso en ciertos aspectos que sean los mismos clientes los encargados de la distribución, también hay que tener en cuenta que los clientes por lo general no disponen de las mismas

cantidades de ancho de banda de subida que un servidor dedicado de una CDN. Esto es particularmente problemático en el caso de la distribución de un stream de video, ya que si el stream tiene un bitrate muy elevado, puede que no haya muchos clientes en la red con los recursos suficientes para transmitirlo. En la Sección 2.4 se presenta una técnica que será utilizada en el modelo propuesto para intentar que esta limitante no imposibilite el funcionamiento de nuestra red.

- El otro problema fundamental de las redes P2P está en la *dinámica* de los nodos: en las redes P2P las conexiones y desconexiones de nodos son *muy* frecuentes, y cada desconexión de un nodo implica que los clientes que estaban recibiendo contenido de él ya no lo recibirán (podemos decir que quedan "colgados"), mientras que cada nodo nuevo que se conecta debe comenzar a recibir el contenido de otro cliente. Todo esto implica que la estructura de la red está cambiando constantemente, y debemos estar atentos a esto para que los clientes del servicio reciban el contenido adecuadamente. Este es el problema principal al que nos enfrentamos en este trabajo; en el Capítulo 3 la dinámica de los nodos se presenta como un problema matemático formal, mientras que en el Capítulo 4 se propone una serie de algoritmos para resolverlo.

2.4. Streaming Multifuente

Dado que trabajaremos en el contexto de una red P2P, es importante tener en cuenta el hecho de que los clientes pueden a su vez actuar como servidores; esto puede ser un inconveniente si tenemos en cuenta que la mayor parte de estos clientes cuenta con conexiones a Internet hogareñas, las cuales son relativamente limitadas, particularmente en lo que a ancho de banda de subida se refiere[5]. En consecuencia, el ancho de banda de subida de los clientes es la mayor limitante a la hora de concebir una red de distribución de video en vivo basada en P2P: en efecto, consideremos que el bitrate del video a transmitir puede ser mayor que el ancho de banda de subida de la mayor parte de los clientes de la red, en tal caso ninguno de ellos podrá transmitirlo a otros. Es por eso que surge la necesidad de plantear una arquitectura para esta red que nos ayude a combatir el límite impuesto por el ancho de banda de los clientes. En [2] se presenta una técnica llamada *Streaming Multifuente* con el objetivo de mejorar la distribución de video en una red de este tipo. A grandes rasgos, el streaming multifuente puede ser explicado de la siguiente manera: si consideramos el stream de video a transmitir como una secuencia de cuadros (*frames*), dividimos este stream en varios "substreams", cada uno de los cuales contendrá solamente algunos de los frames del stream original (se puede considerar el uso de redundancia en esta división, es decir, que un mismo cuadro esté presente en varios substreams). De este modo, cada uno de estos substreams requerirá menos ancho de banda que el stream original para ser transmitido por los clientes. En cuanto a la recepción del video, se deduce de lo anterior que cada cliente recibirá varios substreams (posiblemente de varias fuentes), los cuales deben ser combinados para obtener el stream de video que será visualizado (ver Figura 2.5) . En un momento dado,

[5]Las conexiones ADSL son el ejemplo más claro de esto, siendo además en general la velocidad de subida menor que la de bajada

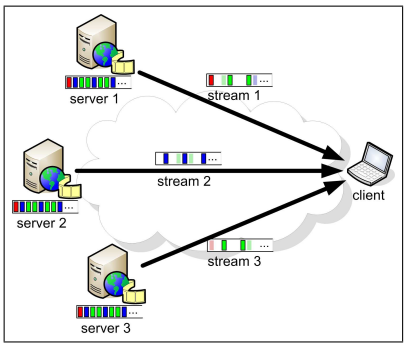


Figura 2.5: Streaming multifuente

la calidad del video que ve cada cliente depende de los substreams que esté recibiendo. En caso de estar recibiendo todos los substreams, es posible regenerar el stream original y la calidad será óptima. Si no se están recibiendo todos los substreams, la calidad variará según varios factores, pudiéndose producir cortes o artefactos en el video (no todos los cuadros del stream tienen la misma influencia sobre la calidad del video, referirse a [22] por más detalles acerca de esto). La división en substreams debe hacerse entonces de acuerdo a la capacidad de subida de los clientes para que éstos puedan actuar como servidores y de este modo transmitirle el video a otros clientes, aprovechando de este modo el ancho de banda de subida del que se dispone en la red, lo cual es uno de los lineamientos principales para una red P2P. En [22] se estudia la configuración óptima de cantidad de substreams y su redundancia por cada tipo de cuadro de forma de maximizar la calidad del video transmitido en una red P2P con dinámica en la conexión de los nodos. Por lo tanto, de acuerdo a lo visto en esta sección y en las anteriores, es viable elegir un modelo basado en P2P para una red de distribución de video en vivo, y el streaming multifuente ataca la principal limitante de este modelo, que es el ancho de banda de los clientes que participan de la red.

Capítulo 3

Modelo Matemático

En este capítulo se presenta la especificación formal para un modelo de asignación dinámica en una red P2P de streaming de video en vivo. Este modelo teórico, originalmente publicado en [23], es la base a partir de la cual se pueden desarrollar algoritmos de asignación para este tipo de redes. Si bien la complejidad y la gran cantidad de restricciones que se presentarán en el modelo impiden la construcción de algoritmos que generen asignaciones óptimas, los conceptos introducidos aquí pueden ser aplicados en heurísticas para obtener posibles soluciones.

3.1. Definición formal del modelo

3.1.1. Stream transmitido

Consideraremos un único stream de video transmitido con ancho de banda BW. Sin embargo, como vimos anteriormente, el ancho de banda de dicho stream puede ser demasiado grande para que los clientes lo retransmitan (recordemos que estamos en el contexto de una red P2P, por lo cual las conexiones de los clientes pueden no tener un ancho de banda importante, en particular de subida). Por lo tanto, el stream original se divide en K substreams,

$$\sigma_1, \sigma_2, ..., \sigma_K$$

siendo cada substream σ_k transmitido con un ancho de banda constante bw_k ($k = 1, 2, ..., K$). El ancho de banda BW del stream original está dado entonces por:

$$BW = \sum_{k=1}^{K} bw_k$$

La calidad del video recibido por cada cliente en un momento dado estará determinada entonces por los substreams que le estén siendo transmitidos en ese momento. Si recibe todos los substreams, entonces la calidad del video será óptima; de lo contrario, se podrán apreciar pérdidas en la calidad en mayor o menor medida (es importante aclarar también que no todos los substreams influyen del

mismo modo en la calidad del video[1]). Más adelante será introducida una medida formal de la calidad del video para expresar rigurosamente estos conceptos.

3.1.2. Nodos que conforman la red

La red que se modela está compuesta, en un instante dado, por un conjunto de nodos $\mathcal{N}$. Distinguimos en particular un nodo fuente s, que dispone del contenido a transmitir (los substreams que componen el stream original) y es el encargado de distribuirlo a una cierta cantidad de clientes[2]; luego, siguiendo la mecánica de las redes P2P, los clientes que reciben algún substream podrán a su vez transmitirlo a otros clientes. Para realizar esta transmisión, cada nodo $v \in \mathcal{N}$ cuenta con un ancho de banda de subida BW_v^{out}. Con respecto al ancho de banda de bajada, simplemente asumimos que cada cliente dispone de suficiente ancho de banda para recibir los K substreams, ya que de lo contrario estaría imposibilitado de ver el video en calidad óptima, y su participación en la red no sería entonces algo muy razonable.

3.1.3. Estructura de la red

En un instante dado, el estado de la red puede modelarse mediante un conjunto de K grafos dirigidos de la siguiente manera: para cada $k = 1, 2, ..., K$, sea $\mathcal{G}_k$ el grafo cuyo conjunto de nodos es $\mathcal{N}$ y una arista $(v_1, v_2) \in \mathcal{N} \times \mathcal{N}$ pertenece a $\mathcal{G}_k$ si y sólo si v_1 le está transmitiendo, o le transmitió en algún momento, el substream σ_k al nodo v_2, lo que también expresaremos a partir de ahora diciendo que v_1 es *padre* de v_2, o que v_2 es *hijo* de v_1. De este modo, cada $\mathcal{G}_k$ modela la transmisión del substream σ_k entre todos los nodos que componen la red. Es lógico además imponer el requerimiento de que no pueda haber ciclos en cada uno de estos grafos, ya que de lo contrario un nodo $v \in \mathcal{N}$ podría recibir un substream de otro nodo que había recibido dicho substream desde v[3]. Por otro lado, cada cliente necesita recibir cada uno de los substreams *desde un solo camino*, por eso también restringiremos el modelo de modo que cada nodo no pueda tener más de un padre para un mismo substream. Pero de acuerdo con estas restricciones, cada grafo $\mathcal{G}_k$ puede expresarse como una colección de árboles dirigidos disjuntos, digamos

$$\mathcal{G}_k = \{\mathcal{P}_k, \tau_{k,1}, \tau_{k,2}, ..., \tau_{k,M_k}\} \qquad k = 1, 2, ..., K$$

en donde $\mathcal{P}_k$ es el árbol cuya raíz es el nodo fuente s, al cual llamaremos *árbol principal*, mientras que los otros M_k árboles $\tau_{k,1}, \tau_{k,2}, ..., \tau_{k,M_k}$ se denominan *árboles desconectados*, debido a que sus nodos no están recibiendo el substream σ_k a través de un camino que parte desde el nodo fuente. El *estado ideal* de la red se define entonces cuando todos los nodos están recibiendo los K substreams, es decir, cuando no existen árboles desconectados, lo cual se puede expresar

[1] Puede haber substreams cuya ausencia no genere pérdidas notorias de calidad, pero también pueden existir otros que de no estarse transmitiendo hagan prácticamente imposible la visualización del video.

[2] En realidad sería razonable considerar la utilización de varios nodos fuente para mejorar la distribución, pero eso no afecta al modelo teórico, ya que únicamente nos interesará el ancho de banda combinado de los nodos fuente, por eso simplificamos el modelo con una sola fuente.

[3] No necesariamente en forma directa, sino que puede haber recibido el substream a través de un camino en el grafo desde v.

mediante la condición

$$M_k = 0 \qquad k = 1, 2, ..., K$$

En la Figura 3.1, podemos ver una representación de la red en un momento dado, tomando a todos los árboles correspondientes a cada substream como una "capa". Nótese el árbol principal $\mathcal{P}_k$ de cada substream, y también podemos ver allí como la desconexión de un nodo genera la aparición de árboles desconectados en las capas 1 y 2.

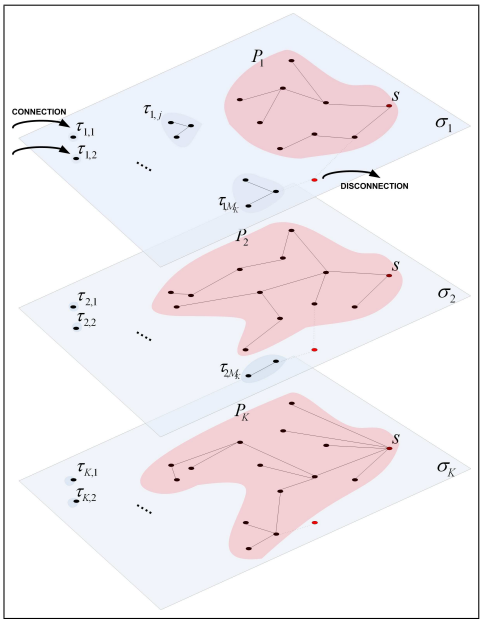


Figura 3.1: Estructura de la red vista en "capas"

3.1.4. Modelo de asignación dinámico

A través del tiempo, el estado de la red varía según la dinámica de los nodos: por un lado, nuevos nodos se conectan a la red para recibir el video, mientras que nodos que eran parte de la red se desconectan voluntaria o involuntariamente. Por lo tanto, para que los nuevos nodos comiencen a recibir el video, se les debe asignar un padre para cada substream que pueda transmitirle el contenido. Por otro lado, si un nodo se desconecta, todos los clientes que estaban recibiendo algún substream de dicho nodo ya no lo recibirán más, por lo cual se hace necesario volver a reconectarlos al árbol principal.

Discretización del tiempo

En primer lugar, observemos que no es viable pretender que la red reaccione ante cada conexión o desconexión de un nodo intentando realizar una asignación para conectar el nuevo nodo o reconectar los árboles desconectados que hayan quedado tras la partida de un nodo: en efecto, si así fuera, cada entrada o salida de un nodo implicaría la realización de nuevas asignaciones (y por lo tanto una reestructuración de la red); dada la alta frecuencia de conexiones y

desconexiones en una red P2P, es evidente que esto requeriría un procesamiento prácticamente constante. Por lo tanto, se divide el tiempo en intervalos de la forma

$$(t_n, t_{n+1}) \qquad n = 0, 1, 2, 3, ...$$

en donde:

$$t_{n+1} = t_n + \Delta$$

siendo Δ la longitud elegida para los intervalos. De este modo, el modelo dinámico de la red se hace sobre un conjunto discreto de puntos en el tiempo. Cada asignación de nodos se hace entonces luego de pasado un tiempo Δ desde la última asignación; a continuación detallaremos el estado y comportamiento de la red en estos intervalos de tiempo.

Estado en el inicio de un intervalo

Para $n \geq 1$, introducimos la notación t_n^- para representar el instante de tiempo inmediatamente anterior a t_n y t_n^+ para el instante de tiempo inmediatamente posterior. En cada t_n^-, la estructura de la red está dada por:

$$\mathcal{P}_k, \tau_{k,1}, \tau_{k,2}, ..., \tau_{k,M_k} \qquad k = 1, 2, ..., K$$

en donde el árbol principal $\mathcal{P}_k$ está formado por los nodos que están recibiendo el substream σ_k, y los árboles desconectados $\tau_{k,1}, \tau_{k,2}, ..., \tau_{k,M_k}$ corresponden a:

- Los nodos que ya no reciben el substream σ_k debido a la desconexión del nodo que se los estaba transmitiendo.
- Los nodos nuevos que se conectaron y están esperando recibir el substream σ_k (en este caso el árbol desconectado está compuesto solamente por el nuevo nodo).

Asignación de árboles desconectados

En el instante t_n, se ejecuta un algoritmo a partir del estado de la red en t_n^- con el objetivo de conectar los árboles desconectados al árbol principal. Desde el punto de vista del modelo teórico, el algoritmo se ejecuta instantáneamente en t_n. En la práctica, esto se traduce requiriendo que el tiempo de ejecución del algoritmo sea despreciable en comparación con el Δ elegido para el intervalo de tiempo. El funcionamiento básico del algoritmo será el siguiente:

1. En primer lugar, se determina el conjunto $\mathcal{P}$ de los nodos que pertenecen a algún árbol principal $\mathcal{P}_k$ y que tienen ancho de banda disponible para transmitir algún substream.
2. Luego, se determina el conjunto $\mathcal{C}$ de las raíces de los árboles desconectados. El objetivo es que algún nodo del árbol principal pueda transmitirles el substream, y de ese modo esos árboles desconectados volverán a unirse al nodo principal. Es importante destacar que el algoritmo no cambiará la estructura de los árboles desconectados (es decir, no se van a separar en subárboles que puedan ser asignados a otros padres), simplemente intentará reconectar los árboles desconectados al árbol principal[4].

[4]Recordemos que el algoritmo debe ejecutarse en un tiempo mínimo, por lo tanto estas simplificaciones son bienvenidas.

3. Finalmente, el algoritmo conecta las raíces de los árboles desconectados de $\mathcal{C}$ a los nodos de $\mathcal{P}$ de acuerdo a algun método de asignación a definir hasta que ocurra una de las dos situaciones siguientes:

   - Para cada uno de los nodos de $\mathcal{C}$ se ha asignados un padre correspondiente en $\mathcal{P}$; es decir, todos los árboles desconectados fueron reconectados al árbol principal.
   - Ningún nodo de $\mathcal{P}$ puede ser asignado como padre de alguna raíz de árbol desconectado que quede en $\mathcal{C}$ debido a que no tiene suficiente ancho de banda disponible.

Evidentemente, para una red con gran cantidad de nodos, puede haber una gran cantidad de maneras distintas de asignar los árboles desconectados. Más adelante se definirá las características que debe cumplir una asignación para ser óptima, de tal modo de fijar el objetivo a alcanzar en la eventual implementación de este algoritmo.

Dinámica de los nodos

Una vez que se reasignaron los subárboles desconectados en t_n, la red se comporta de la siguiente manera entre t_n^+ y t_{n+1}^-:

- Algunos nodos se desconectan, lo cual genera posiblemente nuevos árboles desconectados; es decir,

$$M_k(t_n^+) \leq M_k(t_{n+1}^-) \qquad k = 1, 2, ..., K$$

- Nuevos nodos se conectan a la red, generando entonces cada uno de ellos K árboles desconectados (uno por cada substream).

De este modo, el estado de la red en t_{n+1}^- será el punto de partida para una nueva ejecución del algoritmo de asignación.

Optimización

A la hora de describir el algoritmo, se mencionó que puede haber varias maneras distintas de conectar las raíces de los árboles desconectados del conjunto $\mathcal{C}$ a los nodos del conjunto $\mathcal{P}$. El objetivo es entonces definir las características que debe tener una solución óptima a este problema. En primer lugar, definimos una *asignación* como una terna ordenada,

$$(\upsilon_1, \upsilon_2, k)$$

en donde $\upsilon_1 \in \mathcal{P}$, $\upsilon_2 \in \mathcal{C}$ y $k \in \{1, 2, ..., K\}$. La asignación $(\upsilon_1, \upsilon_2, k)$ corresponde a conectar el árbol desconectado con raíz υ_2 con el nodo υ_1 para el substream k. Podemos definir entonces una *solución factible* $\mathcal{S}$ como un conjunto de asignaciones que cumple una de las dos condiciones siguientes:

- Todas las raíces de los subárboles desconectados de $\mathcal{C}$ tienen asignaciones con nodos de $\mathcal{P}$.
- Ninguno de los nodos de $\mathcal{P}$ cuenta con suficiente ancho de banda para transmitir los substreams correspondientes a los nodos de $\mathcal{C}$ que no han sido asignados.

Ahora debemos definir cuál de todas las soluciones factibles del problema es la óptima: a cada configuración de la red le asignaremos una *calidad percibida esperada global* Q; entonces, la solución óptima será aquella para la cual al aplicar las asignaciones correspondientes, la configuración de la red será tal que Q sea máxima. Es decir, el algoritmo debe realizar asignaciones con el objetivo de maximizar la calidad percibida esperada por el conjunto de los clientes del sistema.

3.1.5. Modelo de asignación estático

A continuación presentaremos formalmente las restricciones que debe cumplir una solución para ser óptima en un instante dado, es decir, desde un punto de vista estático. En este caso lo que se pretende es detallar las condiciones necesarias para que la calidad percibida global de la red en un instante dado sea óptima. Primero definiremos la notación utilizada para describir el modelo, luego veremos la función utilizada para medir la calidad percibida, y finalmente enumeramos las restricciones del modelo.

Definiciones preliminares

Recordemos que cada nodo $v \in \mathcal{N}$ tiene ancho de banda de subida BW_v^{out}, y el ancho de banda de cada substream σ_k es bw_k. Además, para todos $i, j \in \mathcal{N}$ y $k = 1, 2, ..., K$ definimos:

$$x_{i,j}^k = \begin{cases} 1 & \text{si el nodo } i \text{ transmite el substream } \sigma_k \text{ al nodo } j \\ 0 & \text{en caso contrario} \end{cases} \tag{3.1}$$

$$y_{i,j}^k = \begin{cases} 1 & \text{si el nodo } i \text{ precede al nodo } j \text{ en el substream } \sigma_k \\ 0 & \text{en caso contrario} \end{cases} \tag{3.2}$$

Medida de calidad instantánea

Para medir la calidad percibida por un cliente en un instante dado, utilizaremos una función PSQA (*Pseudo-Subjective Quality Assessment* [24] [25]). Según esta medida, la calidad percibida por un nodo en un momento dado depende de los substreams que el cliente esté recibiendo en ese momento. Formalmente,

$$\mathsf{PSQA}(v) = f(y_{s,v}^1, y_{s,v}^2, ..., y_{s,v}^K) \qquad \forall v \in \mathcal{N}$$

Luego, la *calidad global percibida* por los clientes para un estado de la red en un momento dado puede ser definida como el promedio de la calidad percibida por cada uno de los clientes, es decir:

$$\overline{\mathsf{PSQA}} = \frac{1}{|\mathcal{N}|} \sum_{v \in \mathcal{N}} \mathsf{PSQA}(v)$$

Asumimos además que el valor del PSQA está normalizado entre 0 y 1.

Restricciones para el modelo de asignación estático

Ahora que definimos una medida de calidad global, estamos en condiciones de especificar formalmente las restricciones que debe cumplir el modelo de

asignación estático.

$$s* = \arg\max_{s \in \mathcal{S}} (\overline{\mathsf{PSQA}}(s)) \tag{3.3}$$

La calidad global percibida debe maximizarse, esto significa que estamos tomando la solución $s*$ con mayor calidad global percibida de entre todas las del conjunto $\mathcal{S}$ soluciones factibles.

$$y^k_{i,j} + y^k_{j,i} \leq 1 \qquad \forall i, j \in \mathcal{N}, \quad k = 1, 2, ..., K \tag{3.4}$$

No puede haber ciclos en los grafos correspondientes a cada substream.

$$y^k_{i,j} = x^k_{i,j} + \sum_{v \in \mathcal{N}} y^k_{i,v} x^k_{v,j} \qquad \forall i \in \mathcal{N}, \quad \forall j \in \mathcal{N} - \{s\}, \quad k = 1, 2, ..., K \tag{3.5}$$

$y^k_{i,j}$ corresponde a la existencia de un camino desde i hasta j en el árbol del substream σ_k.

$$\sum_{v \in \mathcal{N}} x^k_{v,i} \leq 1 \qquad \forall i \in \mathcal{N}, \quad k = 1, 2, ..., K \tag{3.6}$$

Cada nodo tiene a lo sumo un padre por cada substream (es decir, no recibe un mismo substream de varios padres distintos).

$$\sum_{k=1}^{K} \left(bw_k \sum_{v \in \mathcal{N}} x^k_{i,v} \right) \leq BW^{out}_i \qquad \forall i \in \mathcal{N}, \quad k = 1, 2, ..., K \tag{3.7}$$

Un nodo no puede transmitir una cantidad de substreams que requiera más ancho de banda del que dispone.

$$x^k_{i,j}, y^k_{i,j} \in \{0, 1\} \qquad \forall i, j \in \mathcal{N}, \quad k = 1, 2, ..., K \tag{3.8}$$

3.1.6. Agregando condiciones iniciales al modelo

Podemos agregar más restricciones al modelo presentado anteriormente si consideramos lo expresado acerca de los árboles desconectados: recordemos que el algoritmo de asignación debe mantener la estructura de la red, y simplemente intentar reconectar los árboles desconectados al árbol principal. Podemos fijar entonces las asignaciones preexistentes como restricciones del modelo.

Configuración inicial de la red

Para $k = 1, 2, ..., K$, sea E^k el conjunto de las asignaciones existentes en el substream σ_k al momento de ejecutar el algoritmo. En otras palabras, decimos que la *configuración inicial* de la red es

$$\left(\mathcal{N}, E^1, E^2, ..., E^K\right)$$

Restricciones para el modelo de asignación estático con condiciones iniciales

En este modelo se deben cumplir todas las restricciones del modelo de asignación estático, y además se añaden las asignaciones preexistentes:

$$x^k_{i,j} = 1 \qquad \forall i, j \in \mathcal{N}, \quad k = 1, 2, ..., K \mid x^k_{i,j} \in E^k \tag{3.9}$$

3.1.7. Modelo robusto con condiciones iniciales

Ahora consideremos la configuración de la red en un instante t, esto es,

$$\left(\mathcal{N}_t, E_t^1, E_t^2, ..., E_t^K\right)$$

En este caso, lo que intentaremos será seleccionar la configuración que minimice el impacto de las conexiones y desconexiones de nodos que se producirán en el intervalo entre t y $t + \Delta$. Es decir, las asignaciones deben ser realizadas de modo tal que en el instante $t + \Delta$ el PSQA promedio de la red sea máximo. Evidentemente, para conseguir esto deberíamos conocer el futuro (en este caso, qué entradas y salidas de nodos se producirán durante el próximo intervalo de tiempo), lo cual es imposible. Sin embargo, el estado de la red en el próximo instante de tiempo puede ser estimado si disponemos de ciertas probabilidades[5].

Probabilidad de que un nodo siga en la red

Para cada $v \in \mathcal{N}_t$, sea p_v la probabilidad de que el nodo v siga conectado en el instante $t + \Delta$, es decir, que pertenezca a $\mathcal{N}_{t+\Delta}$. Definimos además las siguientes variables aleatorias para cada $v \in \mathcal{N}_t$:

$$z_v = \begin{cases} 1 & \text{si el nodo } v \text{ sigue conectado a la red en el instante } t + \Delta \\ 0 & \text{en caso contrario} \end{cases}$$

Estado de la red en cada instante

Dados $i, j \in \mathcal{N}_t$ y $k \in \{1, 2, ..., K\}$, el estado de la red en el instante t está dado por:

$$x_{i,j}^k = \begin{cases} 1 & \text{si } i \text{ transmite } \sigma_k \text{ a } j \text{ en } t \\ 0 & \text{en caso contrario} \end{cases} \tag{3.10}$$

$$y_{i,j}^k = \begin{cases} 1 & \text{si } i \text{ precede a } j \text{ para } \sigma_k \text{ en } t \\ 0 & \text{en caso contrario} \end{cases} \tag{3.11}$$

Por otro lado, para el instante $t + \Delta$ definimos:

$$\tilde{x}_{i,j}^k = \begin{cases} 1 & \text{si } i \text{ transmite } \sigma_k \text{ a } j \text{ en } t + \Delta \\ 0 & \text{en caso contrario} \end{cases} \tag{3.12}$$

$$\tilde{y}_{i,j}^k = \begin{cases} 1 & \text{si } i \text{ precede a } j \text{ para } \sigma_k \text{ en } t + \Delta \\ 0 & \text{en caso contrario} \end{cases} \tag{3.13}$$

Calidad percibida global esperada

Como mencionamos anteriormente, el objetivo de este modelo es maximizar la calidad percibida global en el instante $t+\Delta$, con lo cual se minimiza el impacto que puedan tener las desconexiones de nodos ocurridas entre t y $t + \Delta$. Por lo tanto, la calidad percibida esperada de un nodo $v \in \mathcal{N}_t$ en el instante $t + \Delta$ está dada por:

$$\mathsf{PSQA}_{expected}(v) = f\left(\tilde{y}_{s,v}^1, \tilde{y}_{s,v}^2, ..., \tilde{y}_{s,v}^K\right)$$

[5]Estas probabilidades pueden ser obtenidas a partir de datos estadísticos reales.

Luego, la calidad percibida global esperada corresponde al promedio de las calidades percibidas tomado sobre los nodos que aún siguen en la red en el instante $t+\Delta$. Esto se calcula en realidad como un valor esperado, debido a las variables aleatorias z_v:

$$\overline{\mathsf{PSQA}}_{expected} = E\left(\frac{\sum_{v\in\mathcal{N}_t}\mathsf{PSQA}_{expected}(v)}{\sum_{v\in\mathcal{N}_t} z_v}\right)$$

Nótese que los nuevos nodos que se conectaron entre t y $t+\Delta$ no son tenidos en cuenta para el cálculo de la calidad percibida global esperada. Esto es correcto, ya que dichos nodos aún no han sido asignados y por lo tanto no tiene sentido considerarlos para la minimización del impacto causado por las desconexiones de nodos en ese intervalo.

Restricciones del modelo robusto

Ahora estamos en condiciones de enumerar las restricciones de este modelo. Se describen solamente aquellas que difieren de las del modelo estático.

$$s* = \arg\max_{s\in\mathcal{S}}(\overline{\mathsf{PSQA}}_{expected}(s)) \tag{3.14}$$

Esto significa que la solución factible óptima $s*$ es la que nos brinda una calidad percibida global esperada máxima.

$$y^k_{i,j} + y^k_{j,i} \leq 1 \qquad \forall i,j \in \mathcal{N}_t, \quad k = 1,2,...,K \tag{3.15}$$

$$y^k_{i,j} = x^k_{i,j} + \sum_{v\in\mathcal{N}_t} y^k_{i,v}x^k_{v,j} \qquad \forall i \in \mathcal{N}_t, \quad \forall j \in \mathcal{N}_t - \{s\}, \quad k = 1,2,...,K \tag{3.16}$$

$$\sum_{v\in\mathcal{N}_t} x^k_{v,i} \leq 1 \qquad \forall i \in \mathcal{N}_t, \quad k = 1,2,...,K \tag{3.17}$$

$$\sum_{k=1}^{K}\left(bw_k \sum_{v\in\mathcal{N}_t} x^k_{i,v}\right) \leq BW^{out}_i \qquad \forall i \in \mathcal{N}_t, \quad k = 1,2,...,K \tag{3.18}$$

$$x^k_{i,j}, y^k_{i,j} \in \{0,1\} \qquad \forall i,j \in \mathcal{N}_t, \quad k = 1,2,...,K \tag{3.19}$$

$$x^k_{i,j} = 1 \qquad \forall i,j \in \mathcal{N}_t, \quad k = 1,2,...,K \mid x^k_{i,j} \in E^k_t \tag{3.20}$$

$$\tilde{x}^k_{i,j} = z_i z_j x^k_{i,j} \qquad \forall i,j \in \mathcal{N}_t, \quad k = 1,2,...,K \tag{3.21}$$

La conexión entre dos nodos se mantiene si ninguno de los dos se desconecta entre t y $t+\Delta$.

$$\tilde{y}^k_{i,j} = \tilde{x}^k_{i,j} + \sum_{v\in\mathcal{N}_t} z_i\tilde{y}^k_{i,v}\tilde{x}^k_{v,j} \qquad \forall i \in \mathcal{N}_t, \quad \forall j \in \mathcal{N}_t - \{s\}, \quad k = 1,2,...,K \tag{3.22}$$

Un camino entre dos nodos se mantiene si ningún nodo que es parte del camino se desconecta entre t y $t + \Delta$.

$$\tilde{y}_{i,j}^{k} \leq y_{i,j}^{k} \qquad \forall i, j \in \mathcal{N}_t, \quad k = 1, 2, ..., K \tag{3.23}$$

No pueden aparecer nuevos caminos entre dos nodos durante el intervalo de tiempo considerado.

$$z_v \backsim Bern\,(p_v) \qquad \forall v \in \mathcal{N}_t \tag{3.24}$$

Las variables aleatorias z_v tienen distribución Bernoulli con parámetro p_v para cada $v \in \mathcal{N}_t$.

3.2. Comentarios acerca del modelo

3.2.1. Solvers

De acuerdo a todo lo presentado en la sección anterior, debería estar claro que determinar las asignaciones que es necesario efectuar para que el PSQA esperado de la red sea máximo no es ni más ni menos que un problema de *optimización combinatoria* sobre el modelo matemático descrito. En un primer momento, tras haber definido el modelo, se consideró la posibilidad de modelar la red por medio de alguna herramienta como AMPL[6] para después aplicar un *solver* con el cual resolver el problema de optimización mencionado. Sin embargo, tras analizar la complejidad del modelo y la capacidad de herramientas como AMPL para resolver problemas de este tipo, fue evidente el hecho de que no se podría modelar esta red de ese modo, por lo que habría que buscar otras formas de resolver el problema de optimización.

3.2.2. Heurísticas

Ante un problema de optimización de esta complejidad, lo más natural fue pensar en heurísticas para resolverlo. En el siguiente capítulo se describen los algoritmos implementados para manejar la dinámica de los nodos intentando maximizar la calidad global esperada. Además de los algoritmos que se implementaron, otra posibilidad que se consideró en un principio fue la de usar Algoritmos Genéticos, pero tuvo que ser descartada ya que se hacía extremadamente difícil modelar el problema con esta técnica, y era evidente que el modelo matemático de la red se adaptaba más a heurísticas como las que se implementaron finalmente.

[6] http://www.ampl.com

Capítulo 4

Simulador

Habiendo definido el modelo teórico para una red de distribución de video en vivo basada en P2P, el siguiente paso es definir los algoritmos que se usarán para reconfigurar las asignaciones dentro de la red luego de un intervalo de tiempo, en el cual se pueden haber producido conexiones y desconexiones de nodos. El objetivo es que esta reconfiguración de la red se haga de tal modo que se maximice la calidad global percibida en la red, y que se cumplan las restricciones planteadas en el Capítulo 3. Para poder probar el funcionamiento de los algoritmos que se implementen, se desarrolló una aplicación que simula el comportamiento de una red P2P como la que se define en el Capítulo 3, y en la cual se pueden ejecutar los algoritmos de asignación que se construyan. En este capítulo se describen las características del simulador implementado, principalmente su arquitectura y modo de funcionamiento. Luego, se presentan los algoritmos de asignación que fueron desarrollados y probados con el simulador, junto con la base teórica correspondiente.

Este simulador es la base del servidor de control utilizado en la red *Gol!P2P* [1].

4.1. Motor del simulador

En esta sección entramos en detalles acerca de cómo fue construido el simulador, primero describiendo a grandes rasgos su funcionamiento, y luego su arquitectura general, así como las principales estructuras de datos utilizadas, para finalmente dar unos detalles acerca del ambiente, lenguaje y tecnologías utilizadas para el desarrollo.

4.1.1. Introducción

El simulador supone que el funcionamiento de la red será tal como se describe en la Figura 4.1: se maneja una red basada en P2P en la que participan clientes que desean recibir un stream de video en vivo; este stream se distribuye utilizando la técnica de streaming multifuente, dividiendo el stream original en varios substreams y transmitiéndolos desde un nodo fuente (que se asume es el que dispone el contenido), y los clientes que reciben los substreams pueden actuar a su vez como servidores y transmitirle los substreams que estén recibiendo a otros clientes, siempre y cuando dispongan del ancho de banda suficiente para

hacerlo. Tal como vimos anteriormente, luego de cada cierto intervalo se analizan las conexiones y desconexiones de nodos producidas en la red, y se ejecuta un algoritmo de asignación para restablecer las asignaciones de modo de maximizar la calidad global percibida en la red. Cada una de estas ejecuciones después de un cierto intervalo de tiempo se denomina *iteración*. El simulador toma entonces como entrada una descripción de los nodos que participan en la red y la evolución de éstos (conexiones, desconexiones) a través de las diferentes iteraciones; se mantiene una estructura que representa a la red en un momento dado, y tras cada iteración, esta estructura es actualizada según las conexiones y desconexiones producidas en esa iteración, para luego ejecutar el algoritmo de reconfiguración que reasigna las conexiones en la red. Inmediatamente después se lleva a cabo la siguiente iteración, y así sucesivamente hasta completar todas las iteraciones especificadas en la entrada.

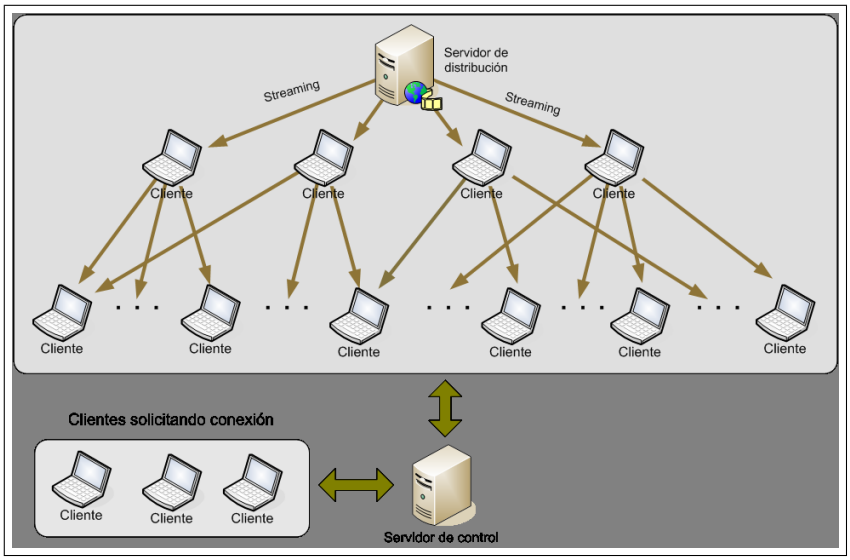


Figura 4.1: La red P2P de video en vivo considerada

4.1.2. Arquitectura del simulador

En la Figura 4.2 se puede apreciar una descripción del funcionamiento del simulador: éste toma como entrada un archivo en el cual se especifican tanto las propiedades estáticas de la red (tales como cantidad de substreams y bitrate de cada uno de ellos, ancho de banda de subida disponible en cada nodo, etc.) así como las dinámicas (se listan las iteraciones que serán ejecutadas en la simulación, indicando para cada una de ellas qué nodos nuevos ingresan a la red y cuales se desconectan). Con los datos provistos por este archivo el motor puede entonces simular el comportamiento de la red a través del tiempo, reflejando las conexiones y desconexiones de nodos en la estructura de datos interna que representa a la red, y aplicando un algoritmo de asignación[1] que reconfigure las conexiones en la red tras cada iteración con el objetivo de maximizar la calidad

[1]Tal como se mencionó anteriormente, es posible implementar nuevos algoritmos con facilidad ya que son independientes del motor de simulación.

global percibida. Como resultado de la simulación se obtiene una serie de datos estadísticos por cada iteración, tales como el valor del PSQA antes y después de cada reconfiguración de la red o tiempo de ejecución del algoritmo. Una

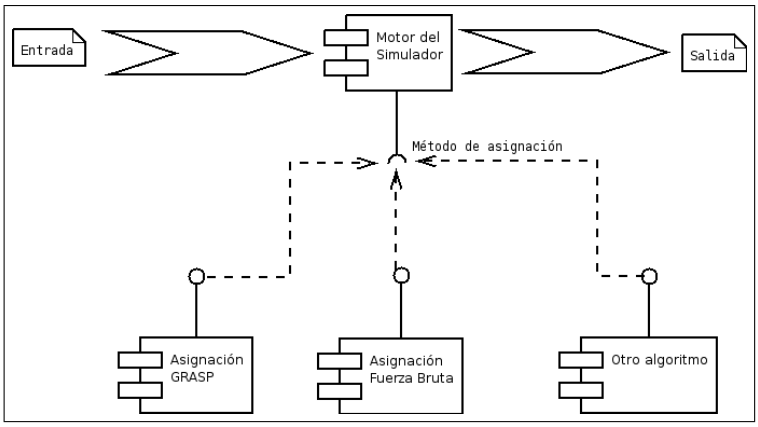


Figura 4.2: Funcionamiento del simulador

visión más clara de los componentes que integran el simulador se presenta en la Figura 4.3, y a continuación se detalla el rol de cada uno de estos componentes en el funcionamiento de la simulación:

Motor: el motor es el núcleo del simulador, este componente es el encargado de llevar a cabo el ciclo de iteraciones especificado en el archivo de entrada, actualizando la estructura de la red al final de cada iteración, según las conexiones y/o desconexiones de nodos que se produzcan y la reconfiguración de asignaciones según los resultados obtenidos por el algoritmo de asignación.

I/O: todas las tareas de acceso a archivos se delegan al componente de I/O (*input/output*, entrada/salida). En este módulo se encapsula el acceso a los datos para realizar la carga de las estructuras en memoria de forma transparente. Para realizar el acceso a archivos, se utilizó la biblioteca *libConfuse*[2], ya que provee funcionalidades de parsing de archivos de configuración fáciles de utilizar y que consumen pocos recursos.

Grafo: la red se representa en el simulador por medio de un grafo dirigido, en donde cada arista corresponde a la transmisión de un cierto substream de un nodo a otro. Junto con el grafo, en este componente se definen además una serie de funciones primitivas para acceder a la estructura, como por ejemplo asignar una conexión entre dos nodos, o conectar/desconectar un nodo en la red. En la sección siguiente se proporcionan más detalles acerca de esta estructura que representa a la red.

Algoritmo de asignación: el algoritmo de asignación sería en realidad un *metacomponente*, ya que no se trata de un elemento fijo en el simulador, puesto que el algoritmo a utilizar es configurable, análogamente al patrón conocido como *Estrategia* en la Programación Orientada a Objetos. El

[2]http://www.nongnu.org/confuse/

algoritmo implementado será ejecutado tras cada iteración y su objetivo es reconfigurar las conexiones de la red para maximizar la calidad global percibida.

PSQA: junto con el modelo teórico se introdujo el concepto de PSQA para un nodo, y vimos que se trata de una función que depende de qué substreams están siendo recibidos en un momento dado por el nodo. En este componente se encapsula el cálculo del PSQA a partir de los substreams recibidos, de tal manera que cualquier modificación futura en el cálculo del PSQA no impacte en el resto de la aplicación.

Sorteos: puesto que según el modelo el objetivo es estimar la calidad global percibida *esperada*, se hace necesario utilizar datos estadísticos para intentar predecir el estado futuro de la red. Por eso es que el simulador debe, en ciertos momentos, realizar sorteos utilizando probabilidades para tratar de "adivinar" qué nodos se desconectarán de la red en la próxima iteración. La responsabilidad de este módulo es realizar los sorteos para predecir posibles estados futuros de la red.

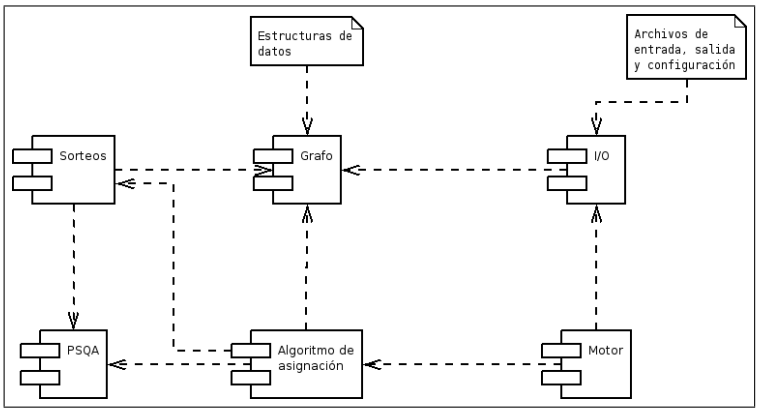


Figura 4.3: Componentes del simulador

4.1.3. Estructuras de datos

Ahora veremos con mayor detalle la representación de la red en el simulador. Para ello se definió una estructura de datos similar a un grafo dirigido, que consiste en un array de los nodos que componen la red, siendo cada nodo una estructura compuesta por los siguientes elementos:

Identificador: a cada nodo se le asigna un identificador numérico único en la red para poder distinguirlo de los demás.

Ancho de banda disponible: en este campo se almacena el ancho de banda de subida que el nodo tiene disponible en un momento dado.

Ancho de banda total: aquí se almacena el ancho de banda de subida *total* para el nodo en cuestión.

Probabilidad de seguir conectado: este valor corresponde a la probabilidad de que el nodo siga conectado en la próxima iteración.

Hijos: para cada substream, se almacena una lista de los nodos a los cuales este nodo les transmite dicho substream.

Padres: este campo es un array en el cual se indica de quién se está recibiendo cada substream (en caso de que se esté recibiendo).

Substreams recibidos: en este campo se almacena una lista que indica para cada substream si existe un camino desde el nodo fuente.

4.1.4. Detalles de implementación

El simulador fue implementado en lenguaje C sobre una plataforma Ubuntu Linux 6, utilizando el GNU C Compiler (`gcc`). La elección del lenguaje se hizo específicamente por la complejidad de los algoritmos implementados, y la necesidad de que estos tengan el menor tiempo de ejecución que sea posible. Según se nos informó acerca de experiencias anteriores en problemas similares, el uso de lenguajes de un poco más alto nivel (como C++ por ejemplo) ya provocaba aumentos inaceptables en los tiempos de ejecución, por lo cual C fue en todo momento la opción más aceptable en este contexto.

4.2. Greedy Randomized Adaptative Search Procedure (GRASP)

4.2.1. Introducción teórica

GRASP o procedimiento de búsqueda ávido, aleatorio y adaptativo es una muy conocida metaheurística, la cual fue aplicada con éxito para resolver diferentes problemas de optimización combinatoria.

Antes de presentar una descripción de las principales ideas de GRASP, formulamos un problema genérico de optimización combinatoria basado en la descripción introducida en [26]. Consideremos:

i) $N = \{n_1, \ldots, n_m\}$ es el conjunto básico finito que contiene los elementos potenciales que podrán formar parte de una solución factible.

ii) F denota el conjunto de soluciones factibles que satisfacen: $F \subseteq 2^N$.

iii) $f : 2^N \to \mathbb{R}$ es la función objetivo. Sin perder generalidad, asumiremos la versión de minimización, ej. el objetivo es encontrar una solución óptima global $S^* \in F$ tal que $f(S^*) \leq f(S)$, $\forall S \in F$.

Estos puntos serán determinados, cuando se especifique el problema de optimización a ser estudiado. Por ejemplo, en el caso del problema de la Cobertura Mínima de Vértices (*Minimum Vertex Covering*):

- $N = \{v_1, \ldots, v_n\}$ es el conjunto de nodos a ser considerados,
- E es el conjunto de aristas que conectan los nodos de N,

- F está compuesto por todos los subconjuntos de N tal que si $S \in F$ cualquier arista de E tiene al menos un punto destino en S,
- $f(S)$ es el número de nodos pertenecientes a S.

GRASP es un proceso multi-arranque o iterativo, en el cual cada iteración consiste en dos fases: construcción y búsqueda local. La fase de construcción genera una solución factible, luego su vecindad (que deberá ser definida cuando se adapte el método a un problema específico) es explorada durante la fase de búsqueda local en busca de una mejora. La mejor solución de todas las iteraciones GRASP es devuelta como resultado.

```
Procedure GRASP(ListSize,MaxIter,Seed);

1  Read_Input_Instance();
2  for k = 1 to MaxIter do
3          InitialSolution ← Construct_Greedy_Randomized_Solution(ListSize, Seed);
4          LocalSearchSolution ← Local_Search(InitialSolution);
5          if cost(LocalSearchSolution) < cost(BestSolutionFound) then
6                  Update_Solution(BestSolutionFound, LocalSearchSolution);
7          end_if;
8  end_for;
9  return BestSolutionFound;
```

Figura 4.4: Pseudocódigo GRASP.

A continuación se describirá una implementación genérica de GRASP donde su pseudocódigo puede ser visto en la Figura 4.4. La heurística GRASP tiene tres parámetros principales: el número de iteraciones $MaxIter$, el tamaño de la lista de candidatos $ListSize$, y un tercer parámetro implícito, la semilla inicial $Seed$ para el generador de números pseudoaleatorios. El primer parámetro corresponde al número de iteraciones en el ciclo externo del algoritmo. El segundo parámetro a groso modo es una medida de cuantas alternativas serán tomadas en cuenta en cada paso de la construcción.

En algunas versiones GRASP el tamaño de la lista de candidatos restringidos es reprocesado dinámicamente (ej. el valor de $ListSize$ no es estacionario), siendo usado en este caso un parámetro de umbral denotado por α.

Como se puede apreciar en el pseudocódigo de la Figura 4.4, las iteraciones GRASP se desarrollan en las líneas 2-8. Cada iteración GRASP consiste de la fase de construcción (línea 3), de la fase de búsqueda local (línea 4) y, si es necesario, de la actualización de la solución (líneas 5-7).

Fase de Construcción

En la fase de construcción, una solución factible es construida. La Figura 4.5 muestra un pseudocódigo genérico para la fase de construcción. La solución es usualmente representada como un grupo de elementos (dependiendo de cada problema específico); la fase de construcción empieza con un grupo vacío e iterativamente agrega un elemento hasta que el grupo corresponda a una solución factible. En cada paso de la fase de construcción, una lista restringida de candidatos (denotada por RCL) es determinada por el ordenamiento de todos los elementos que aún no fueron seleccionados con respecto al procedimiento ávido que mide el beneficio de incluirlos en la solución parcial. En general, este procedimiento evalúa el incremento en la función de costo f(.) cuando se incorpora un

nuevo elemento en la solución en construcción. Específicamente, al aplicar esta función, nosotros construimos el RCL conteniendo esos elementos cuya incorporación a la solución parcial actual induce los incrementos de costos mas pequeños (este es el componente ávido de GRASP). El próximo elemento a ser incluido en la solución parcial es escogido aleatoriamente (uniformemente o en una forma influenciada) del RCL (este es el componente probabilístico del GRASP). En esta forma, el GRASP permite obtener diferentes soluciones en cada iteración GRASP. Cuando el elemento escogido es agregado a la solución parcial, los beneficios asociados con cada elemento no agregado todavía a la solución parcial son actualizados de modo que reflejen el cambio inducido por la inserción de un nuevo elemento. Debido a esto la heurística reprocesa el RCL y reevalúa los costos incrementales (este es el componente adaptativo del GRASP). Una vez que la fase de construcción es terminada, la solución construida es devuelta.

```
Procedure Construct_Greedy_Randomized_Solution(ListSize,Seed);

1  Solution ← ∅;
2  Incremental costs evaluation for the candidate elements;
3  while not_feasible(Solution) do;
4          RCL ← the restricted candidate list;
5          s ← select randomly an element from the RCL;
6          Solution ← Solution ∪ {s};
7          Incremental costs reevaluation;
8  end_while;
9  return Solution;
```

Figura 4.5: Pseudocódigo de la Fase de Construcción.

La solución generada por la fase de construcción no está garantizada para ser localmente óptima con respecto a simples definiciones de la vecindad. Por esta razón, puede ser beneficioso aplicar una búsqueda local para tratar de mejorar cada solución construida. Un algoritmo de búsqueda local trabaja en forma iterativa al reemplazar sucesivamente la solución actual por una mejor tomada de su vecindad. Este finaliza una vez que no hay mejor solución encontrada en la vecindad.

Fase de búsqueda local

La Figura 4.6 muestra un pseudocódigo genérico para la fase de búsqueda local. Tiene como entrada una solución factible *Solution* y luego se busca una mejor solución dentro de la vecindad $N(Solution)$ previamente definida. En la mayoría de los casos, la fase de búsqueda local toma como entrada la solución factible *Solution* entregada por la fase de construcción, pero para ciertas aplicaciones se podrían tener varias fases de búsqueda local trabajando en forma combinada para explorar diferentes vecindades, implicando esto que sus entradas no son necesariamente las soluciones dadas por la fase de construcción.

El éxito al aplicar la fase de búsqueda local está fuertemente relacionado con los siguientes puntos:

- Una elección conveniente de la estructura de la vecindad
- Técnicas eficientes de búsqueda en la vecindad
- Una fácil evaluación de la función de costo cuando se explora la vecindad

```
Procedure Local_Search(Solution);

1  while not_locally_optimal(Solution) do;
2     Find Neighbor_Solution ∈ N(Solution) satisfying f(Neighbor_Solution) < f(Solution);
3     Solution ← Neighbor_Solution;
4  end_while;
5  return Solution;
```

Figura 4.6: Pseudocódigo de la Búsqueda Local.

- La calidad de la solución inicial

La fase de construcción juega un rol muy importante respecto a este último punto, ya que debe de producir buenas soluciones iniciales para este subprocedimiento de búsqueda local. Dependiendo del problema las vecindades usadas no son complejas generalmente.

Existen 2 estrategias básicas para explorar una vecindad, las cuales son:

mejor-mejora: toda la vecindad es explorada y la solución actual es (posiblemente) reemplazada por la mejor solución de la vecindad.

primer-mejora: cuando se encuentra la primer mejor solución mientras se explora la vecindad (ej. cuando su valor de costo es mas pequeño que aquel de la solución actual), la solución actual es reemplazada por esta.

En [26], los autores mencionan que empíricamente (cuando se aplican ambas estrategias en muchas aplicaciones), en la mayoría de los casos, ambas estrategias alcanzan la misma solución final, pero en general la estrategia de primera-mejora toma menor tiempo de procesamiento. Además se ha observado que es más frecuente la convergencia prematura a una solución no-global óptima local usando *mejor-mejora* que *primer-mejora*.

Una característica importante del GRASP es su baja parametrización; se necesitan pocos parámetros para ser ajustada. Esto implica que el esfuerzo principal puede ser enfocado en implementar las estructuras eficientes de datos para obtener iteraciones más rápidas.

A continuación se analizará la influencia de los parámetros del GRASP y la construcción RCL.

Un algoritmo GRASP finaliza una vez realizadas las iteraciones $MaxIter$. Claramente, la probabilidad de encontrar una nueva solución mejorando la actualmente primera mejor solución decrece con el número de iteraciones ya procesadas, la calidad de la mejor solución encontrada, podría sólo mejorar con la última. En general, los tiempos de procesamiento de iteración a iteración son relativamente similares, por lo tanto el tiempo total de computación depende linealmente de $MaxIter$. Debido a esto, cuando se incrementa MaxIter , el tiempo de procesamiento global será incrementado así como la probabilidad de encontrar mejores soluciones.

En cualquier iteración GRASP, denotando a $c(e)$ como el costo incremental asociado con la inserción del elemento $e \in E$ en la solución en construcción y por c_{min} y c_{max} como los costos incrementales más pequeño y más grande respectivamente. Hay dos variantes principales en el proceso del RCL usado en la fase de construcción. A continuación, se describen ambas aproximaciones.

i) Dado un entero positivo $ListSize$, el RCL es compuesto por $ListSize$ elementos de E con los mejores costos incrementales (ej. los mínimos). En este caso, se dice que el RCL es basado en cardinalidad. El tamaño del RCL puede ser más pequeño que $ListSize$ entonces, dependiendo del caso, no se podrían obtener para procesar los mejores $ListSize$ elementos con exactitud.

ii) La segunda variante usa un parámetro de umbral denotado por $\alpha \in [0, 1]$. En este caso el tamaño de RCL es dinámicamente adaptado de acuerdo a la calidad de los elementos a ser agregados (entonces el RCL es basado en valor). Fijo α, el RCL es formado por todos los elementos "factibles" $e \in E$ los cuales pueden ser insertados en la solución parcial en construcción sin perder factibilidad y cuya calidad es superior al valor de umbral; es decir:

$$e \in \text{RCL} \Leftrightarrow c(e) \in [c_{min}, c_{min} + \alpha(c_{max} - c_{min})].$$

Si fijamos $\alpha = 0$ el algoritmo resultante es puramente ávido, y con $\alpha = 1$ se obtiene una construcción aleatoria. Por esta razón, se infiere que α regula el grado de aleatoriedad en la fase de construcción.

Para mayores detalles del GRASP puede consultar las referencias [26, 27, 28, 29, 30, 31], las cuales proveen de un extenso análisis de la metaheurística GRASP basada en muchas aplicaciones. Los temas discutidos incluyen: implementación de técnicas exitosas, estrategias de ajuste (*tuning*) de parámetros, mecanismos alternativos de construcción de soluciones, técnicas para acelerar la búsqueda local, GRASP reactivo, perturbaciones de costo, funciones de sesgo (*bias*), memoria y aprendizaje, búsqueda local en soluciones parcialmente construidas, filtrado, dispersión (*hashing*), implementación de estrategias de intensificación basadas en memoria y técnicas post-optimización usando reencadenamiento de trayectorias (*path-relinking*), hibridación con otras metaheurísticas y estrategias de paralelización.

4.2.2. Aplicación de GRASP al simulador

Como ya vimos en la descripción del modelo, el tiempo es considerado como puntos discretos de la forma $t_n = n\delta$ y se define $I_n = (t_{n-1}, t_n)$. En cada instante t_n, el algoritmo será ejecutado, armando un nuevo sistema a partir del estado t_n^- que es el resultado de la evolución del sistema en el intervalo I_n, es decir, algunos nodos pueden haber dejado el sistema y pueden aparecer nuevos nodos requiriendo conexión. A continuación se presenta el algoritmo basado en GRASP que resuelve nuestro problema de optimización y será ejecutado en cada instante t_n.

La idea principal del algoritmo es iterar varias veces, construyendo una solución factible en cada iteración, comparar estas soluciones obtenidas aplicando la función objetivo y finalmente retornar aquella solución cuya evaluación sea la mejor. La customización de GRASP de nuestro problema es presentado en el pseudocódigo de la Figura 4.7. Como puede apreciarse, hemos optado por no utilizar una fase de búsqueda local que se explicará más adelante en la sección 4.2.2.

El algoritmo tiene dos entradas, g_0, que es el grafo que representa la red al final de un determinado intervalo, y cfg, que representa la estructura con los

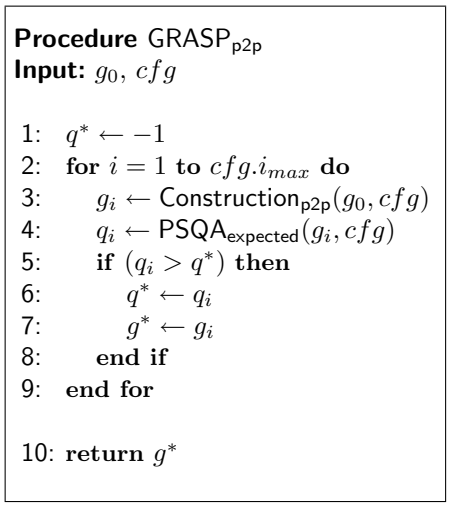


Figura 4.7: Algoritmo GRASP

parámetros de configuración del GRASP, como el $cfg.i_{max}$ (línea 2) que indica la cantidad máxima de iteraciones del algoritmo.

El procedimiento comienza inicializando la variable q^* (línea 1), donde se irá almacenando la mejor calidad de las soluciones construidas. A continuación, se realiza el proceso iterativo de construcción de soluciones (líneas 2-9). En cada iteración, se construye una solución factible (línea 3), se evalúa la calidad esperada futura de la red (nuestra función objetivo) de dicha solución (línea 4) y luego en las líneas 5-8 se chequea si es mejor cualitativamente la solución construida respecto a las anteriores y de ser así, ésta es almacenada para comparación. Finalmente en la línea 10 se retorna la mejor solución encontrada. Dicho de otra forma, si el simulador se configura con m iteraciones, se construyen m soluciones $S_1, S_2, \ldots, S_m$; la solución que nos interesa es la que se adapte mejor a la dinámica de la red permitiendo obtener una mejor calidad global futura (función $\text{PSQA}_{expected}()$ que se llama en el paso 4 de la Figura 4.7). Esto se realiza prediciendo para cada una de las m soluciones, un estado futuro de la red en el próximo instante de tiempo, es decir, sorteando la próxima salida de nodos en base a sus probabilidades de permanecer conectados en el próximo intervalo de tiempo I_{n+1}. Esta predicción o sorteo lo realizamos p veces para cada solución y para cada uno de los estados futuros obtenidos, calculamos el PSQA de la red o grafo resultante y luego tomamos el promedio de estos resultados, como se muestra a continuación:

$$\text{PSQA}_{expected}(S_i) = \frac{\sum_{j=1}^{p} \text{PSQA}(Sorteo_j(S_i))}{p}$$

Finalmente, la solución elegida S^* será la que tenga mayor PSQA esperado (promedio de los PSQA futuros sorteados), a saber:

$$S^* = \text{argmax}\{\text{PSQA}_{expected}(S_1), \text{PSQA}_{expected}(S_2), \ldots, \text{PSQA}_{expected}(S_m)\}$$

Procedure Construction$_{\text{p2p}}$
Input: g, cfg

1: $C \leftarrow$ FindAllAssignmentsCandidates(g)
2: $A \leftarrow \varnothing$
3: **while** $C \neq \varnothing$ **do**
4: $\quad C_{RCL} \leftarrow$ GenerateRCL$_{\text{p2p}}(g, cfg, C)$
5: $\quad a \leftarrow$ SelectRandom(C_{RCL})
6: $\quad A \leftarrow A \cup \{a\}$
7: $\quad C \leftarrow$ UpdateCandidates(g, a, C)
8: **end while**
9: $g_c \leftarrow$ ApplyAssignments2Graph(g, A)

10: **return** g_c

Figura 4.8: Construcción de GRASP

Un punto clave en el diseño del algoritmo es la fase de construcción de una solución factible, que se presenta en la Figura 4.8. En la línea 1 del algoritmo se obtienen todos los candidatos de posibles asignaciones. Esto se realiza analizando el grafo de entrada y obteniendo para cada substream $s = 1, \ldots, k$, todos los nodos raíces de los subárboles desconectados (aquellos que no están recibiendo el substream), que denotaremos como conjunto H^s y los nodos del árbol principal (aquel cuya raíz es el nodo fuente) que aún tienen ancho de banda disponible para dicho substream, que denotaremos P^s.
Con estos datos, se determina el conjunto C de todas las diferentes asignaciones que es posible realizar en el grafo, siendo una asignación cualquier relación de un nodo "desconectado" o "hijo" con un nodo "padre", mientras estos cumplan las restricciones recién mencionadas dentro del mismo substream. Formalmente este conjunto se puede definir como:

$$C = \{(p, h)^s \mid p \in P^s \ , \ h \in H^s \ ; \ s = 1, \ldots, k\}$$

Luego de determinar todos los posibles candidatos de asignación, ya se puede comenzar con el proceso constructivo indicado por las líneas 3-8. En la línea 4 se construye una lista restringida de candidatos, que son los mejores según el criterio que se haya especificado en la configuración (cfg). Estos criterios se describen en la sección 4.2.2. Luego en la línea 5 se selecciona uno de ellos al azar (componente aleatorio del GRASP). En la línea 6 se agrega la asignación elegida al conjunto de asignaciones que se van seleccionando en cada iteración y finalmente en el paso 7 se debe actualizar el conjunto C de los posibles candidatos, eliminando el candidato que se acaba de seleccionar y agregando nuevos si es que la asignación realizada así lo permite. Esto último puede suceder por ejemplo si asignamos un subárbol desconectado que tiene muchos nodos con ancho de banda disponible que podrán ser también padres para futuras asignaciones. Finalmente cuando ya no haya candidatos para posibles asignaciones ($C = \varnothing$, "condición de parada"de nuestro algoritmo), lo que significa que ya no existen posibles padres donde colgar o ya fueron colgados todos los nodos

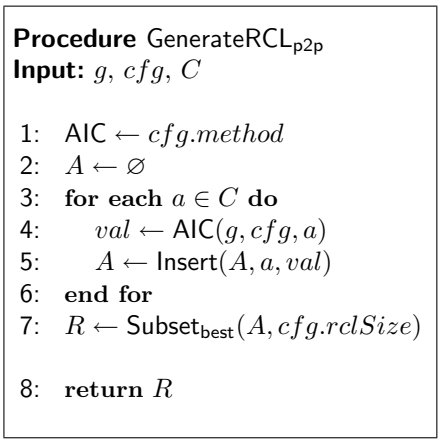


Figura 4.9: Generación de la lista restringida de candidatos

desconectados, se procede a aplicar las asignaciones seleccionadas A al grafo de entrada, y el resultado será devuelto como una solución factible.

Así como el procedimiento de construcción es clave en nuestro algoritmo GRASP de la Figura 4.7, el procedimiento de generación de la lista restringida de candidatos lo es para la fase de construcción. En la Figura 4.9 se muestra el pseudocódigo de este procedimiento.

Criterios de mejora de una asignación

La Figura 4.9 muestra el pseudocódigo del procedimiento GenerateRCL$_{p2p}$, donde en la línea 1 se determina, según el archivo de configuración del simulador, la función que permitirá evaluar la mejora parcial que se obtiene al realizar una determinada asignación. Esta función encapsula el criterio de mejora de una asignación simple y representa el componente ávido del algoritmo GRASP. En este trabajo, se han desarrollado tres criterios:

- *Ancho de Banda*
- *PSQA Actual*
- *PSQA Futuro*

El criterio de *"Ancho de Banda"* prioriza el aumento de ancho de banda del árbol principal al realizar una asignación, el cual se calcula como la suma de los anchos de banda disponibles del subárbol que se conecta al realizar la asignación. Esto estaría permitiendo aumentar la cantidad de posibles nuevos padres para futuras asignaciones.

El criterio de *"PSQA Actual"* prioriza el aumento de calidad o PSQA del grafo resultante al realizar determinada asignación. Esto tipo de criterio nos estaría permitiendo ampliar la calidad instantánea de la red que estaría implicando en principio colgar primero los subárboles desconectados con mayor cantidad de nodos.

Por último el criterio de *"PSQA Futuro"* prioriza la mejora de calidad o PSQA en un posible instante futuro, algo similar al PSQA esperado que se evalúa luego de la fase de construcción en el algoritmo GRASP de la Figura 4.7. De esta forma, se aplica una asignación al grafo actual y al grafo resultante se le realizan varios sorteos (la cantidad está especificada en el archivo de configuración para el algoritmo GRASP) para finalmente promediar el PSQA de los posibles grafos futuros obtenidos de los sorteos. La mejor asignación será aquélla que permita el mayor valor promedio de los PSQA evaluados a futuro.

Continuando con el procedimiento de la selección de la lista restringida de candidatos, luego de determinar el criterio que se aplicará al algoritmo, se comienza con dicha evaluación para cada asignación a de la lista de candidatos C y se va guardando cada asignación con su respectivo valor retornado en el paso 4 en el conjunto A. Finalmente, una vez realizadas todas las evaluaciones de las asignaciones candidatas, se toma la lista (con el tamaño especificado en la configuración) de aquellas asignaciones que tengan mayor valor dentro del conjunto A, como se muestra en el paso 7 para finalmente retornar ese subconjunto de asignaciones.

Búsqueda Local

Nuestro algoritmo GRASP se diferencia respecto al procedimiento GRASP clásico en que no contiene una fase de búsqueda local donde se explora estructuras de vecindad. Como ya se ha presentado en la introducción teórica, en esta fase se intenta mejorar la solución obtenida en la fase de construcción, explorando un espacio "cercano" de posibles soluciones, aplicando pequeñas modificaciones a la solución recién construida con el fin de obtener una solución aún mejor.

En el análisis de como aplicar la búsqueda local al simulador, nos encontramos con la dificultad para definir una función de vecindad, la cual debería permitirnos realizar cambios en las asignaciones que fueron aplicadas en la fase de construcción. El problema que surge es que el cambio de alguna de estas asignaciones repercute fuertemente en la factibilidad de la solución. Para explicar este fenómeno, supongamos que en la fase de construcción de una solución se hayan aplicado las n asignaciones $(a_1, a_2, ..., a_k, ..., a_n)$. Ahora si intentáramos cambiar la asignación a_k, puede suceder que alguna o varias de las asignaciones posteriores $a_{k+1}, ..., a_n$ no sean posibles de realizar. Esto se ve claramente si por ejemplo la asignación a_{k+1} involucra a alguno de los posibles nodos "padres"del subárbol desconectado que fue colgado al realizar la asignación a_k y de esta manera la asignación a_{k+1} debería ser cambiada también para mantener la factibilidad, con el riesgo de que este fenómeno se replique de forma similar para las próximas asignaciones. Este es un problema muy complejo que no se intentará resolver en este trabajo.

Sin embargo intentaremos utilizar algún método embebido en GRASP que nos permita obtener mejores soluciones como forma de sustitución de la fase búsqueda local. Como ya se ha mencionado anteriormente el punto clave del GRASP es la fase de construcción, si no se construye una solución "buena", no hay mucho que pueda hacer la fase de búsqueda local. Por lo tanto, si pudiéramos alternar

el procedimiento para la selección de una asignación dentro de la fase de construcción entonces se podría explorar otras zonas del espacio de búsqueda que nunca serían visitadas por el algoritmo GRASP que acabamos de presentar.

En nuestro caso el método elegido para hacerlo es RNN (Random Neural Network) que se presentará en las secciones siguientes.

4.3. Random Neural Network

4.3.1. Introducción teórica

La Random Neural Network (RNN) es un modelo novedoso introducido por Gelenbe [32, 33, 34] que puede ser aplicado en optimización así como en configuraciones de aprendizaje. La RNN difiere sustancialmente de los modelos conexionistas ya existentes. La principal diferencia respecto a otros modelos existentes es que se puede resolver numéricamente (computando una ecuación de punto fijo) muy fácilmente y sin usar simulaciones Monte Carlo paso a paso. Específicamente en problemas de optimización, la RNN ha demostrado ser eficiente y computacionalmente muy rápida comparada con los modelos clásicos ya existentes [35, 36, 37, 38, 39]. En particular, el modelo RNN ya ha sido aplicado a los siguientes problemas de optimización combinatorios NP-Duros:

- *Problema del Vendedor Viajero* (TSP). En este caso, se ha mostrado en [39] que RNN dinámico obtiene soluciones óptimas o cercanamente óptimas en la mayoría de las instancias de referencia probadas.
- *Problema de Cobertura Minimal de Vértices* (MVCP). En [37], los autores comparan el rendimiento de RNN, el algoritmo convencional Ávido, la red Hopfield, y el Recorrido (Annealing) Simulado, aplicado al MVCP; los resultados revelan que los resultados obtenidos por la heurística RNN son significativamente superiores a los otros en términos de optimización en general.
- *Problema de Steiner en Grafos* (SPG). En [38], los autores usan RNN como un mejorador de las soluciones retornadas por las heurísticas clásicas SPG: *Heurística del problema del Árbol de Extensión Mínimo* (MSTH) y la *Heurística de la Distancia Promedio* (ADH) aplicadas a un grupo de prueba compuesto de grafos generados aleatoriamente. Se reporta que las soluciones obtenidas tienen una mejor calidad que la heurística inicial correspondiente, logrando una diferencia en costo de menos de uno o dos puntos de porcentaje del óptimo.

El modelo RNN funciona de la siguiente forma. Señales en la forma de impulsos de unidad de amplitud circulan entre un conjunto de neuronas. Estas pueden ser positivas o negativas que representan excitación e inhibición respectivamente. Cada neurona i tiene asociado un estado $k_i(t)$, que es su *potencial* en el tiempo t, representado por un entero no-negativo. El estado la red de n-neuronas en el tiempo t, representado por el vector de enteros no-negativos $k(t) = (k_1(t), \ldots, k_n(t))$. Los valores del vector de estado y el estado de la $i-th$ neurona es usualmente denotado por k y k_i respectivamente. Por definición, una neurona i está excitada sii su potencial es estrictamente positivo. Si una neurona

i está excitada puede transmitir impulsos a otras neuronas o hacia fuera de la red (decimos que la neurona i "dispara"). El potencial de la neurona se incrementa en 1 cuando una señal positiva llega o se reduce de igual forma si la señal es negativa. También se reduce en 1 cuando la neurona dispara. Si una neurona i emite un impulso, sea de excitación o de inhibición, éste perderá potencial de una unidad, pasando del estado cuyo valor es k_i al estado cuyo valor es $k_i - 1$.

Una neurona i "dispara"si se encuentra *excitada*, es decir, si su potencial es estrictamente positivo. Los impulsos serán enviados como un proceso de Poisson de tasa r_i con intervalos de distribución $i.i.d.$ exponencial. Los impulsos serán enviados a una neurona j con probabilidad p_{ij}^+ como señales excitativas, o con probabilidad p_{ij}^- como señales inhibitorias. Una neurona también puede enviar señales fuera de la red con probabilidad d_i, por lo tanto $d_i + \sum_{j=1}^{n}(p_{ij}^+ + p_{ij}^-) = 1$. Sean i y j dos neuronas, las tasas excitativa e inhibitoria de i a j están dadas por: $w_{ij}^+ = r_i p_{ij}^+$ y $w_{ij}^- = r_i p_{ij}^-$ respectivamente. Así la tasa de disparos de la neurona i es $r_i = \sum_{j=1}^{n}(w_{ij}^+ + w_{ij}^-)$. Las matrices $\mathcal{W}^+ = \{w_{ij}^+\}$ y $\mathcal{W}^- = \{w_{ij}^-\}$ pueden ser vistas análogamente a los pesos sinápticos en los modelos conexionistas, aunque representan tasas de emisión de impulsos excitativos e inhibitorios. Notemos que éstos son no-negativos ya que sus valores son producto de tasas y probabilidades.

Asimismo, señales exógenas excitativas e inhibitorias llegan a la neurona i de acuerdo a procesos de Poisson con tasas α_i y β_i respectivamente.

A continuación, se resume las dinámicas del modelo RNN considerando las posibles transiciones de estado. El estado de la neurona $k_i(t)$ puede ser modificado cuando ocurren ciertas transiciones, más precisamente:

i) El potencial $k_i(t)$ de una neurona puede decrementarse en uno al disparar tanto una señal de excitación como una de inhibición. También, cuando una señal exógena inhibitoria llega desde fuera de la red a la neurona i, su potencial disminuye a $k_i(t) - 1$. Por último, la neurona i disminuirá su potencial $k_i(t)$ en uno si recibe un impulso inhibitorio desde otra neurona j.

ii) Cuando llega una señal excitativa desde fuera de la red o desde otra neurona dentro de la red, ésto resultará en un incremento del potencial de la neurona en uno, obteniendo así $k_i(t) + 1$.

iii) El valor del estado de la $i - th$ neurona permanece sin cambios cuando ninguno de los eventos anteriores ocurre.

Además, existen dos condiciones de borde que previenen que ocurran algunas de las transiciones; éstas son:

- Una neurona puede disparar sólo si su potencial es positivo (como se mencionó anteriormente).

- Cuando la neurona tiene potencial cero, el arribo de nuevas señales inhibitorias no hacen que decrezca su valor.

El análisis del modelo RNN está focalizado en dos medidas: La probabilidad de distribución del estado de la red $p(k,t) = Pr[k(t) = k]$ y la probabilidad marginal que la neurona i sea excitada $q_i(t) = Pr[k_i(t) > 0]$. El comportamiento del modelo se describe por un sistema infinito de ecuaciones Chapman-Kolmogorov para sistemas Markovianos discretos de espacio de estados de tiempo continuo.

La información en el RNN está representado por la frecuencia a la cual las señales viajan. Las neuronas pueden ser consideradas como *moduladores y demoduladores de frecuencia*. Intuitivamente, cada neurona de este modelo es un modulador de frecuencia, ya que la neurona i envía señales excitativas e inhibitorias a tasas (o frecuencias) $q_i(t)r_ip_{ij}^+$, $q_i(t)r_ip_{ij}^-$ a cualquier neurona j. De esta forma, cuando una neurona i está excitada, ésta enviará señales a la neurona j a la frecuencia $w_{ij} = w_{ij}^+ + w_{ij}^-$. Estas señales serán emitidas en intervalos aleatorios distribuidos exponencialmente. A su turno, cada neurona se comporta como un demodulador de frecuencia no-lineal ya que transforma las tasas de llegada de señales excitativas e inhibitorias en una "amplitud", que es $q_i(t)$, la probabilidad de que la neurona i sea excitada en el tiempo t.

El análisis asintótico del modelo de la RNN consiste el procesar la distribución de probabilidad estacionaria, i.e. los limites:

$$p(k) = \lim_{t\to+\infty} p(k,t),$$
$$q_i = \lim_{t\to+\infty} q_i(t),\ \forall i \in (1,\ldots,n).$$

Gelenbe prueba el siguiente teorema e introduce ecuaciones las cuales proveen la forma para procesar iterativamente la distribución de probabilidad estacionaria para el modelo de la RNN.

Teorema 4.3.1.1 *El vector $q = (q_1,\ldots,q_n)$ satisface el sistema de ecuaciones no lineales:*

$$q_i = \frac{\lambda_i^+}{r_i + \lambda_i^-},\ \forall i \in 1,\ldots,n. \tag{4.1}$$

donde λ_i^+ y λ_i^- satisfacen las siguientes ecuaciones simultaneas:

$$\lambda_i^+ = \sum_j q_j w_{ji}^+ + \alpha_i, \tag{4.2}$$

$$\lambda_i^- = \sum_j q_j w_{ji}^- + \beta_i. \tag{4.3}$$

Además, otro teorema introducido por Gelenbe garantiza el procesamiento del vector q, ya que λ_i^+ y λ_i^- pueden ser procesados siempre. Al reemplazar λ_i^+ y λ_i^- en la ecuación 4.1, obtenemos una ecuación de punto fijo $F(q) = q$, cuya expresión explicita esta dada por:

$$q_i = \frac{\sum_{j,j\neq i}(q_j w_{ji}^+ + \alpha_i)}{r_i + \sum_{j,j\neq i}(q_j w_{ji}^- + \beta_i)},\ \forall i \in 1,\ldots,n. \tag{4.4}$$

Cuando iterativamente se resuelve la ecuación $F(q) = q$, si en cierta iteración se obtiene un $q_i > 1$, entonces forzamos a $q_i = 1$ hasta que la convergencia (decimos que la neurona i es saturada). Empíricamente, en la mayoría de los casos la convergencia es alcanzada en pocas iteraciones. Además, hay que notar que la ecuación de punto fijo $F(q) = q$ puede ser resuelta en paralelo, permitiendo así reducir significativamente el tiempo de procesamiento. Para mayores detalles de la RNN, puede consultar las siguientes referencias [40, 32, 33, 34].

4.3.2. Aplicación de GRASP + RNN al simulador

Como se ha mencionado anteriormente el modelo RNN reemplazaría al método de selección de una asignación en la fase de construcción del GRASP. La idea es intercalar el método de selección basado en el modelo RNN con el método de selección del GRASP mientras se va construyendo la solución. La frecuencia del uso del modelo RNN se especifica en un parámetro de entrada que indica con que probabilidad se aplicará el algoritmo RNN para la selección de una asignación.

En nuestro modelo RNN, definimos una *neurona* como el par (nodo,substream), denotado por v^k, la cual pertenece a una de las siguientes clases de neuronas:

$$\mathcal{A} = \{v^k \mid v \in \mathcal{P}_k \ , \ BW_v^{out} \geq bw_k \ ; \ k = 1, \ldots, K\} \tag{4.5}$$

$$\mathcal{O} = \{v^k \mid v \in \{\tau_{k,1}, \ldots, \tau_{k,M_k}\} \ , \ M_k \geq 0 \ ; \ k = 1, \ldots, K\} \tag{4.6}$$

El conjunto $\mathcal{A}$ representa aquéllos nodos que son capaces de transmitir el/los substream(s) que reciben a otros nodos ya que tienen suficiente ancho de banda disponible; el conjunto $\mathcal{O}$ son los nodos raíces de los subárboles desconectados para todos los substreams. Usaremos $\mathcal{A}^k$ y $\mathcal{O}^k$ para denotar el subconjunto de neuronas de $\mathcal{A}$ y $\mathcal{O}$ respectivamente, donde los nodos de las neuronas están contenidos en el substream σ_k. Además, definimos como *posible asignación* al par (i, j) donde $i \in \mathcal{A}^k$ y $j \in \mathcal{O}^k$, $\forall k = 1, \ldots, K$. Las tasas excitativas e inhibitorias de la red neuronal se establecen como:

$$w_{ij}^+ = \frac{AIC_{ij}}{\overline{AIC}_i}, \quad \text{si } (i,j) \text{ es una } \textit{posible asignación} \tag{4.7}$$

$$w_{ij}^- = 1, \quad \text{en caso contrario} \tag{4.8}$$

donde,

- AIC ("Assignment Improvement Criteria") es la función de mejora de una asignación que corresponde a algunos de los criterios definidos en 4.2.2.
- AIC_{ij} es la evaluación de la función luego de que la asignación (i, j) se realiza.
- $\overline{AIC}_i$ es el promedio de las evaluaciones de todas las posibles asignaciones que contienen a i como padre, formalmente definido como $\overline{AIC}_i = \sum_{j \in \mathcal{O}^k} AIC_{ij} / \left|\mathcal{O}^k\right|$ donde $i \in \mathcal{A}^k, \forall k \in \{1, \ldots, K\}$.

El resto de los parámetros RNN son cero.

Usando esta configuración, nótese que la tasa de disparos es computada como $r_i = \sum_j (w_{ij}^+ + w_{ij}^-)$. Las tasas de conexión han sido elegidas de una forma que permita capturar información sobre las mejoras que se obtienen de cada posible asignación. Esto significa que una neurona excitada tendrá un mayor efecto excitativo sobre aquellas neuronas que al realizar una asignación provoque una mayor mejora según el criterio ávido seleccionado. Para esto, las neuronas *padres* serán excitadas artificialmente, ya que son las únicas que puede emitir señales excitativas a las neuronas *huérfanas*. Formalmente, $q_i = 1$, $\forall i \in \mathcal{A}$. Luego la red neuronal iterará hasta converger usando la ecuación de punto fijo 4.4. A

continuación se presenta una simplificación de la ecuación 4.4 que reduce la complejidad para computar $F(q)$:

$$
\begin{aligned}
q_j &= \frac{\sum_i q_i w_{ij}^+ + \alpha_j}{r_j + \sum_i q_i w_{ij}^- + \beta_j} \\
&= \frac{\sum_i q_i w_{ij}^+}{\sum_i (w_{ji}^+ + w_{ji}^-) + \sum_i q_i w_{ij}^-} \\
&= \frac{\sum_{i \in \mathcal{A}^k} w_{ij}^+}{\sum_i w_{ji}^- + \sum_{i \in \mathcal{A}^l \cup \mathcal{O}, l \neq k} q_i} \\
&= \frac{\sum_{i \in \mathcal{A}^k} w_{ij}^+}{\sum_i w_{ji}^- + (\sum_{p \in \mathcal{A}^l, l \neq k} q_p + \sum_{c \in \mathcal{O}, c \neq j} q_c)} \\
&= \frac{\sum_{i \in \mathcal{A}^k} w_{ij}^+}{|\mathcal{A}| + |\mathcal{O}| - 1 + (|\mathcal{A}| - |\mathcal{A}^k| + \sum_{c \in \mathcal{O}} q_c)} \\
&= \frac{\sum_{i \in \mathcal{A}^k} w_{ij}^+}{2|\mathcal{A}| - |\mathcal{A}^k| + |\mathcal{O}| - 1 + \sum_{c \in \mathcal{O}, c \neq j} q_c}, \ \forall j \in \mathcal{O}^k \text{ con } k = 1, \ldots, K.
\end{aligned}
\tag{4.9}
$$

Una vez que la iteración converja, la neurona *huérfana* j con mayor q_j será seleccionada para formar parte de la asignación. Si $j \in \mathcal{O}^k$, elegiremos al padre $i \in \mathcal{A}^k$ que tenga el mayor $\overline{AIC}_i$. La asignación (i, j) será la seleccionada por el algoritmo.

La Figura 4.10 muestra un ejemplo de una posible red RNN en el instante de seleccionar una asignación. Esta red está compuesta por el conjunto de neuronas *padres* $\mathcal{A} = \{i^1, k^1, k^2\}$ y por el conjunto de neuronas *huérfanas* $\mathcal{O} = \{j^1, m^1, j^2, n^2\}$.

Las flechas completas representan las tasas positivas de excitación, que sucede entre neuronas *padres* y neuronas *huérfanas* del mismo substream. Las flechas punteadas son las tasas inhibitorias para cualquier otra relación. Las flechas punteadas que se ven entre los cuadros que delimitan los árboles de distribución o substreams, representan todas las combinaciones entre las neuronas de un substream con las neuronas del otro substream, que siempre son inhibitorias por no ser factible una asignación de este tipo.

La Figura 4.11 muestra el pseudocódigo del algoritmo RNN. Como se puede apreciar, se tiene como entrada al conjunto de asignaciones candidatas C que contiene todas las combinaciones posibles entre neuronas *padres* y neuronas *huérfanas*. En las línea 1-3, se computan las tasas excitativas para dichas asignaciones posibles que serán necesarias para computar $F(q)$ en las líneas 5 y 8. Luego, se inicializa el vector q que corresponde a las neuronas *huérfanas* (línea 4). Este vector se inicializa usando para cada componente valores cercanos a cero, ya que de esta forma la convergencia es más rápida. Luego, en las líneas 6-10 se presenta la computación iterativa de la ecuación de punto fijo. Gene-

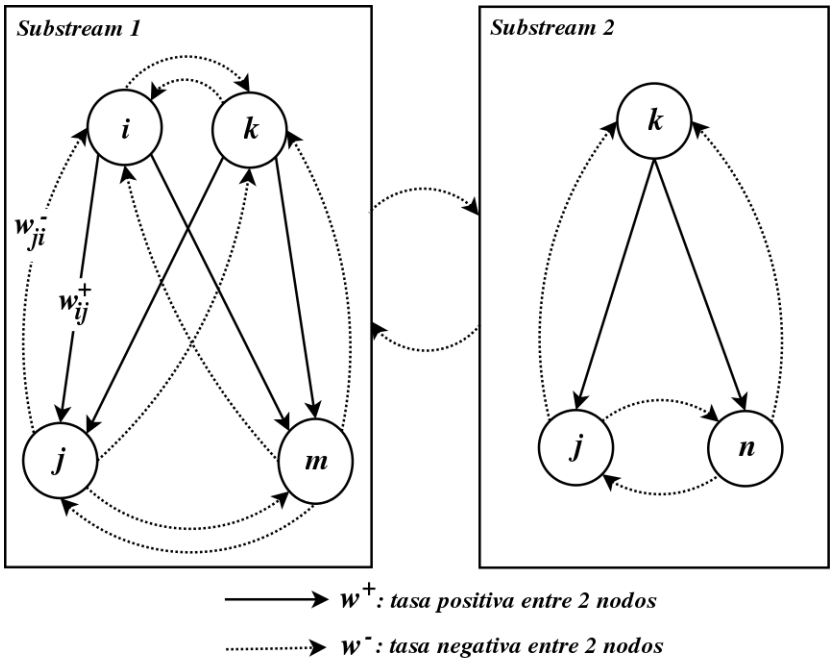


Figura 4.10: Ejemplo de una posible representación RNN

ralmente el número de iteraciones es pequeño. Este proceso iterativo consiste básicamente de comparar el vector computado $F(q)$ con el vector q y evaluar la diferencia de norma en base 2 de estos vectores y comparar este valor con una tolerancia α que se pasa como entrada al algoritmo. En caso de que este proceso no logre la convergencia, se pasa otro parámetro que define la cantidad máxima de iteraciones para permitir continuar $MaxIter$. Una vez que ya tenemos el vector q calculado, se elige aquella neurona j con mayor probabilidad de estar excitada, como se muestra en la línea 11 del algoritmo. Luego, en la línea 12, se elige a la neurona *padre* i que, dentro del mismo substream, tenga la mayor mejora promedio de todas las asignaciones posibles con ella como neurona *padre* (mayor $\overline{AIC}_i$). Finalmente se retorna la asignación (i, j) como solución.

4.4. Otros algoritmos

En esta sección presentaremos dos algoritmos que pueden usarse con el simulador.

El primero se trata de un algoritmo de *fuerza bruta*, es decir, que evalúa todas las posibles combinaciones de asignaciones. Este algoritmo por supuesto está limitado pero se podrán realizar pruebas usando instancias sencillas de pocos nodos y de baja frecuencia en la dinámica de la red, que nos permitirán tener un marco de referencia para comparar los algoritmos heurísticos desarrollados.

El segundo algoritmo, llamado *Proba*, fue desarrollado por Xavier Fisher en el IRISA[3](Rennes, Francia). Este algoritmo fue incorporado con éxito al simulador

[3]http://www.irisa.fr

Procedure Select_RNN_Assignment$_{\text{p2p}}$
Input: C, α, $MaxIter$

1: **for each** $(i, j) \in C$ **do**
2: **compute** w_{ij}^{+} // usando la ecuación 4.7
3: **end for**
4: $q = (q_1, \ldots, q_n) \leftarrow \mathsf{Initialize}(n)$ // vector con n neuronas huérfanas
5: **compute** $F(q)$ // usando la ecuación 4.9
6: **while** $||F(q) - q||_2 > \alpha$ **and** $iter < MaxIter$ **do**
7: $q \leftarrow F(q)$
8: **compute** $F(q)$ // usando la ecuación 4.9
9: $iter \leftarrow iter + 1$
10: **end while**
11: $j^* \leftarrow \underset{j}{\text{argmax}}\{q_j \mid j = 1, \ldots, n\}$
12: $i^* \leftarrow \underset{i}{\text{argmax}}\{\overline{\mathsf{AIC}}_i \mid (i, j^*) \in C\}$ // AIC representa el criterio ávido. Ver 4.7
13: **return** (i^*, j^*)

Figura 4.11: Algoritmo RNN

y la configuración necesaria para su uso se presenta en el apéndice A.

4.4.1. Algoritmo de Fuerza Bruta

El algoritmo de Fuerza Bruta funciona explorando todo el espacio de búsqueda. De esta forma, es posible determinar cuál es la solución óptima global a nuestro problema de optimización. En la práctica este algoritmo sólo funciona para casos donde la cantidad de combinaciones posibles de asignaciones sea pequeña por lo tanto no puede ser usado en la realidad. El único objetivo de su implementación es poder realizar comparaciones de los distintos algoritmos heurísticos implementados con el algoritmo óptimo para casos de desarrollo simples. En el capítulo de pruebas experimentales 5 se presentarán dos casos de prueba sencillos donde será posible correr el algoritmo de Fuerza Bruta. En la Figura 4.12 se puede visualizar el pseudocódigo del algoritmo. De forma similar a los demás algoritmos, se tiene como entrada al grafo g y a la estructura cfg con la configuración necesaria. En este caso en particular, se tiene el parámetro de salida $best_g$, donde se irá almacenando la mejor solución encontrada a través de la ejecución recursiva del algoritmo. El procedimiento comienza buscando todos los posibles candidatos con las asignaciones que se pueden realizar dado el estado actual de la red (línea 1). En la línea 2, se verifica si no hay más candidatos posibles, es decir que ya no se pueden realizar más asignaciones (caso base de la recursión) Si es el caso y además el estado actual de la red es mejor con respecto a la solución anterior encontrada, la solución actual será almacenada como la mejor (línea 4). Luego en la línea 6, se sale del procedimiento para que se aplique la recursión adecuadamente. Si todavía hay asignaciones que pueden aplicarse, se ejecutarán las líneas 8-12, donde se recorre cada asignación posible,

Procedure $\mathsf{BruteForce}_{\mathsf{p2p}}$
Input: g, cfg
Output: $best_g$

1: $C \leftarrow \mathsf{FindAllAssignmentsCandidates}(g)$
2: **if** $C = \varnothing$ **then**
3: **if** $(\mathsf{PSQA}_{\mathsf{expected}}(g, cfg) > \mathsf{PSQA}_{\mathsf{expected}}(best_g, cfg))$ **then**
4: $best_g \leftarrow g$
5: **end if**
6: **exit procedure**
7: **else**
8: **for each** $a \in C$ **do**
9: $\mathsf{ApplyAssignment}(g, cfg, a)$ // se aplica la asignación al grafo
10: $\mathsf{BruteForce}_{\mathsf{p2p}}(g, cfg, best_g)$// llamada recursiva
11: $\mathsf{RevertAssignment}(g, cfg, a)$ // se quita la asignación al grafo
12: **end for**
13: **end if**

Figura 4.12: Pseudocódigo del algoritmo de Fuerza Bruta

y para cada una de ellas se le aplica la asignación al grafo (línea 9), para luego realizar la llamada recursiva en la línea 10. Cuando la llamada recursiva finaliza, se revierte la asignación en la línea 11 para que se explore el espacio con la siguiente asignación. Cuando ya se haya realizado todas las llamadas recursivas posibles, $best_g$ contendrá la mejor solución.

4.4.2. Algoritmo Proba

Proba es un algoritmo ávido, que permite encontrar soluciones muy rápidamente. El pseudocódigo se puede visualizar en la Figura 4.13. Se puede notar una estructura muy similar al procedimiento de construcción en el GRASP que se presentó en la Figura 4.8. La principal diferencia con éste último procedimiento reside en las líneas 4-7, Primero se ordena a los candidatos de forma de priorizar aquellas asignaciones que tienen padres con alta probabilidad de permanecer conectados (línea 4). Esta probabilidad se define formalmente como $p'_i = p_i \times \prod_{j \in \Omega} p_j$ donde $\Omega = \{n \mid y^k_{n,i} = 1\}$[4] siendo σ_k el substream del nodo i. Esto se deduce del hecho de que para un nodo, la probabilidad de continuar recibiendo un substream en la siguiente iteración es la probabilidad de que todos los ancestros al nodo permanezcan conectados. Una vez que los candidatos fueron ordenados tomando en cuenta a los padres, se procede a ordenar tomando en cuenta a los nodos huérfanos (línea 5). Esta segunda ordenación prioriza aquellas asignaciones que tienen como nodo raíz a aquellos subárboles de mayor cantidad de nodos. Si dos o más subárboles tienen la misma cantidad de nodos, se elige aquel nodo raíz que tenga mayor ancho de banda disponible. A continuación, se toma la primer asignación de la lista ordenada de candidatos para cada substream (línea 6), obteniendo así una cantidad máxima de los mejores

[4]Véase ecuación 3.2

Procedure $\text{Proba}_{\text{p2p}}$
Input: g, cfg

1: $C \leftarrow$ FindAllAssignmentsCandidates(g)
2: $A \leftarrow \varnothing$
3: **while** $C \neq \varnothing$ **do**
4: $C \leftarrow$ SortCandidatesByParentsProb(g, cfg, C)
5: $C \leftarrow$ SortCandidatesByOrphansSizeAndBw(g, cfg, C)
6: $R \leftarrow$ GetFirstCandidatesBySubtream(g, cfg, C)
7: $a \leftarrow$ SelectBest(R, cfg) // se selecciona la mejor asignación usando alguno de los criterios definidos en la sección 4.2.2
8: $A \leftarrow A \cup \{a\}$
9: $C \leftarrow$ UpdateCandidates(g, a, C)
10: **end while**
11: $g_c \leftarrow$ ApplyAssignments2Graph(g, A)

12: **return** g_c

Figura 4.13: Pseudocódigo del algoritmo Proba

candidatos como cantidad de substreams haya, y éstas se comparan usando el criterio ávido que se haya especificado para finalmente retornar la mejor asignación (línea 6). El resto del procedimiento es idéntico al de la Figura 4.8. Para obtener más información sobre el algoritmo *Proba* véase [41] y [1].

Capítulo 5

Pruebas Experimentales

En el presente capítulo se desarrollarán las pruebas y se discutirán los resultados obtenidos de dichas pruebas. Las pruebas tienen tres objetivos principales:

1. Probar la *correctitud* del simulador
2. *Aplicar* el simulador a un caso de prueba *real*
3. *Comparar* los diferentes algoritmos desarrollados.

Para el primer objetivo se ha desarrollado un programa generador de casos de prueba de desarrollo, que se presentará en la sección 5.1.1, con el cual generaremos un caso de prueba sencillo que permita mostrar en detalle todos los pasos de una corrida del simulador.

El segundo objetivo implica simular el comportamiento de una situación de la vida real para un servicio de distribución de video en vivo. Para esto se nos ha otorgado datos reales de **AdinetTV**[1], un servicio de distribución de video en vivo por Internet focalizado en distribuir contenido para uruguayos alrededor del mundo. A partir de estos datos, podemos obtener información de los usuarios como en que momento se conectaron, cuanto tiempo permanecieron conectados, etc., la cual nos permite crear instancias de prueba "reales" como entrada a nuestro simulador. Luego en la sección 5.2.1 se explica como se realiza este proceso de generación de entradas para el simulador a partir de los archivos de log provistos por AdinetTV.

Para el último objetivo, debemos presentar las instancias de prueba generadas a partir de los datos "reales", y se realizará una comparación entre los distintos algoritmos presentados en el capítulo anterior y se discutirán los resultados obtenidos.

Los pruebas que se presentarán en las próximas secciones fueron corridas en máquinas Pentium 4 y AMD Athlon 64 con 1 GB de RAM usando Linux (distribución Ubuntu 6.10) como sistema operativo.

[1] `http://www.adinettv.com.uy.`

5.1. Casos de prueba de desarrollo

5.1.1. Generación de casos de desarrollo

Para probar la correctitud de los algoritmos de asignación implementados, es conveniente contar con casos de prueba con poca cantidad de nodos. La razón de esto es que en este tipo de pruebas, más que los resultados nos interesa seguir paso por paso el funcionamiento del simulador, analizando exactamente cuales asignaciones se realizaron en cada iteración. Puesto que la cantidad de posibilidades de asignación se vuelve difícil de manejar cuando el número de nodos aumenta, estos casos de prueba deben tener pocos nodos. A continuación, se detalla el método por medio del cual fueron generados estos casos de prueba, así como las características de los casos de prueba generados.

Generación automática de casos

Si bien es posible crear manualmente archivos de entrada con unos pocos nodos para probar el simulador, se decidió desarrollar una aplicación relativamente simple que genere casos de prueba, con el objetivo de facilitar la creación de nuevos casos. Esta aplicación, llamada `tcgen` (por *Test Case Generator*) lee ciertos parámetros a partir de un archivo de configuración y genera un archivo de entrada para el simulador con el formato adecuado.

Configuración del generador

El archivo de configuración a partir del cual el programa `tcgen` genera casos de prueba es el siguiente:

```
# Cantidad de nodos para el caso generado
nodes 10

# Ancho de banda de subida del nodo fuente
source_bw 1024

# Probabilidad de seguir conectado para el nodo fuente
source_p 1.0

# Ancho de banda disponible mínimo para cada nodo de la red
bw_min 0

# Ancho de banda disponible máximo para cada nodo de la red
bw_max 128

# Substreams con sus bitrates correspondientes
substream 0 128
substream 1 256

# Cantidad de iteraciones
iterations 4

# Cantidad mínima de nodos nuevos en cada iteración
```

```
min_in_nodes  0

# Cantidad máxima de nodos nuevos en cada iteración
max_in_nodes  3

# Cantidad mínima de nodos que se desconectan en cada iteración
min_out_nodes 0

# Cantidad máxima de nodos que se desconectan en cada iteración
max_out_nodes 3
```

Características de los casos generados

Los casos de prueba generados por el `tcgen` son creados de acuerdo a los siguientes lineamientos:

- Para el nodo fuente, tanto el ancho de banda como la probabilidad de seguir conectado[2] se obtienen directamente a partir de lo especificado en la configuración del generador.
- La información de los substreams también es directa desde el archivo de configuración (esto es, la cantidad de substreams en los que se divide el stream original y el bitrate de cada uno de ellos).
- Para el caso de prueba se genera una red inicial con la cantidad de nodos especificada en el parámetro `nodes` de la configuración del generador, en la cual la distribución de los nodos se organiza de tal manera que los árboles correspondientes a cada substream estén balanceados. Para simplificar el proceso de generación, los árboles de todos los substreams tienen la misma distribución de los nodos (si bien esto no es del todo realista, facilita enormemente el proceso de generación y se obtiene una distribución adecuada de los nodos, sobretodo teniendo en cuenta que estos casos de prueba se usarán para probar la correctitud del simulador y no pretenden asemejarse a ningún caso real).
- Para cada nodo de la red inicial el ancho de banda de subida se calcula sumando los bitrates de los substreams que esté transmitiendo y un excedente que es en realidad un valor aleatorio entre las dos cotas especificadas en la configuración (`bw_min` y `bw_max`).
- Finalmente, la cantidad de iteraciones que figurarán en el archivo de entrada generado por el `tcgen` corresponde al número especificado en el parámetro `iterations` de la configuración, y la cantidad de nodos que se conectan y desconectan en cada iteración son números aleatorios entre las cotas definidas por `min_in_nodes` y `max_in_nodes` para las conexiones y `min_out_nodes` y `max_out_nodes` para las desconexiones respectivamente.

[2]Según nuestras hipótesis, el nodo fuente siempre está conectado, por lo que esta probabilidad debería dejarse siempre en 1.

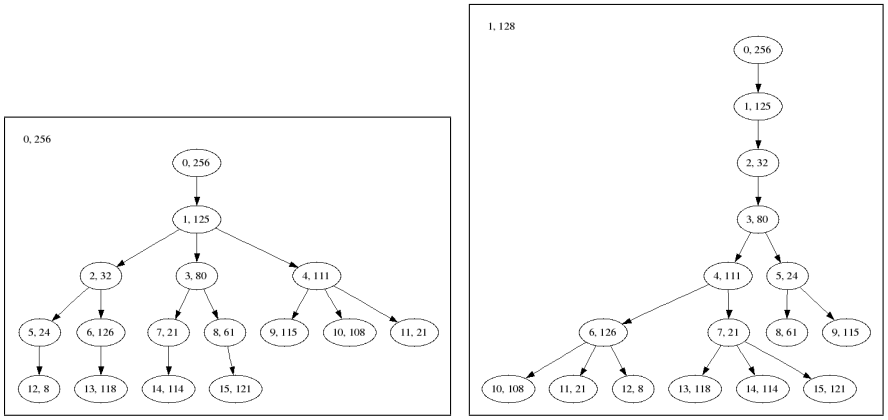
5.1.2. Instancias de prueba

Como se ha mencionado anteriormente, nos interesa generar instancias de prueba simples, con pocos nodos y con baja frecuencia en la dinámica, para poder analizar más adecuadamente el funcionamiento del simulador. Para esto usaremos el generador `tcgen` 5.1.1, con el cual generaremos 2 instancias de prueba que usaremos en el resto de esta sección. Los archivos de entrada y salida del generador para estas instancias están completamente especificados en el apéndice C.

Ademas de validar el funcionamiento del simulador, estas instancias permitirán cumplir con un objetivo aún mayor: comparar los resultados de los algoritmos heurísticos con los del algoritmo de fuerza bruta. Esto es posible por el uso de instancias de prueba sencillos que permiten obtener soluciones usando el algoritmo de fuerza bruta que de otra forma no sería posible.

A continuación describiremos paso a paso la ejecución del simulador para el caso de desarrollo 1.

El caso de desarrollo 1 fue generado para una red con una cantidad inicial de 16 nodos, donde el video se divide en 2 substreams, el primero de 256 kbps y el segundo de 128 kbps.

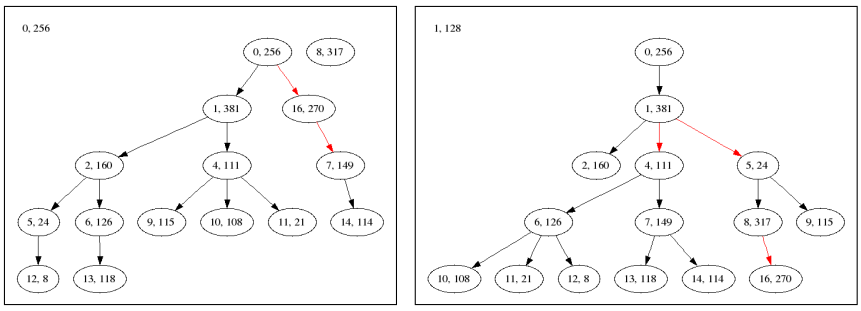


(a) Substream 0 (b) Substream 1

Figura 5.1: Estado inicial de la red

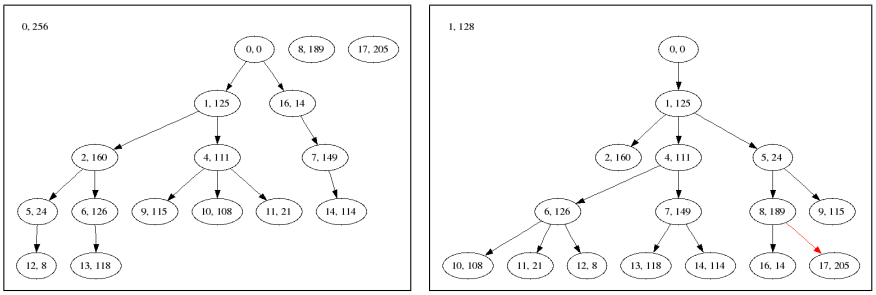
El estado inicial de la red para el primer caso de prueba se puede visualizar en la Figura 5.1 donde la Figura 5.1(a) representa la red de distribución para el primer substream y la Figura 5.1(b) representa la red para el segundo substream.

Antes de continuar, explicaremos la estructura de las figuras. Cada figura individual representa el estado de la red en un instante dado para un determinado substream. El número identificador del substream y su ancho de banda se pueden visualizar en la esquina superior izquierda de la figura. Luego, cada nodo se representa con un óvalo dentro del cual se indica primero el identificador del nodo y luego de la coma, el ancho de banda disponible. Las flechas negras muestran claramente que se esta distribuyendo ese substream entre los nodos correspondientes. Las flechas rojas indican las asignaciones que se van realizando luego que se producen conexiones y desconexiones en la red. El nodo fuente tiene como identificador al 0 y todos los subárboles desconectados y nodos suel-



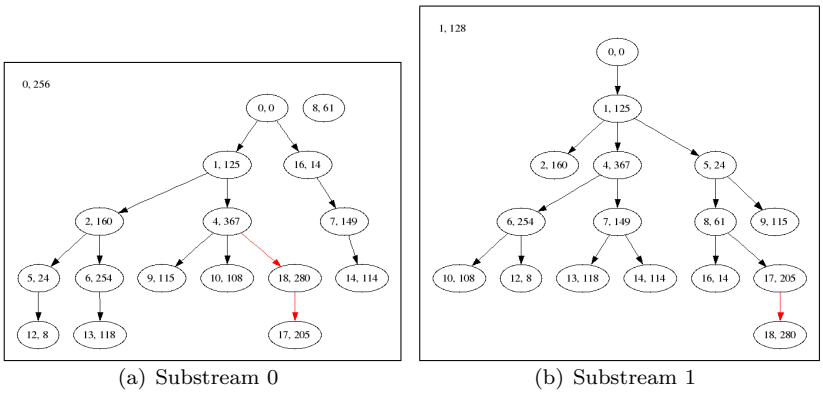
(a) Substream 0 (b) Substream 1

Figura 5.2: Iteración 1



(a) Substream 0 (b) Substream 1

Figura 5.3: Iteración 3



(a) Substream 0 (b) Substream 1

Figura 5.4: Iteración 5

tos se van ubicando a la derecha del subárbol de distribución que tiene como raíz al nodo fuente.

Ahora presentaremos en detalle el primer caso de prueba. Esta simulación consiste en 5 iteraciones o intervalos, luego de las cuáles se evaluará el estado de la red y si se han producido alguna conexión/desconexión se procederá a ejecutar el algoritmo de fuerza bruta, que evalúa todas las posibles combinaciones de

reasignaciones de los afectados y de los nuevos nodos que acabaron de ingresar a la red. Para este caso, solamente se produce alguna conexión y/o desconexión en las iteraciones 1, 3 y 5. Al inicio (Figura 5.1), la red se encuentra en el estado ideal, ya que todos los nodos reciben los dos substreams, es decir, reciben el video con calidad máxima. Durante la iteración 1, el nodo 16 intenta ingresar a la red y los nodos 3 y 15 parten de la misma. Este último no producirá ningún efecto de pérdida de calidad en su partida ya que no transmitía a ningún nodo el video, sin embargo, el nodo 3 transmitía a varios nodos, como se puede visualizar en la Figura 5.1. Por este motivo, al finalizar la primer iteración, tenemos para el substream 0, el subárbol desconectado que tiene como raíz al nodo 7 y los nodos 8 y 16 sin recibir dicho substream. Para el substream 1, tenemos al nodo 16 que acaba de ingresar y a los subárboles que tienen como raíces a los nodos 4 y 5 desconectados. El estado de la red luego de ejecutar el algoritmo de reconfiguración se puede visualizar en la Figura 5.2. Las asignaciones que se realizaron se visualizan con las flechas rojas. Cabe destacar que esta figura representa en realidad el estado inmediatamente anterior a realizar estas asignaciones, motivo por el cual los anchos de banda disponibles de los nodos no se muestran actualizados en la figura. Luego, en la iteración 3, el nodo 17 solicita el ingreso a la red y en la iteración 5, un nuevo nodo (18) aparece y el nodo 11 se desconecta. Las reconfiguraciones realizadas para la iteración 3 y 5 se pueden visualizar en las figuras 5.3 y 5.4 respectivamente.

5.1.3. Comparación de algoritmos

En esta sección se compararán los diferentes algoritmos heurísticos con el algoritmo de fuerza bruta. Para esto debemos primero darle valores a los parámetros configurables de cada algoritmo. A continuación se presenta cada uno de estos parámetros y se discute cada valor seleccionado.

Sorteos

Este parámetro, al cual llamamos `draws`, especifica la cantidad de sorteos futuros que se realizarán a una solución factible. Para más información sobre este parámetro, referirse al apéndice A. Este parámetro es aplicable a todos los algoritmos, por lo tanto tomaremos como referencia al algoritmo de fuerza bruta. Se hicieron pruebas con los valores {3, 5, 10, 20}. Los resultados se pueden visualizar en la tabla 5.1. Esta tabla muestra para los 2 casos de prueba y cantidades de sorteos, la calidad o PSQA promedio de las iteraciones, el cual es un valor entre 0 y 1 siendo 1 la calidad óptima, y el tiempo de ejecución total para la correr la simulación, que está representada en segundos. Los resultados muestran claramente que la mejor calidad promedio de las 5 iteraciones para el caso de prueba 1 es cuando la cantidad de sorteos es 10. Para el caso de prueba 2, el mejor escenario es cuando la cantidad de sorteos es 3. Sin embargo, para 10 sorteos el valor es casi igual al mejor, por lo tanto, elegiremos 10 como la cantidad de sorteos para evaluar la calidad futura esperada.

RCL para el algoritmo GRASP

Los casos de prueba de desarrollo son instancias pequeñas donde sabemos que para el algoritmo GRASP, no tendremos listas numerosas de candidatos

Cuadro 5.1: Evaluación del algoritmo de fuerza bruta para distintos valores de sorteos

Caso	Sorteos	PSQA	$t(s)$
1	3	0.952667	1
	5	0.973333	1
	10	0.980000	3
	20	0.952667	5
2	3	0.928333	0
	5	0.922500	26
	10	0.925833	8
	20	0.907333	77

restringidos. Por esta razón, no intentaremos comparar para distintos valores de RCL porque sabemos que no encontraremos un aporte importante con un análisis de este tipo. Finalmente, se tomará 5 como el tamaño del RCL para realizar las comparaciones entre los distintos algoritmos.

Cantidad de iteraciones para el algoritmo GRASP

La cantidad de iteraciones de GRASP equivale a la cantidad de distintas soluciones factibles construídas, que serán comparadas entre si para seleccionar la mejor de estas. Se realizaron pruebas con los valores de 5, 10 y 20. La tabla 5.2

Cuadro 5.2: Resultados para GRASP variando la cantidad de iteraciones.

Iteraciones	Caso	PSQA (t_i^+)	PSQA (t_i^-)	(PSQA (t_i^+)− PSQA (t_{i+1}^-))
5	1	0.946000	0.830667	0.1544
	2	0.906833	0.818000	0.1524
10	1	0.939500	0.830667	0.1544
	2	0.932500	0.823667	0.1628
20	1	0.850500	0.749667	0.1528
	2	0.937833	0.837500	0.1568

muestra los resultados de ejecutar el algoritmo GRASP para distintas cantidad de iteraciones. Todos los valores de PSQA son promedios respecto a la cantidad de iteraciones[3]. Para el caso de prueba 1 es evidente que el mejor resultado se obtiene cuando la cantidad de iteraciones es 5. Sin embargo, para el caso de prueba 2, en este escenario, obtenemos el peor resultado respecto al PSQA (t_i^+) y al PSQA (t_i^-), aunque en este último caso la diferencia es muy leve. Lo que llama la atención es el último dato, donde obtenemos el mejor valor de toda la tabla. Este valor está implicando que el impacto de pérdida de calidad en iteraciones

[3]PSQA (t_i^+) representa el promedio de los PSQA evaluados luego de ejecutar el algoritmo de reconfiguración y el PSQA (t_i^-) es evaluado inmediatamente antes de la ejecución de algoritmo, t_i denota la iteración o intervalo i, que en este caso en particular se cumple que $i \in \{1, ..., 5\}$

es la menor, lo cual es parte de nuestro objetivo, es decir, minimizar el impacto de pérdida de calidad global por desconexiones. Por este motivo, compararemos los algoritmos usando 5 iteraciones dentro del algoritmo GRASP.

Sorteos para el criterio de PSQA futuro

Para cada asignación individual dentro del proceso iterativo del GRASP, se debe ponderar a las distintas posibles asignaciones mediante algún criterio. El criterio de PSQA futuro, evalúa el PSQA esperado luego de realizar una determinada asignación. La idea es la misma que el parámetro de sorteos definido en 5.1.3. Véase apéndice A para más información. Se ejecutó el algoritmo GRASP usando el criterio PSQA futuro para tres valores de sorteos: 3, 5 y 10. Los resultados se pueden visualizar en la tabla 5.3. Los mejores resultados son obtenidos

Cuadro 5.3: GRASP usando criterio de PSQA futuro.

Sorteos	Caso	PSQA (t_i^+)	PSQA (t_i^-)	(PSQA (t_i^+)− PSQA (t_{i+1}^-))
3	1	0.918667	0.803833	0.1538
	2	0.907667	0.819167	0.1682
5	1	0.843833	0.749667	0.1528
	2	0.890333	0.812833	0.1502
10	1	0.980000	0.857500	0.1550
	2	0.920667	0.830833	0.1442

cuando la cantidad de sorteos es 10. Por lo tanto, es valor será seleccionado para realizar la comparación con los demás algoritmos.

Criterios "ávidos" para el GRASP

El parámetro del criterio ávido del algoritmo GRASP debe ser elegido como alguno de los criterios explicados en la sección 4.2.2. La tabla 5.4 muestra los resultados obtenidos. Como se puede apreciar, hay una leve diferencia entre el

Cuadro 5.4: Evaluación de los distintos criterios ávidos del GRASP

Criterios	Caso	PSQA (t_i^+)	PSQA (t_i^-)	(PSQA (t_i^+)− PSQA (t_{i+1}^-))
Ancho de	1	0.986667	0.857500	0.1550
banda	2	0.924333	0.815833	0.1722
PSQA	1	0.946000	0.830667	0.1544
actual	2	0.906833	0.818000	0.1524
PSQA	1	0.980000	0.857500	0.1550
futuro	2	0.920667	0.830833	0.1442

criterio de ancho de banda y el PSQA futuro. Elegiremos como mejor al PSQA futuro ya que produce para el caso de prueba 2 un menor impacto promedio de pérdida de calidad entre las distintas iteraciones.

Resultados comparativos entre los distintos algoritmos implementados

En el capítulo anterior 4 se presentaron los distintos algoritmos implementados. Estos son: un algoritmo basado en la metaheurística GRASP, un algoritmo similar a este último que difieren en que se intercala el modelo RNN para la búsqueda de soluciones y el algoritmo de fuerza bruta. El algoritmo que usa RNN contiene un parámetro estadístico que indica con que tanta frecuencia se utiliza el RNN y la utilización del mismo se sorteará en base a esta probabilidad. Para la comparación de los algoritmos, presentaremos tres porcentajes posibles: 25 %, 50 % y 100 %. Los resultados de la comparación se pueden visualizar en la tabla 5.5. De estos datos, se puede concluir que GRASP es el mejor algoritmo.

Cuadro 5.5: Comparación de los distintos algoritmos

Algoritmo	Caso	PSQA (t_i^+)	PSQA (t_i^-)	(PSQA (t_i^+)− PSQA (t_{i+1}^-))
Fuerza	1	0.980000	0.857500	0.1550
bruta	2	0.925833	0.839333	0.1410
GRASP	1	0.980000	0.857500	0.1550
	2	0.920667	0.830833	0.1442
GRASP +	1	0.878167	0.776833	0.1534
RNN 25 %	2	0.914333	0.825500	0.1682
GRASP +	1	0.863667	0.749667	0.1528
RNN 50 %	2	0.908833	0.816000	0.1646
GRASP +	1	0.843833	0.749667	0.1528
RNN 100 %	2	0.924667	0.837167	0.1422

El uso del modelo RNN empeora en general los resultados obtenidos, sin embargo para el caso del uso de RNN de 100 % se ve que para el caso de prueba 2, es el mejor algoritmo, quedando muy próximo al de fuerza bruta. En la siguiente sección, se realizarán más pruebas con casos generados a partir de datos reales que nos permitirán obtener una mejor conclusión de como se comportan estos algoritmos heurísticos.

5.2. Casos de prueba "reales"

Una vez que la correctitud del simulador y los algoritmos de asignación implementados fueron probados con suficientes casos de desarrollo, surge la necesidad de evaluar el comportamiento de dichos algoritmos en situaciones más cercanas a la realidad, con una mayor cantidad de nodos y una dinámica similar a la que se daría en una red ya existente. Aquí surge un problema, y es la carencia de redes de streaming de video en vivo basadas en arquitecturas P2P de las cuales podamos obtener información para generar casos de prueba. Sin embargo, notemos que el comportamiento de los clientes en una red de streaming de video es esencialmente el mismo, independientemente de si la arquitectura subyacente está basada en CDN o en P2P; es decir, los clientes se conectan

cuando desean ver el contenido en vivo, y se desconectan cuando pierden interés o cuando termina un programa, o hay una desconexión involuntaria causada por problemas del hardware o software. Si bien las desconexiones afectan más la estructura de una red P2P que la de una CDN clásica, el comportamiento de los clientes es el mismo en ambas. Por lo tanto, consideramos que los datos de los clientes de una CDN pueden ser tan útiles como los de una red P2P a la hora de generar archivos de entrada para el simulador. Vista entonces la posibilidad de utilizar datos de la dinámica de los clientes de una CDN para generar casos de prueba, se obtuvieron varios logs correspondientes al servicio de distribución de contenido AdinetTV. En lo que sigue, se tratará la generación de archivos de entrada para el simulador a partir de los logs de AdinetTV, luego se describirán las instancias de prueba utilizadas y finalmente se presentarán los resultados obtenidos para los diferentes algoritmos probados. Los casos de prueba reales fueron generados en el Instituto IRISA (Rennes, Francia) [1].

5.2.1. Generación de inputs a partir de logs

Tras haber obtenido los logs de AdinetTV, el primer paso fue el desarrollo de un método para generar los archivos de entrada para el simulador a partir de los logs de AdinetTV. Para ello, fue necesario extraer de los logs el comportamiento de los clientes durante un intervalo de tiempo, es decir, cuántas conexiones y desconexiones hubieron, y cuánto tiempo permaneció conectado cada cliente. También se utilizaron estos datos para estimar la probabilidad de que un cliente siga conectado en la próxima iteración, según el tiempo de conexión del cliente y el tiempo transcurrido en el intervalo desde que el cliente se conectó. El bitrate elegido para cada substream surje de la optimización que se detalla en [22] y [1]. Por otro lado, otros datos de los que no se disponía en los logs, como por ejemplo el ancho de banda de subida de cada cliente, debieron ser aleatorizados. Por más detalles acerca de la generación de casos de prueba a partir de los logs de AdinetTV, por favor referirse al Apéndice D.

5.2.2. Instancias de prueba

Para las pruebas que servirán para comparar los diferentes algoritmos, se generaron instancias usando datos de AdinetTV obtenidos durante un partido de fútbol. En un momento dado de la transmisión, un servidor falla en la CDN, forzando la reconexión masiva de los clientes en un período muy corto de tiempo. Esto nos permitirá estudiar los comportamientos de los distintos algoritmos en el caso de una falla mayor.

El bitrate del stream de origen es 512 kbps, y aplicando una redundancia del 25 %, el ancho de banda total es 640 kbps. En la Tabla 5.6 se puede apreciar el bitrate de los cada uno de los substreams para las diferentes configuraciones de streaming multifuente que van desde 1 hasta 10 substreams; las instancias de prueba fueron generadas para cuatro tipos distintos de estas configuraciones: se trata de las divisiones del stream original en 1, 2, 4 y 10 substreams.

Cada instancia tiene una duración total de una hora. Se divide este período en intervalos de 10 segundos para cada reconfiguración, por lo cual cada caso de prueba contiene un total de 360 iteraciones. La Figura 5.5 muestra la dinámica

Cuadro 5.6: Ancho de banda consumido en el servidor i, para la técnica de streaming multifuente con K substreams

i/K	1	2	3	4	5
1	492.7	239.9	109.2	46.5	35.7
2		400.1	188.4	88.9	54.7
3			342.4	171.3	91.0
4				332.4	160.6
5					295.7
total	492.7	640.0	640.0	639.0	637.5
i/K	6	7	8	9	10
1	13.3	11.9	11.1	1.2	0.7
2	24.3	17.0	13.2	2.4	1.3
3	44.8	26.7	17.5	4.8	2.5
4	83.6	45.0	26.0	9.5	5.0
5	157.6	80.5	43.0	18.8	9.8
6	299.4	150.0	76.9	37.2	19.4
7		286.8	144.8	73.9	38.3
8			280.4	146.7	75.7
9				291.5	149.9
10					296.7
total	622.9	617.8	612.8	585.9	599.2

de la red, es decir, la evolución de la cantidad de usuarios conectados durante el período de tiempo considerado. Entre las iteraciones 100 y 150 se puede apreciar una desconexión masiva de nodos provocada por la caída de uno de los servidores de la granja utilizada en la CDN. Como mencionamos anteriormente,

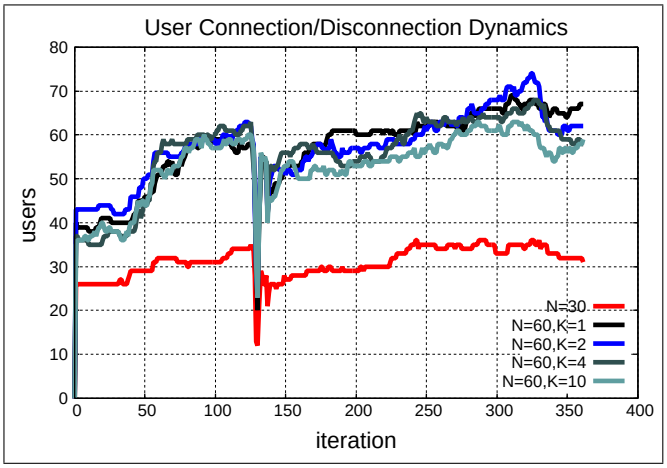


Figura 5.5: Dinámica de las conexiones/desconexiones de clientes

el ancho de banda de subida de cada cliente no está disponible en la información auditada por AdinetTV, por lo tanto se asignaron valores aleatorios, utilizando una distribución uniforme entre 256 kbps y 1024 kbps. En la Figura 5.6 se puede ver el ancho de banda de subida de los nodos. En cuanto a la probabilidad de cada cliente de seguir conectado en la próxima iteración, habíamos mencionado que se estima de acuerdo a los tiempos de conexión de los nodos. La Figura 5.7 muestra los valores de estas probabilidades para los nodos de los casos de prueba generados. Por último, las Tablas 5.7 y 5.8 muestran, para las configuraciones con 2 y 4 substreams respectivamente, como se define la función Q_i de la calidad percibida dependiendo de que substreams está recibiendo el

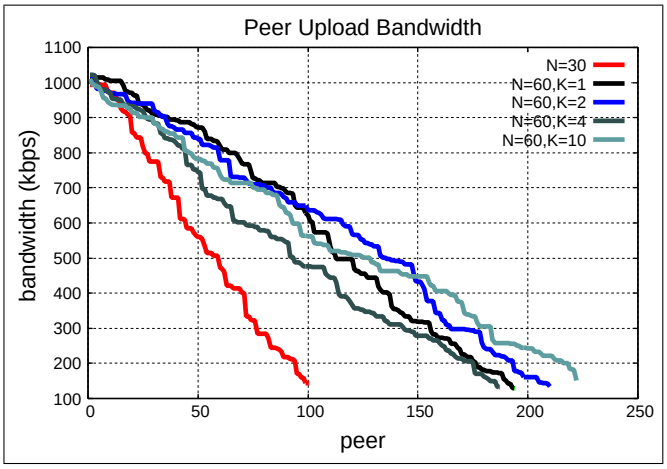


Figura 5.6: Ancho de banda de subida de los clientes

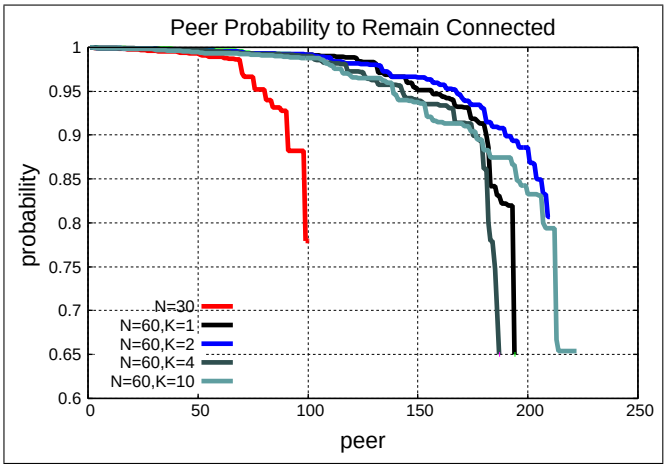


Figura 5.7: Probabilidad de seguir conectado en la próxima iteración

cliente i (el caso para 1 substream es trivial, y la tabla correspondiente al caso de 10 substreams es demasiado grande, por lo cual se omite ya que resultaría más confusa que informativa). Estos valores para la función de calidad surgen de la experimentación, tal como se explica en [1]. Por último, es importante aclalar que en cada caso, el PSQA correspondiente es simplemente el valor de Q_i normalizado entre 0 y 1.

Para realizar las pruebas, consideraremos ocho casos distintos con las características mencionadas anteriormente, cuatro con un máximo de 50 clientes y cuatro con un máximo de 100, donde los cuatro difieren de la cantidad de substreams de la configuración (1, 2, 4 o 10). Por lo tanto, la notación utilizada para identificar a cada una de estas instancias de prueba será de la forma $NcKs$, siendo c la cantidad de nodos promedio y s la cantidad de substreams en la que se divide el stream original. De esta manera, nuestras instancias de prueba son: $N30K1$, $N30K2$, $N30K4$, $N30K10$, $N60K1$, $N60K2$, $N60K4$, $N60K10$. Para las instancias con una misma cantidad promedio de nodos, se utiliza la misma

Cuadro 5.7: Función de calidad percibida, $Q_i = f(y^1_{s,i}, y^2_{s,i})$, para $K = 2$

$y^1_{s,i}$	$y^2_{s,i}$	$f()$
0	0	0.00
0	1	3.64
1	0	2.28
1	1	10.00

Cuadro 5.8: Función de calidad percibida, $Q_i = f(y^1_{s,i}, y^2_{s,i}, y^3_{s,i}, y^4_{s,i})$, para $K = 4$

$y^1_{s,i}$	$y^2_{s,i}$	$y^3_{s,i}$	$y^4_{s,i}$	$f()$
0	0	0	0	0.00
0	0	0	1	2.31
0	0	1	0	1.40
0	0	1	1	4.82
0	1	0	0	1.04
0	1	0	1	3.45
0	1	1	0	2.09
0	1	1	1	7.70
1	0	0	0	0.90
1	0	0	1	2.88
1	0	1	0	1.75
1	0	1	1	6.17
1	1	0	0	1.28
1	1	0	1	4.36
1	1	1	0	2.66
1	1	1	1	10.00

dinámica (es decir, los nodos que entran y salen en cada iteración son los mismos en los 4 casos), mientras que lo que cambia es la cantidad de substreams.

5.2.3. Calibración de los parámetros

Para la comparación de los diferentes algoritmos, es necesario definir los valores de los parámetros que mejor se adecúen para cada algoritmo en particular. El algoritmo GRASP/RNN presenta varios parámetros configurables:

1. Cantidad de iteraciones del GRASP.
2. Tamaño de la RCL (lista restringida de candidatos).
3. Método o criterio ávido que será utilizado por GRASP para priorizar asignaciones (y, en el caso de elegir el criterio "Futuro", también se puede configurar la cantidad de sorteos realizada en cada asignación individual).
4. Porcentaje de utilización del algoritmo RNN dentro del GRASP.

Cantidad de iteraciones del GRASP

La cantidad de iteraciones del algoritmo corresponde a la cantidad de veces que se construye una solución factible diferente con el fin de compararla las demás y tomar la mejor de ellas. Se hicieron pruebas con los valores 3, 5, 10 y 20. Los resultados obtenidos no mostraron mayor diferencia en términos de calidad en las soluciones, por lo cual se eligió 5 como valor a utilizar ya que no incurre en un tiempo computacional tan grande como para 10 y 20, y por otro lado, 3 es un valor que consideramos demasiado pequeño.

Tamaño de la RCL

Por cada caso de prueba, se ejecutó el algoritmo GRASP con tres tamaños distintos de RCL. En particular, exploramos los resultados para los valores de $RCL \in \{5, 10, 25\}$. Los resultados que presentamos para cada tamaño de RCL son los correspondientes al promedio de todos los criterios ávidos de asignación implementados: el criterio "Ancho de banda", que selecciona las asignaciones que signifiquen un mayor aumento en el ancho de banda disponible en la red, "PSQA Actual", que selecciona las asignaciones que implican la mejor calidad global percibida en ese instante, y "PSQA Futuro", que da mayor prioridad a las asignaciones que resultan en una mayor calidad global percibida *esperada* para la próxima iteración (como se mencionó anteriormente, este último criterio es parametrizable en cuanto a la cantidad de sorteos que se hacen para predecir el estado futuro de la red).

Tras la ejecución de las pruebas, se tomó la calidad global percibida (PSQA) de la red luego de ejecutar el algoritmo de asignación en cada iteración. En la Tabla 5.9 se puede ver el promedio de la calidad global percibida según el tamaño de la RCL. Estos valores corresponden al promedio de los resultados obtenidos tomado sobre *todas* las iteraciones. Como se puede apreciar, estos resultados no muestran diferencias significativas en cuanto a la calidad global percibida para los diferentes tamaños de RCL elegidos. Por esta razón, se decidió tomar $RCL = 5$, por ser el valor que implica una menor complejidad computacional.

Cuadro 5.9: Calidad global percibida según tamaño de RCL

Casos	$RCL = 5$	$RCL = 10$	$RCL = 25$
N30K1	0.46	0.46	0.44
N30K2	0.7	0.71	0.69
N30K4	0.81	0.82	0.83
N30K10	0.91	0.91	0.91
N60K1	0.45	0.47	0.46
N60K2	0.59	0.62	0.60
N60K4	0.56	0.56	0.58
N60K10	0.87	0.86	0.89

Criterios ávidos del GRASP

Los algoritmos de asignación implementados utilizan un criterio de selección configurable para elegir, en cada paso del algoritmo, la "mejor" asignación a ser aplicada en la construcción de la solución (por favor referirse a la Sección 4.2.2 para una descripción detallada de estos algoritmos). En este contexto, al hablar de la "mejor" asignación nos referimos a la que proporciona una mejora mayor en la solución que se está construyendo. Los criterios de mejora implementados actualmente son "Ancho de banda", "PSQA actual" y "PSQA futuro". En este último caso, se decidió hacer 3 proyecciones de estados futuros en cada asignación, dado que al incrementar este valor notamos que el tiempo de ejecución se incrementa significativamente, lo cual no es deseable dada la necesidad de que el algoritmo se ejecute en un período corto de tiempo; por otro lado, tampoco se apreciaron mejoras significativas en la calidad percibida de la red al aumentar este valor. Por esta razón, parece adecuado realizar solamente 3 proyecciones a futuro al usar este criterio de mejora.

En la Tabla 5.10 podemos ver el rendimiento de estos tres criterios de mejora, utilizando 3 iteraciones para el GRASP, una RCL con tamaño 5 y las 3 proyecciones a futuro ya mencionadas para dicho criterio.

Cuadro 5.10: Comparación de criterios de asignación

Criterios	BW		Actual		Futuro	
Casos	$t(s)$	PSQA	$t(s)$	PSQA	$t(s)$	PSQA
$N30K1$	0.2	0.55	0	0.43	0.2	0.43
$N30K2$	0	0.79	0	0.68	1.8	0.66
$N30K4$	0.4	0.84	0.2	0.84	13.6	0.81
$N30K10$	16.6	0.95	20.8	0.92	4587.2	0.92
$N60K1$	0	0.61	0.4	0.39	0.8	0.36
$N60K2$	0.2	0.69	0.8	0.6	15.6	0.52
$N60K4$	1	0.72	2	0.53	100.8	0.47
$N60K10$	4.4	0.94	19.4	0.86	2108.2	0.89

De acuerdo con los resultados obtenidos, podemos afirmar con seguirdad que el método basado en el ancho de banda (BW) es el que genera mejores resultados, tanto a nivel de calidad percibida en la red como de tiempo de ejecución, superando con claridad a los demás. Un punto interesante a notar es en cuanto al criterio basado en proyecciones futuras, con sus altos tiempos de ejecución (lo cual era esperable, dado que las operaciones que realiza son mucho más complejas que los otros), pero no por ello obtiene mejores resultados en lo que respecta a la calidad como podríamos suponer, sino que están a la par de aquellos correspondientes al criterio basado en el PSQA actual.

Aplicación del algoritmo RNN dentro del GRASP

Al usar el algoritmo que combina GRASP y RNN, surge la necesidad de definir el porcentaje de uso de RNN. Este valor debe estar entre los límites 0 % (todas las asignaciones se seleccionarán utilizando el algoritmo GRASP) y 100 % (todas las asignaciones se seleccionarán utilizando la técnica basada en RNN). Se ejecutó entonces la simulación usando el criterio de asignación basado en el ancho de banda (de acuerdo a lo comentado en la sección anterior) con 5 valores distintos para el porcentaje de uso de RNN: 0 %, 25 %, 50 %, 75 % y 100 %. La Tabla 5.11 nos muestra los resultados de las pruebas realizadas.

Cuadro 5.11: Calidad global percibida según porcentaje de utilización de RNN

Casos	RNN 0 %	RNN 25 %	RNN 50 %	RNN 75 %	RNN 100 %
$N30K1$	0.54	0.56	0.57	0.51	0.58
$N30K2$	0.81	0.84	0.81	0.82	0.69
$N30K4$	0.91	0.93	0.76	0.9	0.72
$N30K10$	0.98	1	0.97	1	0.81
$N60K1$	0.61	0.61	0.62	0.56	0.64
$N60K2$	0.73	0.72	0.6	0.76	0.64
$N60K4$	0.75	0.75	0.68	0.79	0.62
$N60K10$	0.99	1	0.97	1	0.72
Promedio	0.79	0.80	0.75	0.79	0.68

Por otro lado, se evaluó como impacta el porcentaje de uso de RNN en el tiempo de ejecución del algoritmo. Para esto, se hicieron pruebas utilizando el

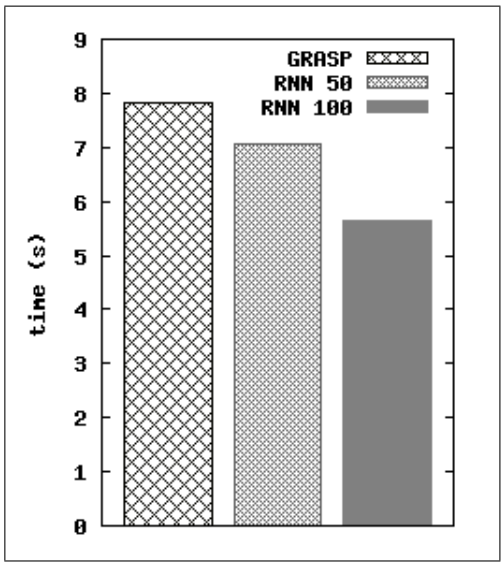


Figura 5.8: Tiempo de ejecución según el porcentaje de uso de RNN

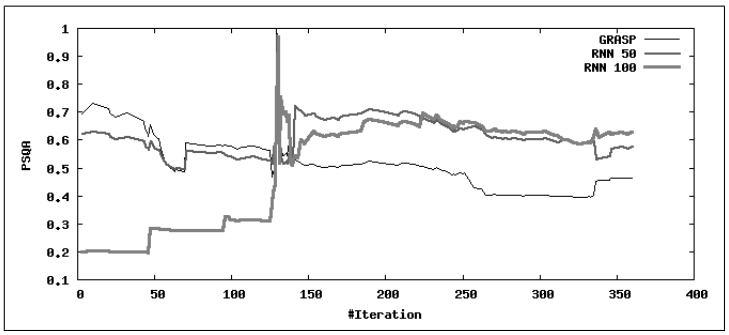
criterio basado en el "PSQA futuro", ya que en este caso los tiempos de ejecución son mayores que en el resto de los algoritmos (como se vio en la sección anterior), y es más fácil ver el impacto del RNN en estos casos ya que se maneja un rango más amplio de tiempos de ejecución. La Figura 5.2.3 permite ver el tiempo de ejecución promedio con 0 %, 50 % y 100 % RNN para el criterio "PSQA futuro".

Otro aspecto interesante del RNN puede verse en las Figuras 5.9(a) y 5.9(b), que muestran la evolución del PSQA a lo largo de toda la simulación para distintos valores del porcentaje de uso de RNN. Allí se puede apreciar que cuando hay mayor uso de RNN, si bien no se genera una distribución de la red tan buena como la del GRASP original, rápidamente se estabiliza hasta obtener buenos niveles de calidad, siendo en la parte final de la simulación incluso mejores que los del algoritmo que no utiliza RNN.

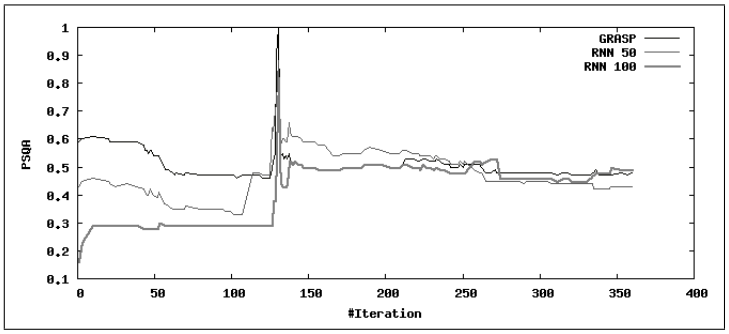
De acuerdo con los resultados obtenidos, el aumento en el uso de RNN puede ayudar a reducir el tiempo de ejecución del algoritmo, pero no necesariamente implica mejoras en la calidad global percibida. Por esta razón, parece aceptable tomar un porcentaje de uso de RNN de por ejemplo 25 % para obtener una buena calidad sin comprometer demasiado el tiempo de ejecución.

5.2.4. Comparación de algoritmos

En esta sección realizaremos una comparación de los diferentes algoritmos heurísticos implementados: por un lado, tenemos el algoritmo basado en GRASP, y sus diferentes variantes según el porcentaje de utilización de la alternativa con RNN descrita anteriormente. Por otro lado, está Proba, el algoritmo descrito en la Sección 4.4.2 que fue implementado en el IRISA (Rennes, Francia). Se ejecutaron entonces los casos generados a partir de logs de AdinetTV descritos anteriormente: $N30K1$, $N30K2$, $N30K4$, $N30K10$, $N60K1$, $N60K2$, $N60K4$ y $N60K10$. Como se indicó en las secciones anteriores, los parámetros de confi-



(a) Evolución del PSQA (criterio "actual")



(b) Evolución del PSQA (criterio "futuro")

Figura 5.9: Evolución del PSQA en el algoritmo basado en GRASP/RNN

guración utilizados para ejecutar estas pruebas fueron los siguientes:

- Cantidad de iteraciones del GRASP = 5
- Tamaño de la lista restringida de candidatos (RCL) = 5
- Criterio de asignación = Ancho de banda (BW)
- Porcentaje de uso de RNN = 25 %

Con el objetivo de evaluar el rendimiento de estos algoritmos, al final de cada iteración se evalúan los siguientes valores, los cuales son promediados sobre el total de iteraciones:

- Tiempo t de ejecución del algoritmo (en segundos)
- Calidad global percibida (PSQA) de la red tras ejecutar el algoritmo de asignación
- Diferencia ΔQ de PSQA ocurrida tras la conexión y desconexión de nodos (esto puede ser descrito como el *impacto* en la calidad provocado por la dinámica de nodos)

En la Tabla 5.12 se pueden apreciar los promedios de estos valores sobre el total de iteraciones para los ocho casos mencionados.

Cuadro 5.12: GRASP+RNN vs. Proba

	GRASP/RNN			Proba		
	$t(s)$	PSQA	ΔQ	$t(s)$	PSQA	ΔQ
$N30K1$	0	0.55	0.01	0	0.58	0.01
$N30K2$	0	0.82	0.02	0	0.84	0.02
$N30K4$	0	0.90	0.02	0	0.94	0.02
$N30K10$	18	0.97	0.03	1	0.98	0.02
$N60K1$	0	0.60	0.01	0	0.61	0.01
$N60K2$	0	0.70	0.02	0	0.72	0.02
$N60K4$	2	0.72	0.02	0	0.48	0.01
$N60K10$	4	0.96	0.03	0	0.97	0.03
Promedio	3.00	0.78	0.02	0.13	0.77	0.02

Al analizar estos resultados, lo primero que observamos es que ambos algoritmos obtienen resultados similares en lo que a calidad percibida se refiere; en todos los casos la calidad promedio es al menos aceptable, y en varias oportunidades es excelente, esto se nota en particular para los casos de prueba en los cuales el stream original está dividido en mayor cantidad de substreams. En efecto, esto era de esperar pues al haber más substreams, éstos tienen un menor bitrate y es más fácil distribuirlos entre los clientes de la red, ya que habrá más nodos que tengan el ancho de banda suficiente para transmitirlos. Justamente en estos casos con más substreams se puede apreciar la principal ventaja del algoritmo Proba con respecto al que está basado en GRASP+RNN: observemos que en los casos con 10 substreams, se puede ver claramente que el algoritmo Proba tiene un tiempo de ejecución notoriamente menor que el otro algoritmo. Esto es una consecuencia del hecho de que el algoritmo basado en GRASP+RNN hace una exploración más detallada del espacio de soluciones que Proba y por lo tanto requiere más tiempo al ejecutarse. Sin embargo, a pesar de esto no notamos que el algoritmo GRASP+RNN obtenga resultados significativamente mejores que los de Proba.

5.3. Conclusiones generales

En este capítulo se cumplieron los objetivos planteados al inicio: esto es, probar la correctitud del simulador y los algoritmos implementados (lo cual se hizo con la ayuda de casos de prueba pequeños), aplicar el simulador a un caso real (para lo cual se generaron archivos de entrada a partir de logs de un servicio de distribución de video ya existente) y comparar los algoritmos de asignación basados en heurísticas que han sido implementados hasta el momento, y que habían sido introducidos en capítulos anteriores. Evidentemente, la facilidad con la que se puede por ejemplo implementar nuevos algoritmos para el simulador abre muchas posibilidades para trabajos futuros, esto se discute en el próximo capítulo.

Capítulo 6

Conclusiones

En este trabajo hemos explorado un modelo teórico de asignación robusta para resolver el problema de la dinámica de nodos en una red P2P de distribución de video en vivo. Este modelo presenta un problema de optimización combinatorio, para el cual hemos desarrollado dos algoritmos que lo resuelven basados en las metaheurísticas GRASP y GRASP+RNN.

El primer algoritmo $\mathsf{GRASP}_{\mathsf{p2p}}$, está basado en la metaheurística GRASP y permite una alta configuración. Uno de los parámetros más destacables es el criterio ávido que se utiliza para evaluar a las posibles asignaciones individuales y así crear la lista de candidatos restringidos durante la fase de construcción del GRASP. Hemos desarrollado tres criterios para evaluar una asignación individual: la que favorece el ancho de banda disponible total en los árboles principales de distribución, la que favorece la calidad global actual de la red y la que favorece la calidad global esperada futura de la red, esto último se realiza sorteando posibles estados futuros de la red luego de realizada la asignación y finalmente se promedia los valores de las evaluaciones de calidad aplicados a los posibles estados futuros de la red. Cabe destacar que el algoritmo se ha implementado permitiendo la fácil incorporación de nuevos criterios ávidos al momento de comparar distintas posibles asignaciones individuales mientras se va construyendo una solución.

El segundo algoritmo, que llamaremos $\mathsf{GRASP\_RNN}_{\mathsf{p2p}}$, es una versión más completa del primer algoritmo que incorpora el uso del modelo RNN como algoritmo para seleccionar una asignación individual durante el proceso constructivo del GRASP. La idea es variar la forma de exploración en la búsqueda de soluciones. Sin embargo, no intentamos reemplazar por completo el método de selección de GRASP sino que iremos intercalando ambos métodos donde la frecuencia de intercalación se especifica en la configuración del algoritmo.

Para probar éstos algoritmos hemos desarrollado un simulador, que fue diseñado para permitir una fácil incorporación de nuevos algoritmos y donde se provee varias bibliotecas útiles y eficientes para el manejo de las estructuras necesarias para trabajar sobre una red de este tipo. Esto se puede comprobar con el algoritmo *Proba* que fue desarrollado en el instituto IRISA (Rennes, Francia) de forma externa a nuestro proyecto y actualmente se encuentra funcionando satisfactoriamente. El simulador también permite analizar de forma visual el estado de la red en cada iteración o intervalo de tiempo, pudiendo detectar por ejemplo cuáles fueron las asignaciones que se realizaron luego de correr algún

algoritmo de reconfiguración.

Un resultado importante es que el simulador fue utilizado en el prototipo real Gol!P2P [1], el cual fue probado su desempeño en la red PlanetLab[1].

Para las pruebas de correctitud de los algoritmos, hemos desarrollado un generador automático de casos de pruebas que sirven de entrada al simulador. Asimismo, hemos logrado ejecutar simulaciones sobre instancias generadas a partir de datos reales provistos por AdinetTV[2], el servicio de distribución de video en vivo por Internet de ANTEL[3]. El estudio comparativo de los resultados de cada algoritmo se realizó usando 8 instancias creadas a partir de datos reales obtenidos de la transmisión de un partido de fútbol.

Los resultados revelan que el uso del modelo RNN respecto al GRASP puro, mejora del orden del 1,5 % la calidad global promedio de la red durante una simulación completa con 360 iteraciones cuando fue aplicado el 25 % de la veces como algoritmo de selección de una asignación individual. Al incrementar el uso del RNN a valores mayores al 25 %, la calidad global promedio disminuye del orden del 5 % al 10 %, sin embargo, también disminuye el tiempo total de ejecución hasta un 30 %, mostrando una mejora significativa en el rendimiento del algoritmo. Hemos realizado comparaciones de nuestro algoritmo $\mathsf{GRASP\_RNN_{p2p}}$ con el algoritmo *Proba*, obteniendo resultados similares en términos de calidad global promedia, sin embargo el algoritmo *Proba* mejora significativamente el rendimiento debido a que realiza una exploración más acotada durante la búsqueda de soluciones. Además, hemos podido observar que el algoritmo *Proba* se comporta mejor en circunstancias de catástrofe como una caída masiva de la red, donde hay que reconectar rápidamente. Sin embargo, en circunstancias de dinámica del orden 10 % de la cantidad total de nodos, se obtienen mejores resultados con el algoritmo $\mathsf{GRASP\_RNN_{p2p}}$.

Como trabajo futuro, se podría profundizar en el estudio estadístico de los casos de prueba generados a partir de datos reales, de manera de obtener valores más confiables respecto a la probabilidad de nodo de permanecer conectado en la próxima iteración. En una aplicación de uso real, el mecanismo de control para los usuarios y sus comportamientos deberá ser cuidadoso para intentar obtener valores de probabilidad de permanecer conectado lo más reales posibles.

Otra línea de futura investigación podría ser el análisis de comportamientos específicos de la red (por ejemplo una caída masiva) y estudiar para esos casos, que algoritmos se adecúan mejor. La idea por detrás sería utilizar todos los algoritmos implementados de forma complementaria, usando en cada reconfiguración, el que mejor se adecúe a la situación que se está enfrentando. Esto puede estar acompañado con el desarrollo de nuevos algoritmos o nuevas versiones híbridas de los algoritmos ya existentes.

Por último, se podrían introducir variantes al modelo teórico tales como permitir que existan nodos de respaldo, es decir, que un nodo tenga un padre y un padre de respaldo en caso de que el primero se desconecte. Esto implicaría asimismo realizar cambios en los algoritmos para adecuarse a este tipo de modelo.

[1] `http://www.planet-lab.org`
[2] `http://www.adinettv.com.uy`
[3] `http://www.antel.com.uy`

Apéndice A

Archivos de configuración

En esta sección se describe la estructura de los archivos de configuración utilizados por el simulador para obtener los parámetros utilizados en los diferentes algoritmos. En todos los casos, se trata de archivos de texto plano en los cuales se indican los diferentes valores. Es posible también insertar comentarios en estos archivos: todas las líneas que comiencen con el caracter `#` serán ignoradas por el simulador.

A.1. Configuración del motor

A.1.1. Estructura del archivo

Para poder ejecutar el simulador, debe existir en el directorio actual un archivo llamado `config.txt`, que contiene la configuración de los parámetros utilizados por el motor de la simulación. Cada parámetro debe especificarse en una línea con la forma `nombre-parámetro valor-parámetro`. Este archivo tiene la estructura siguiente:

```
# P2PTV Simulator -- Configuration file

# Input file
input input.txt

# Output file
output output.txt

# Number of draws
draws 3

# Max. nodes connected
max_nodes 10000

# DOT file name prefix
dot dot_input

# Random seed
```

```
seed 543

#Assignment algorithm
algorithm grasp
```

La presencia de todos estos parámetros es necesaria para el correcto funcionamiento del simulador. A continuación se explicará detalladamente la función de cada uno de estos parámetros durante la ejecución del simulador.

A.1.2. input

En el parámetro `input` debe especificarse el nombre del archivo de entrada que se desea utilizar para la simulación. El archivo de entrada también es un archivo de texto plano en el cual se indican las entradas y salidas de nodos que se producirán durante la simulación, así como la cantidad y bitrates de los substreams a utilizar, y por último las características de los nodos que participarán de la simulación (ancho de banda de subida, probabilidad de seguir conectado).

A.1.3. output

En el parámetro `output` debe especificarse el nombre del archivo de salida que se desea utilizar para la simulación. Este archivo de salida será generado durante la simulación, y contiene, para cada iteración una serie de indicadores calculados a partir de las asignaciones realizadas durante la simulación. El objetivo de tener estos indicadores es para poder evaluar el rendimiento de los diferentes algoritmos de asignación utilizados en el simulador.

A.1.4. draws

Este parámetro especifica la cantidad de sorteos que se le aplicará a una solución factible para evaluar la calidad futura esperada. Como se mencionó en la descripción de modelo, el objetivo del problema es maximizar la calidad futura de la red en el próximo intervalo de tiempo y esto se realiza sorteando posibles estados futuros de la red, evaluando sus PSQA y tomando el promedio de éstos valores. La cantidad de éstas evaluaciones de posibles estados futuros está dada por el párametro `draws`.

A.1.5. max_nodes

El parámetro `max_nodes` será utilizado por el simulador en el momento de asignar la memoria necesaria para crear las diferentes estructuras necesarias. Este parámetro corresponde a la cantidad máxima de nodos que estarán conectados simultáneamente durante la simulación. Evidentemente, lo ideal sería usar exactamente la cantidad máxima de nodos que estarán conectados en un momento dado; cuanto más grande sea el valor de este parámetro, mayor será la memoria consumida por el simulador.

A.1.6. dot

En el parámetro `dot` se especifica el prefijo de los nombres de los archivos `.dot` de salida que contienen una representación de la red en forma de grafos.

En cada iteración, se genera un archivo `.dot` por cada substream. Para generar el nombre de cada uno de estos archivos se concatena el prefijo indicado en el parámetro `dot` con el número de iteración, un infraguión (_) y finalmente el número de substream y la correspondiente extensión `.dot`. Por ejemplo, el archivo

`dot_input142_2.dot`

corresponde al grafo generado para el substream 2 en la iteración número 142, habiendo utilizado `dot_input` como valor del parámetro `dot`.

A.1.7. seed

El valor de la semilla utilizada para generar números pseudo-aleatorios durante la simulación se especifica en el parámetro `seed`. Para las corridas que utilicen el mismo valor en el parámetro `seed`, los números pseudo-aleatorios generados serán los mismos. Por lo tanto, este parámetro puede usarse para garantizar que dos corridas sean idénticas, aunque el algoritmo de asignación utilice números aleatorios. Esto es muy útil a la hora del debug de alguna corrida que pueda estar fallando, de este modo se puede ejecutar una y otra vez la misma corrida para detectar el error.

A.1.8. algorithm

En el parámetro `algorithm` se indica el algoritmo de asignación utilizado por el simulador. Al momento de realizar este documento, existen tres valores posibles para este parámetro: `grasp`, `proba` y `greedy`. Cada uno de estos parámetros corresponde a los algoritmos de asignación implementados hasta el momento; en caso de implementarse otro algoritmo, se debe modificar el simulador para que acepte otros valores en este parámetro. Para cada algoritmo utilizado, se necesita el archivo de configuración correspondiente, esto se discute en la próxima sección.

A.2. Configuración GRASP

A.2.1. Estructura del archivo

El algoritmo basado en GRASP utiliza un archivo de configuración `grasp.txt` que debe encontrarse en el directorio actual al ejecutar el simulador. La estructura de este archivo es la siguiente:

```
# GRASP configuration file

# Number of assignments
assignments = 5

# Restricted candidate list size
rcl_size = 4

# Assignment method
method = "future-psqa"
```

```
# Number of draws used by future-psqa method
draws = 3

# Use of RNN probability
RNN_use_prob = 0.4

# RNN convergence (tolerance)
RNN_tolerance = 1e-35

# RNN convergence (maximum no. of iterations)
RNN_max_iteration = 10
```

Ahora veremos en detalle cada uno de los parámetros utilizados por el algoritmo basado en GRASP.

A.2.2. assignments

La cantidad de soluciones factibles generadas en la fase de construcción del GRASP se indica en el parámetro `assignments`. Recordemos que de entre todas las soluciones factibles elegimos la que determine un PSQA esperado mayor.

A.2.3. rcl_size

El parámetro `rcl_size` corresponde al tamaño de la lista restringida de candidatos utilizada por el algoritmo GRASP, en la cual se almacenan las "mejores" asignaciones posibles (es decir, qué árbol desconectado colgar primero y en qué nodo colgarlo) a realizar en un momento dado según un criterio definido (ancho de banda, PSQA actual, PSQA futuro).

A.2.4. method

Aquí se indica qué método utilizar para determinar qué asignación es mejor realizar en un momento dado. Actualmente este parámetro tiene tres valores posibles, `bandwidth`, `current-psqa` y `future-psqa`, aunque otros criterios de asignación pueden ser implementados en el algoritmo GRASP. A continuación analizamos cada uno de estos tres criterios:

bandwidth

Con este criterio se priorizan las asignaciones que generen un mayor ancho de banda disponible en los nodos del árbol principal tras realizar la asignación; cuanto más ancho de banda genera una asignación, más prioridad tiene.

current-psqa

El criterio `current-psqa` prioriza las asignaciones que impliquen un mayor PSQA resultante en el árbol principal al realizar la asignación. Es decir, lo que se busca con esto es mejorar lo más posible la calidad global percibida *actual* en la red.

future-psqa

Este criterio, a diferencia del anterior, intenta maximizar la calidad global percibida en el próximo instante de tiempo. Para ello se intenta determinar cuáles serán las próximas entradas y salidas de nodos en la red mediante las probabilidades de desconexión de los nodos. Se realiza una cantidad especificada de sorteos y se promedia el PSQA de todos los "estados futuros de la red" generados a partir de los sorteos, obteniendo así un "PSQA futuro" de la red en el próximo instante. Luego, se priorizan las asignaciones que, al realizarlas, impliquen un mayor "PSQA futuro".

A.2.5. draws

El número de sorteos realizados al utilizar el criterio de asignación `future-psqa` se especifica en el parámetro `draws`. Nótese que este parámetro aplica solamente cuando el criterio de asignación elegido es `future-psqa`.

A.2.6. RNN_use_prob

En cada iteración, se realiza un sorteo según la probabilidad indicada en el parámetro `RNN_use_prob` para determinar si se utilizará GRASP o RNN para la búsqueda de soluciones. Como se comentó en la sección correspondiente, de este modo se intercala el uso de GRASP y RNN en la simulación. Este parámetro debe ser un número de punto flotante entre 0,0 y 1,0 ya que se trata de una probabilidad.

A.2.7. RNN_tolerance

Tal como se vio en la sección dedicada a RNN, al resolver la ecuación de punto fijo se itera de modo tal de converger en un cierto valor. La tolerancia utilizada para determinar la convergencia se obtiene a partir de este parámetro. Cabe resaltar que si no se usa RNN, este valor no afecta la ejecución del algoritmo.

A.2.8. RNN_max_iteration

Al igual que para la tolerancia, se fija en este parámetro un número máximo de iteraciones para alcanzar la convergencia en la resolución iterativa de la ecuación de punto fijo. Cabe resaltar que si no se usa RNN, este valor no afecta la ejecución del algoritmo.

A.3. Configuración Proba

El algoritmo Proba, presentado en la sección 4.4.2 utiliza un archivo de configuración `greedy.txt` que debe encontrarse en el directorio actual al ejecutar el simulador. La estructura de este archivo es la siguiente:

```
# PROBA configuration

# Method
```

```
method = "future-psqa"

# Method parameters
draws = 10

#variante
variante = "1"
```

Los parámetros `method` y `draws` son los mismos que se presentaron en la sección anterior A.2. El parámetro `variante` especifica diferentes versiones del algoritmo `greedy`, algoritmo anterior al *Proba* que sirvió de base para su desarrollo. Por más información consultar [41].

A.4. Configuración PSQA

La función PSQA proporciona una medida instantánea de la calidad del video percibida para un cliente en un momento dado. Como vimos anteriormente, esta medida de calidad depende de qué substreams esté recibiendo el cliente en ese momento. Para evitar la complejidad de los cálculos estadísticos requeridos por la función PSQA en el simulador, se encapsuló el componente que realiza el cálculo del PSQA de tal modo de poder asignarle una implementación sencilla y, en caso de querer utilizar el cálculo estadístico real del PSQA en un futuro, poder modificar fácilmente su implementación. En lo que concierne al simulador, simplemente se configura un archivo `psqa.txt` en el cual se indica el valor del PSQA según los substreams que se estén recibiendo en ese momento. Por ejemplo, en el caso de 4 substreams el archivo podría ser el siguiente:

```
0 0 0 0 0.0740028
0 0 0 1 0.231052
0 0 1 0 0.140022
0 0 1 1 0.481704
0 1 0 0 0.104347
0 1 0 1 0.345033
0 1 1 0 0.209019
0 1 1 1 0.770166
1 0 0 0 0.0895008
1 0 0 1 0.287907
1 0 1 0 0.174586
1 0 1 1 0.617045
1 1 0 0 0.128442
1 1 0 1 0.435579
1 1 1 0 0.265818
1 1 1 1 1
```

En cada fila de este archivo, las cuatro primeras columnas corresponden a cada uno de los cuatro substreams, en donde un 0 indica que no se está recibiendo dicho substream y un 1 indica que sí se está recibiendo. La última columna de cada fila indica el valor del PSQA correspondiente; por ejemplo, la cuarta fila del archivo nos dice que el valor de la función PSQA cuando se están recibiendo el tercer y cuarto substreams es de 0,481704.

Apéndice B

Guía de uso del simulador

En esta sección se describe el procedimiento de instalación y ejecución del simulador implementado. El simulador debe ser instalado y ejecutado en un ambiente Linux, en el cual se debe disponer de un compilador de C (se recomienda `gcc`) y la herramienta `make`.

B.1. Instalación

El simulador se distribuye como un archivo empaquetado `simp2ptv.tar.gz`. Para realizar la instalación se debe, en primer lugar, descomprimir el archivo por medio del siguiente comando:

```
tar -zxvf simp2ptv.tar.gz
```

Esto creará un subdirectorio `simulador` en el directorio actual, que contendrá los fuentes del simulador y de las bibliotecas que éste requiera. Una vez extraídos los archivos, debemos acceder al directorio creado y ejecutar el script de configuración previa a la compilación:

```
cd simulador
```

```
./configure
```

Una vez que el proceso de configuración concluye exitosamente, estamos en condiciones de dar inicio a la compilación por medio del siguiente comando:

```
make
```

En caso de que la compilación se realice de forma correcta, se generará un archivo binario ejecutable `simulador` en el subdirectorio `src` del directorio actual. Este será el archivo que utilizaremos para ejecutar el simulador.

B.2. Ejecución

Para ejecutar el simulador, deben existir en el directorio actual los archivos de configuración mencionados en el apéndice anterior, según el algoritmo de asignación con el cual se desea ejecutar el simulador. Luego, para dar inicio

a la simulación se ejecuta el archivo binario `simulador` generado durante la compilación. Tras completar la simulación, se generan en el directorio actual los siguientes archivos:

- Un archivo `output.txt` con diferentes indicadores estadísticos para cada una de las iteraciones ejecutadas en la simulación.
- Un conjunto de archivos `.dot` con las representaciones en formato `dot` de la red en cada iteración para cada substream.

Estos archivos pueden ser convertidos a un formato visualizable (por ejemplo, PostScript) ejecutando el comando siguiente:

```
dot -Tps archivo.dot archivo.ps
```

Un modo más sencillo de hacer esto para no tener que repetir este procedimiento para cada uno de los archivos, es ejecutar el siguiente script Perl `alldot2ps.pl`:

```
#!/usr/bin/perl

@LS = ‘find *.dot -maxdepth 1‘;
foreach $file (@LS) {
        chop($file);
        $file_aux = substr($file, 0, -4);
        $command = "dot -Tps $file_aux.dot -o ps/$file_aux.ps";
        system("$command");
}
```

Apéndice C

Casos de desarrollo

En esta sección se presentan los archivos utilizados para los casos de desarrollo estudiados en la sección 5.1.

C.1. Entradas para el generador

A continuación se presentan los archivos de entrada para el generador de casos de desarrollo `tcgen` 5.1.1 para los dos casos de desarrollo.

C.1.1. Caso de prueba 1

```
# P2PTV Simulator -- Input file generator configuration

# No. of nodes
nodes 16

# Source node bandwidth
source_bw 256

# Source node probability of staying connected
source_p 1.0

# Minimum node bandwidth
bw_min 0

# Maximum node bandwidth
bw_max 128

# No. of new nodes
new_nodes 0

# No. of disconnected nodes
disconnected_nodes 0

# Substreams
```

```
substream 0 256
substream 1 128

#Iterations
iterations 5

min_in_nodes  0
max_in_nodes  1
min_out_nodes 0
max_out_nodes 2
```

C.1.2. Caso de prueba 2

```
# P2PTV Simulator -- Input file generator configuration

# No. of nodes
nodes 8

# Source node bandwidth
source_bw 192

# Source node probability of staying connected
source_p 1.0

# Minimum node bandwidth
bw_min 0

# Maximum node bandwidth
bw_max 150

# No. of new nodes
new_nodes 0

# No. of disconnected nodes
disconnected_nodes 0

# Substreams
substream 0 64
substream 1 128
substream 2 192
substream 3 256

#Iterations
iterations 5

min_in_nodes  0
max_in_nodes  1
min_out_nodes 0
max_out_nodes 1
```

C.2. Entradas para el simulador

Los archivos de entrada para el simulador se obtienen como resultado de ejecutar el generador de casos de prueba usando como entrada los archivos presentados en la sección anterior. A continuación se presentan, para el caso de prueba 1 y 2 respectivamente, los archivos de entrada para el simulador.

C.2.1. Caso de prueba 1

```
# P2PTV Simulator -- Autogenerated input configuration

# Substreams
substream 0 256
substream 1 128

# Nodes
node 0 640 1.000000
node 1 1021 0.520000
node 2 672 0.520000
node 3 848 0.300000
node 4 1135 0.220000
node 5 536 0.800000
node 6 766 0.600000
node 7 661 0.400000
node 8 317 0.110000
node 9 115 0.120000
node 10 108 0.740000
node 11 21 0.500000
node 12 8 0.250000
node 13 118 0.240000
node 14 114 0.630000
node 15 121 0.070000

# Edges
edge 0 1 0
edge 0 1 1
edge 1 2 0
edge 1 3 0
edge 1 4 0
edge 1 2 1
edge 2 5 0
edge 2 6 0
edge 2 3 1
edge 3 7 0
edge 3 8 0
edge 3 4 1
edge 3 5 1
edge 4 9 0
edge 4 10 0
edge 4 11 0
```

```
edge 4 6 1
edge 4 7 1
edge 5 12 0
edge 5 8 1
edge 5 9 1
edge 6 13 0
edge 6 10 1
edge 6 11 1
edge 6 12 1
edge 7 14 0
edge 7 13 1
edge 7 14 1
edge 7 15 1
edge 8 15 0

# Dynamic
iteration 1
new_node 16 270 0.980000
disconnect 15
disconnect 3

iteration 2

iteration 3
new_node 17 205 0.790000

iteration 4

iteration 5
new_node 18 280 0.510000
disconnect 11
```

C.2.2. Caso de prueba 2

```
# P2PTV Simulator -- Autogenerated input configuration

# Substreams
substream 0 64
substream 1 128
substream 2 192
substream 3 256

# Nodes
node 0 1792 1.000000
node 1 903 0.250000
node 2 796 0.610000
node 3 880 0.640000
node 4 584 0.630000
node 5 9 0.000000
node 6 91 0.200000
```

```
node 7 72 0.440000

# Edges
edge 0 1 0
edge 0 2 0
edge 0 1 1
edge 0 2 1
edge 0 3 1
edge 0 1 2
edge 0 2 2
edge 0 3 2
edge 0 1 3
edge 0 2 3
edge 1 3 0
edge 1 4 0
edge 1 4 1
edge 1 5 1
edge 1 4 2
edge 1 3 3
edge 2 5 0
edge 2 6 0
edge 2 7 0
edge 2 6 1
edge 2 5 2
edge 2 4 3
edge 3 7 1
edge 3 6 2
edge 3 7 2
edge 3 5 3
edge 4 6 3
edge 4 7 3

# Dynamic
iteration 1
new_node 8 165 0.140000
disconnect 7

iteration 2

iteration 3
new_node 9 184 0.920000

iteration 4
disconnect 3

iteration 5
disconnect 1
```

Apéndice D

Casos de prueba

D.1. Generacion a partir de logs

D.1.1. Introducción

Como fue mencionado al presentar las pruebas realizadas, se intentó generar casos de prueba en los cuales la red tuviera un comportamiento lo más cercano posible a lo que sería un caso real. Para ello, se obtuvieron logs de las conexiones realizadas a una CDN correspondiente a un ISP (*Internet Service Provider*, Proveedor de Servicios de Internet) de tamaño medio. El objetivo de esto era extraer de estos datos la frecuencia de conexiones y desconexiones de nodos dentro de un intervalo de tiempo, a partir de lo cual se podrían generar archivos de input para el simulador asimilables a un caso real. Esta parte del trabajo fue realizada por Xavier Fischer en el IRISA (Rennes, Francia).

D.1.2. Información extraída

Los logs obtenidos estaban en el formato del servidor de contenido multimedia Windows Media Services[1]. En una primera instancia, se obtuvo la siguiente información de cada cliente conectado:

- Identificador de usuario
- Número de conexión
- Duración de la conexión
- Fecha y hora de conexión
- Fecha y hora de desconexión (calculada a partir del inicio y la duración de la conexión)

D.1.3. Reducción a un intervalo

A continuación se filtraron los nodos para dejar solamente los que estuvieron presentes en algún momento de un cierto intervalo de tiempo. Este intervalo

[1] www.microsoft.com/windows/windowsmedia/9series/server.aspx

fue luego dividido en subintervalos, que corresponden a las iteraciones que se usarán en la simulación. Cabe resaltar que en este punto se eliminaron los casos de aquellos clientes que se conectaron y desconectaron dentro de una misma iteración, ya que estos nodos no tendrían influencia alguna luego de discretizar el intervalo en iteraciones.

D.1.4. Asignación de anchos de banda

A cada nodo se le asignó un ancho de banda aleatorio, de tal modo que la distribución fuera lineal. Este fue el único camino a seguir ya que no se disponía en los logs de ninguna información acerca del ancho de banda disponible en cada nodo.

D.1.5. Cálculo de probabilidades

La probabilidad P de que un nodo siga conectado en la siguiente iteración se calculó de la siguiente manera:

$$P = \frac{1}{n} \sum_{i=1}^{n} \frac{t_c(i)}{|t_c(i)|}$$

en donde los $t_c(i)$ (para $i = 1, 2, ..., n$) son los períodos de tiempo (dentro del intervalo considerado) en los cuales el nodo estuvo conectado a la red.

D.1.6. Generación del archivo de entrada

Teniendo ahora el ancho de banda y probabilidad para cada nodo, así como las conexiones y desconexiones en cada una de las iteraciones definidas, fue posible generar el archivo de entrada para utilizar en el simulador. Por más detalles acerca de la generación de casos de prueba a partir de logs de AdinetTV, en [41] el lector puede encontrar la documentación completa de este proceso.

Bibliografía

[1] P. Rodríguez-Bocca. *Quality-centric design of Peer-to-Peer systems for live-video broadcasting.* PhD thesis, INRIA/IRISA, Univ. Rennes I, Rennes, France, April 2008. Available on-line at: http://goalbit.sourceforge.net/publications/these.pdf.

[2] P. Rodríguez-Bocca, G. Rubino, and L. Stábile. Multi-Source Video Streaming Suite. In *7th IEEE International Workshop on IP Operations and Management (IPOM'07)*, San José, California, United States, 2007.

[3] Daniel De Vera, Pablo Rodríguez-Bocca, and Gerardo Rubino. QoE Monitoring Platform for Video Delivery Networks. In *7th IEEE International Workshop on IP Operations and Management (IPOM'07)*, San José, California, United States, October 31 - November 2 2007.

[4] B. Krishnamurthy, C. Willis, and Y. Zhang. On the use and performance of Content Distribution Networks. Technical Report TD-52AMHL, AT&T Labs – Research, August 2001.

[5] S. D. Gribble, M. Welsh, R. von Behren, E. A. Brewer, D. Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. Joseph, R. H. Katz, Z. M. Mao, S. Ross, and B. Zhao. The Ninja Architecture for Robust Internet-Scale Systems and Services. Technical report, The University of California at Berkeley.

[6] R. Lüling. Hierarchical Video-on-Demand Servers for TV-Anytime Services. In *8th International Conference on Computer Communications and Networks (ICCN)*, pages 110–117, Boston, USA, October 1999. IEEE Press.

[7] X. Jiang, Y. Dong, D. Xu, and B. Bhargava. GnuStream: a P2P Media Streaming System Prototype. Technical report, Department of Computer Sciences, Purdue University, West Lafayette, IN 47907.

[8] T. Sun, M. Tamai, K. Yasumoto, N. Shibata, M. Ito, and M. Mori. MTcast: Robust and Efficient P2P-based Video Delivery for Heterogeneous Users.

[9] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai. Distributing streaming media content using cooperative networking. Technical Report MSR-TR-2002-37, Microsoft Research, One Microsoft Way, Redmond, WA 98052, April 2002.

[10] X. Zhang, J. Liu, B. Li, and T. Yum. Coolstreaming/donet: A data-driven overlay network for efficient live media streaming.

[11] M. Castro, P. Druschel, A. Rowstron, A. Kermarrec, A. Singh, and A.Ñandi. Splitstream: High-bandwidth multicast in cooperative environments. In *SOSP '03*, Bolton Landing, New York, USA, October 2003.

[12] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. Mohr. Chainsaw: Eliminating trees from overlay multicast.

[13] C. Gkantsidis and P. Rodriguez Rodriguez. Network coding for large scale content distribution. In *IEEE Infocom 2005*, 2005.

[14] M. Hefeeda, A. Habib, D. Xu, B. Bhargava, and B. Botev. PROMISE: Peer-to-peer media streaming using CollectCast. In *ACM Multimedia 2003*, pages 45–54, Berkeley, CA, November 2003. ACM Multimedia.

[15] M. Hefeeda, A. Habib, D. Xu, B. Bhargava, and B. Botev. CollectCast: A peer-to-peer service for media streaming. *ACM/Springer Multimedia Systems Journal*, October 2003.

[16] S. Birrer and F. Bustamante. Resilient peer-to-peer multicast without the cost. In *MMCN*, January 2005.

[17] A. Legout. Understanding BitTorrent: An experimental perspective. Technical report, I.N.R.I.A., Sophia Antipolis, France, July 2005.

[18] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh. In *SOSP*, 2003.

[19] Y. Chu, A. Aganjam, T.Ñg, S. Rao, K. Sripanidkulchai, J. Zhan, and H. Zhang. Early Experience with an Internet Broadcast System. In *USENIX Annual Technical Conference*, 2004.

[20] J. Jannotti, D. Gifford, K. Johnsonand M. Kaashoek, and J. O'Toole. Overcast: reliable multicasting with an overlay network. In *Symposium on Operating Systems Design and Implementation*, October 2000.

[21] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal Multicast. Technical report.

[22] A. P. Couto da Silva, P. Rodríguez-Bocca, and G. Rubino. Optimal Quality–of–Experience Design for a P2P Multi-source Video Streaming. In *IEEE International Conference on Communications (ICC 2008)*, San José, California, United States, May 2008.

[23] Hector Cancela, Franco Robledo Amoza, Pablo Rodríguez-Bocca, Gerardo Rubino, and Ariel Sabiguero. A robust P2P streaming architecture and its application to a high quality live-video service. *Electronic Notes in Discrete Mathematics.*, 30C:219–224, 2008.

[24] P. Rodríguez-Bocca, H. Cancela, and G. Rubino. Perceptual quality in P2P video streaming policies. In *50th IEEE Global Telecommunications Conference (GLOBECOM '07)*, Washington D.C., United States, 2007.

[25] P. Rodríguez-Bocca, H. Cancela, and G. Rubino. Video quality assurance in multi-source streaming techniques. In *IFIP/ACM Latin America Networking Conference (LANC '07)*, San José, Costa Rica, 2007.

[26] M.G.C. Resende and C.C. Ribeiro. Greedy Randomized Adaptive Search Procedures. Technical Report TD-53RSJY, AT&T Labs Research, 2002.

[27] M. Poggi de Aragão, E. Uchoa C.C. Ribeiro, and R.F. Werneck. Hybrid local search for the Steiner problem in graphs. In *Abstracts of the 4th Metaheuristics International Conference (MIC 2001)*, pages 429–433, 2001.

[28] T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computionally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

[29] T.A. Feo and M.G.C. Resende. Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[30] S.L. Martins, C.C. Ribeiro M.G.C. Resende, and P.M. Pardalos. A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy. *Journal of Global Optimization*, 17:267–283, 2000.

[31] C.C. Ribeiro, E. Uchoa, and R.F. Werneck. A hybrid GRASP with perturbations for the Steiner problem in graphs. *INFORMS Journal on Computing*, 14(3):228–246, 2002.

[32] E. Gelenbe. Random neural network with negative and positive signals and product form solution. *Neural Computation*, 1(4):502–511, 1989.

[33] E. Gelenbe. Stability of the random neural network model. *Neural Computation*, 2(2):239–247, 1990.

[34] E. Gelenbe and A. Stafylopatis. Global behaviour of homogeneous random neural systems. *Applied Mathematical Modelling*, 15:534–541, 1991.

[35] E. Gelenbe and F. Batty. Minimum cost graph covering with the random neural network. *Computer Science and Operations Research. (New York: Pergamon)*, pages 139–147, 1992.

[36] E. Gelenbe. Hopfield energy of the random neural network. In *Proceedings of the IEEE World Congress on Computational Intelligence*, volume 7, pages 4681–4686, 1994.

[37] A. Ghanwani. A qualitative comparison of neural networks models applied to the vertex covering problem. *Elektrik*, 2(1):11–18, 1994.

[38] E. Gelenbe, A. Ghanwani, and V. Srinivasan. Improved neural heuristics for multicast routing. *IEEE Journal on Selected Areas in Communications*, 15(2):147–155, 1997.

[39] E. Gelenbe, V. Koubi, and F. Pekergin. Dynamical random neural network approach to the traveling salesman problem. In *Proceedings of the IEEE Symposium on Systems Engineering in the Service of Humans*, pages 630–635. Systems, Man and Cybernetics, 1993.

[40] H. Bakircioglu and T. Kocak. Survey of random neural network applications. *European Journal of Operational Research*, 126:319–330, 2000.

[41] X. Fischer. Simulateur P2PTV. Technical report, IRISA, 2007.