UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA UNIVERSITARIA DE INFORMÁTICA

PROYECTO FIN DE CARRERA

Estudio, desarrollo y simulación de protocolos distribuidos enfocados al reparto equitativo de ancho de banda

Miguel Ángel Hernández Castro

Marzo de 2011

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA UNIVERSITARIA DE INFORMÁTICA

PROYECTO FIN DE CARRERA

Estudio, desarrollo y simulación de protocolos distribuidos enfocados al reparto equitativo de ancho de banda

Autor: *Miguel Ángel Hernández Castro*
Titulación: *Ingeniero Técnico en Informática de Sistemas*
Tutores: *Alberto Mozo / Andrés Sevilla*
Curso Académico: *2010/2011*

Programa siempre tu código como si el tipo que va a tener que mantenerlo en el futuro fuera un violento psicópata que sabe dónde vives.
– Martin Golding

PREFACIO

El presente texto constituye la documentación final del Proyecto de Fin de Carrera titulado *"Estudio, desarrollo y simulación de protocolos distribuidos enfocados al reparto equitativo de ancho de banda"*, realizado para la obtención del título en la Escuela Universitaria de Informática de la *Universidad Politécnica de Madrid* por el alumno de Ingeniería Técnica en Informática de Sistemas, *Miguel Ángel Hernández Castro*.
El proyecto en cuestión se llevó a cabo a partir de junio de 2010 y hasta febrero de 2011 bajo la tutoría de don *Alberto Mozo* y don *Andrés Sevilla.*

AGRADECIMIENTOS

En primer lugar quería agradecer a don Alberto Mozo Velasco y a don Andrés Sevilla de Pablo la oportunidad que me han brindado para realizar este proyecto y aprender de ellos, y los departamentos de "*Arquitectura y Tecnología de Computadores*" e "*Informática Aplicada*" de la *Escuela Universitaria de Informática* de la *UPM* el permitirme realizarlo.

También agradecérselo a los miembros del proyecto de investigación "DRESPO", para el que trabajo, sobre cuyo tema de estudio se basaba este proyecto, y que me han orientado durante su realización. A todos mis profesores, desde el colegio hasta la universidad, a todos mis compañeros, mis amigos**,** y a mi familia; por su apoyo, sus consejos, su ayuda, su confianza, y su orientación; ya que todo esto ha llevado a que esté escribiendo estas líneas.

RESUMEN

El objetivo del proyecto es ofrecer un entorno completo de simulación de protocolos de red para la asignación de ancho de banda equitativo de manera distribuida. Para ello se ha creado el entorno "SPBA" (*Simulador de protocolos de asignación de ancho de banda*) basado en el simulador *PeerSim*. Este entorno ofrece la posibilidad de codificar fácilmente los protocolos, y ejecutar simulaciones de distintos tipos y tamaños para comprobar el funcionamiento y el rendimiento de manera sencilla.

El entorno emula de la forma más real posible el tráfico de paquetes por la red virtualizada, y compara automáticamente los resultados que el protocolo ofrece sobre la asignación de anchos de banda con los resultados reales obtenidos de un algoritmo centralizado. Bajo este entorno han sido codificados y probados protocolos de terceros y protocolos para un proyecto de investigación. Además se ha diseñado y codificado un protocolo con nombre "FDBU" (*Distribución equitativa mediante actualizaciones*) que se incluye como muestra en el simulador.

ABSTRACT

The aim of this project is to offer a complete environment for network protocols simulation for distributed and fair bandwidth allocation. To this purpose it has been created the "SPBA" environment (*Simulator Protocols for Bandwidth Allocation*) based on *PeerSim* simulator.
This environment provides the ability to easily implement protocols, and run simulations of various types and sizes to test the functionality and performance.

The environment emulates as realistically as possible the packet traffic through the virtualized network, and automatically compares the results given by the protocol for the bandwidth allocation to actual results obtained from a centralized algorithm. Third-party protocols and protocols for a research project have been implemented and tested under this environment. It has also been designed and implemented a protocol named "FDBU" (*Fair distribution by updating*) which is included as an example in the simulator.

ÍNDICE DE CONTENIDOS

LISTA DE FIGURAS

LISTA DE TABLAS

GLOSARIO DE TÉRMINOS/ACRÓNIMOS

- Downstream - Dígase de un paquete que se envía en la dirección del flujo de información.
- Upstream - Dígase de un paquete que se envía en dirección contraria al flujo de información.
- Backward - Dígase de un paquete que se envía en la dirección opuesta al paquete actual.
- Fairness - Justicia distributiva que se refiere a la asignación justa de recursos (en este caso ancho de banda).
- DFBU - Fair Distribution by Updating. Es el nombre del protocolo desarrollado como ejemplo.
- SPBA - Simulator Protocols for Bandwidth Allocation. Es el nombre del simulador desarrollado.
- Flujo - Cantidad de información que pasa por unidad de tiempo. Caudal de fluido continuo.
- Montículo – Heap. Estructura de datos del tipo árbol con información perteneciente a un conjunto ordenado.
- Nodo - Espacio en el que confluyen parte de las conexiones con otros espacios con los que compartes sus mismas características y que a su vez también son nodos.
- Enlace – Edge. Unión entre dos nodos. Si es bidireccional ambos nodos son emisores y receptores. Si es unidireccional el enlace será emisor para un nodo y receptor para el otro.
- Topología (de red) - Es la cadena de conexiones usada por los nodos que conforman la red para comunicarse. La topología de red la determina únicamente la configuración de las conexiones entre nodos. No pertenecen a la topología de la red, aunque si pueden verse afectados por la misma, la distancia entre nodos, las interconexiones físicas, las tasas de transmisión y los tipos de señales.

  Más en el apartado 1.3 "Características de los protocolos".

CAPÍTULO 1

INTRODUCCIÓN

El proyecto comenzó poco después de que empezase a trabajar en un grupo de investigación. Uno de los objetivos del grupo era diseñar un protocolo distribuido de asignación equitativa de ancho de banda. Mi trabajo se centraba en las simulaciones de los protocolos, para ello he implementado alrededor de 9 protocolos de red y su entorno de simulación basado en *Peersim*. El entorno *Peersim* ha sido modificado en gran parte quedándose con su idea del tratamiento de eventos mediante un montículo (*heap*), y adaptándolo para poder hacer comparaciones entre protocolos, generar estadísticas, comprobar resultados, etc. Así se ha creado un entorno de simulación al que hemos llamado SPBA (*Simulator Protocols for Bandwidth Allocation*).

Parte de los protocolos implementados son "novedosos", creados para la investigación, y otros han sido publicados por otros autores [3 y 4]. La mayoría de los protocolos implementados de terceros no llegaban a estabilizar la red completamente en ciertas situaciones complejas pese a que sus pseudocódigos habían sido publicados en sitios importantes.

En esta memoria descubrirá el entorno de simulación que se ha utilizado y se está utilizando para probar los diferentes protocolos. Además se explicará y probará un protocolo diseñado concretamente para el proyecto de fin de carrera.

1.1 Objetivos

Hoy en día, el determinar cuánto tráfico de cada flujo debe ser admitido por la red satisfaciendo los requisitos de alta utilización de la misma y garantizando la justicia para todas las sesiones, es uno de los retos más grandes en el diseño de las redes de telecomunicaciones.
La pregunta que surge es ¿qué principio seguir para cumplir con algún criterio de justicia a la hora de asignar flujos a las demandas de ancho de banda?
Una primera solución que se nos podría ocurrir es asignar la mayor cantidad de recursos a cada demanda, al mismo tiempo que se intenta mantener los recursos asignados lo más equitativamente posible. Esto nos lleva a utilizar un esquema de asignación de recursos llamado *Max-Min Fairness* ("MMF"). [2,3,5 y 7]

El objetivo del proyecto es desarrollar algún protocolo de asignación de ancho de banda equitativa que sea distribuido y compararlo con otros protocolos publicados por otros autores. Para ello se desarrollará un entorno que permita la simulación lo más realista posible de las transmisiones de paquetes por una red dada.
Se crearán varias redes de distintos tamaños para hacer diferentes tipos de pruebas; generando para cada una de ellas su tabla de enrutamiento centralizada.
Se implementarán protocolos de varios tipos y métodos para comprobar sus resultados por medio de un algoritmo centralizado.
Se desarrollará un programa que se encargue de ejecutar las simulaciones generando las acciones que se le indiquen y elaborando automáticamente gráficas con los resultados obtenidos. Este entorno será usado para las simulaciones de los protocolos desarrollados en el presente trabajo, otros protocolos publicados por terceros, y para el protocolo FDBU.

La memoria de este trabajo comenzará con una Introducción donde se muestren los objetivos y algunos conceptos básicos de este campo. Continuará explicando todos los detalles del entorno de simulación (SPBA), como pueden ser su estructura y su funcionamiento. Tras esto se mostrará la descripción, diseño e implementación del protocolo FDBU. Finalmente se incluirá un manual de usuario que explica de forma clara y sencilla cómo ejecutar una simulación.

1.2 Algoritmos de "max-min fairness"

En las redes de comunicación y de multiplexación, una división de los recursos de ancho de banda se dice que es de "max-min fairness" cuando la velocidad de datos mínima que alcanza un flujo de datos se maximiza.

La ventaja de la equidad max-min sobre FCFS (primer llegado primer servido) es que ofrece una modulación del tráfico, lo que significa que un mal comportamiento de flujo, que consiste en paquetes de datos grandes o ráfagas de paquetes, sólo se castigará a sí mismo y no a otros flujos. En consecuencia, estos algoritmos evitan en cierta medida congestión de la red.

El nombre de "max-min" viene de la idea de que el algoritmo trata de hacer lo más grande posible (máxima) la tasa de los flujos más pequeños (o mínima). De ahí que den una mayor prioridad relativa a los flujos pequeños. Sólo cuando un flujo pide consumir más de C / N (la capacidad del enlace / número de sesiones o flujos) tiene algún riesgo de tener un ancho de banda limitado por el algoritmo. La idea básica de "max-min fairness" es incrementar lo máximo posible el ancho de banda (BW) de una demanda sin que sea a expensas de otra demanda.

Un enlace "saturado" para un flujo de datos es un enlace que se utiliza por completo y todos los flujos de datos de este enlace consumen la máxima velocidad de datos del enlace. Teniendo en cuenta esta definición, no se prohíbe un enlace "saturado" único que se comparta entre varios flujos, y un flujo puede tener más de un "saturado" de igual restricción.

Una asignación de velocidad de datos es "max-min fair", si y sólo si, un flujo de datos entre dos nodos tiene al menos un enlace "saturado".

En todo protocolo "max-min fair" que tenga un funcionamiento correcto, ninguna sesión recibe más de lo que solicita y aquellas cuya demanda no se puede satisfacer se reparten equitativamente el recurso.

Ejemplo sencillo A:

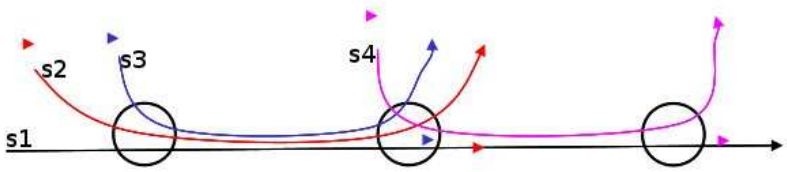

Figura 1 - Ejemplo *max-min*

Ejemplo de reparto de ancho de banda entre cuatro sesiones. En la "Figura 1 - Ejemplo *max-min*" vemos un ejemplo de reparto en que suponeos ancho de banda "1" para cada enlace. El enlace de la izquierda esta "saturado" puesto que entre sus 3 sesiones ("s1", "s2" y "s3") se reparten su ancho de banda (1) y todas ellas consumen el máximo posible (bw/numSes = 1/3). Esto permite que la sesión "s4" obtenga un ancho de banda mayor dado que solo tiene consumido la parte de la sesión "s1" (1- 1/3 = 2/3).

El reparto *max-min* entre las sesiones sería:

```
s1=1/3
s2=1/3
s3=1/3
s4=2/3
```

Ejemplo sencillo B:

En este ejemplo 4 sesiones atacan a un enlace con distinta demanda cada una.

Recurso: 10 (ancho de banda a repartir por el enlace)

Demandas:

```
s1=2
s2=2.6
s3=4
s4=5
```

Reparto:

Repartimos 10 entre 4 sesiones:

```
10/4 = 2.5
Demasiado para la primera sesión.
Asignamos 2 a la primera sesión y sobran 0.5
```

Repartimos los 0.5 entre las 3 sesiones restantes:

```
0.5/3 = 0.167
Asignaciones [2, 2.67, 2.67, 2.67]
```

```
Demasiado para la segunda sesión.
Asignamos 2.6 a la segunda sesión y sobran 0.07.
```

Repartimos 0.07 entre las 2 sesiones restantes.

```
0.07/2 = 0.035
Asignaciones [2, 2.6, 2.705, 2.705]
Total= 2+2.6+2.705+2.705 ≈ 10
```

El reparto final entre las sesiones sería:

```
s1=2
s2=2.67
s3=2.705
s4=2.705
```

1.3 Características de los protocolos

A continuación se describen los distintos tipos de protocolos que se pueden implementar bajo el entorno SPBA según sus caraterísticas.

Distribuidos - Protocolos que reparten su trabajo entre los nodos que lo usan, realizando cada nodo una parte del algoritmo. Son los que trataremos en este proyecto.

Centralizados - Protocolos que realizan todos sus cálculos en un solo sistema.

Asíncronos - Diremos que son asíncronos los protocolos que no respondan a un reloj constante, que no tienen lugar en total correspondencia temporal con otro suceso. Un ejemplo de protocolo asíncrono sería uno que se adate a cambios en la red sin necesidad de sondeo.

Síncronos - Diremos que son síncronos los protocolos que repiten un proceso cada intervalo de tiempo constante. (Por ejemplo hacen sondeo para detectar cambios en la red)

Sondeo – Polling. Protocolos que realizan consulta constante, para crear una actividad sincrónica y detectar así cambios en la red para la reasignación correcta de anchos de banda. En redes con pocos cambios esto genera paquetes innecesarios.

Frugales - Protocolos que no guardan información particular de cada sesión por lo que su memoria no crece con el número de sesiones. Esta característica es muy importante para redes que tienen gran cantidad de sesiones, no siendo factible guardar información de cada una de ellas.

CAPÍTULO 2

ENTORNO DE SIMULACIÓN (SPBA)

En este capítulo se expone el entorno de simulación que ha sido creado para realizar las pruebas de los protocolos de asignación equitativa de ancho de banda. Empieza presentando el entorno *PeerSim*, en el cual se basa nuestro SPBA, y continúa con las modificaciones, estructura y demás detalles del *Simulator Protocols for Bandwidth Allocation*.

2.1 PeerSim

PeerSim [6 y 8] es un interesante entorno de simulación para protocolos P2P en Java. Entre sus características principales destaca, una sencilla configuración y numerosas funciones.

PeerSim se ha desarrollado con una escalabilidad extrema y con la dinamicidad en mente. Se ha puesto a disposición del público bajo la licencia GPL de código abierto. Está escrito en Java y está compuesto de dos motores de simulación, uno simplificado (basado en ciclos) y otro basado en eventos (usado en este proyecto).

Los principales desarrolladores de *PeerSim* son:

Márk Jelasity, Alberto Montresor, Gian Paolo Jesi, Spyros Voulgaris.

Otros participantes y los probadores:

Stefano Arteconi, David Hales, Andrea Marcozz, Fabio Piccon.

2.2 Modificaciones/adaptaciones internas de PeerSim

En el proyecto se ha utilizado el motor orientado a eventos del emulador *Peersim*, haciendo uso del montículo junto con el gestor de memoria. El montículo ha sido modificado para que pueda contener elementos tanto de tipo "paquete" como de tipo "temporizador". Estos son procesados en los instantes que les corresponden. El emulador va procesando secuencialmente eventos del montículo, sacando el siguiente elemento con menor "tiempo", expresando "tiempo" el **microsegundo** en que debe ser procesado dicho elemento.

Para dar mayor realismo a las simulaciones se han agregado colas a cada enlace de salida. Los tiempos de transmisión de los paquetes se han implementado mediante temporizadores que dependen de su tamaño y de las velocidades de los enlaces.

Con el funcionamiento original un nodo enviaba a otro directamente un paquete (sin modelar el tiempo ni el enlace que lo enviaba). Ahora cada nodo hace uso de una tabla de rutas centralizada (por simplicidad) para calcular por qué enlace de salida debe enviar el paquete, le pasa el paquete al correspondiente objeto "Edge" y éste se encarga de enviarlo cuando esté libre y con el tiempo de transmisión adecuado.

Se han hecho también otras modificaciones menores en el simulador original como pueden ser trazas agregadas o quitadas.

2.3 Estructura del entorno

El *Simulator Protocols for Bandwidth Allocation* está implementado en Java. Incluye los paquetes base del *Peersim* y otros agregados para la adaptación al problema concreto.

Consta de los siguientes paquetes principales:

"peersim": contiene las clases del emulador *Peersim* con las modificaciones indicadas que se han realizado sobré el código.

"comunes": contiene las clases principales del emulador creado, y las clases comunes que usan las clases de los protocolos específicos para heredar de ellas.
"algcentral": contiene las clases necesarias para la ejecución del algoritmo centralizado.
Paquetes de protocolos: Además de estos paquetes principales están los correspondientes a cada protocolo implementado.

PeerSim:

Mirar el apartado 2.2 "Modificaciones/adaptaciones internas de PeerSim".

Comunes:

Contiene las siguientes clases:
Accion.java - Clase que indica una acción a llevar a cabo con sus nodos de los extremos (hosts) implicados, su paso (momento de ejecución), y su tipo de acción (close, open).
En el apartado 2.8 "Acciones" puede encontrarse más información.

DijkstraAlgorihm.java - Algoritmo de Dijkstra para calcular la tabla de rutas general sobre el grafo de enrutadores.

EdgeComun.java - Clase de la que heredan las que implementan el código de los enlaces de los protocolos específicos. Ofrece de antemano información sobre los nodos que une, su ancho de banda total, la lista de paquetes de salida, y la programación del envío de paquetes.

EdgeTimer.java - Temporizador usado para programar los envíos de paquetes de la forma más real posible.

Ejecutar.java - Clase principal del emulador creado. Para más información mirar el apartado 2.11 "Programa 'Ejecutar.java'".

Floyd.java - Algoritmo de Floyd para calcular la tabla de rutas general sobre el grafo de enrutadores.

GenerarAcciones.java - Clase usada para la generación de acciones distintas para cada simulación.

NetworkFicheroComun.java - Crea la red usando un fichero de definición. De ésta heredan los protocolos para indicar cuáles son sus clases "Edge", "NodeHost" y "NodeRouter" específicas. Para más información mirar el apartado 2.7 "Creación de la red".

NodeComun.java - Clase abstracta común a todos los nodos de la red. Posee métodos comunes a todos los nodos (de forma abstracta o no), como pueden ser "debug", "error", "processEvent", etc.

NodeHost.java - Clase plantilla (o directa) para los nodos host de la red. Contiene un solo enlace de salida, y por tanto uno de entrada aunque no le pertenece directamente.

NodeRouter.java - Clase plantilla (o directa) para los nodos enrutadores de la red. Contiene una lista de enlaces de salida, y puede ser de tipo *transit* o *stub*.

NodeTimer.java - Temporizador para los nodos (por si son necesarios).

Packet.java - Clase común para la herencia de todos los paquetes de los protocolos específicos.

PacketType.java - Clase común para la herencia de las clases "PaqueteTipo" que representan los tipos de paquetes existentes en cada protocolo específico.

SesionComun.java - Clase que puede usar para extender las clases "Sesion" de los protocolos específicos que ofrece la identificación de la sesión expresada por el origen, el destino, y un contador.

Sim.java - Clase principal del emulador (basada en la clase "Simulator" de *PeerSim*).

TablaRutas.java - Clase serializable que contiene los datos para el encaminamiento de paquetes de todos los enrutadores de la red.

Timer.java - Clase común para todos los Timer que implementa un "Destination". Y por tanto puede ser insertado en el montículo del emulador.

TransportControlComun.java - Clase que implementa un "Control" del emulador. Se encarga de generar estadísticas, cargar acciones del fichero de acciones, ejecutar las acciones, y poner trazas del estado del montículo cada cierto tiempo. De ésta heredan los "TransportControl" de los protocolos, que permiten generación de estadísticas específicas y la generación del fichero de resultados de asignación final llamado "results_<protocolo>.txt".

Util.java - Ofrece procedimientos estáticos útiles como pueden ser la generación de números aleatorios, redondeos, copia de ficheros, búsqueda en ficheros, diferenciación de ficheros, comparaciones con precisión, etc.

AlgCentral:

AlgCentral.java - Clase que compone el núcleo del algoritmo centralizado para el cálculo de las velocidades equitativas asignadas a cada sesión que queda abierta al final de la ejecución de la simulación.

AlgCentral2.java - Versión mejorada del AlgCentral.java.

EdgeAlg.java - Clase que implementa a un enlace usado para la ejecución del algoritmo centralizado. Contiene su identificación mediante su nodo origen y su nodo destino, su velocidad total, y su número de sesiones. También contiene métodos útiles como el cálculo de la velocidad equitativa (*double bwEq()*).

NodeRouter.java - Clase que implementa a un enrutador usado para cargar la topología indicada en la ejecución del algoritmo centralizado.

NodoAlg.java - Clase que implementa a un nodo usado para la ejecución del algoritmo centralizado. Contiene los enlaces que salen de ese nodo (objetos EdgeAlg).

Se.java - Clase que contiene lo necesario de una sesión usada para la ejecución del algoritmo centralizado. Contiene la información que identifica a una sesión como son su origen y su destino, su velocidad asignada, un campo para guardar su velocidad final, y un listado de enlaces por los que pasa (usado en el segundo algoritmo centralizado: AlgCentral2.java). Puede encontrarse más información en el apartado 2.9 "Alg. Centralizado y comparación".

En la "Figura 2 - Herencias" se puede ver un gráfico simplificado de herencias entre clases.

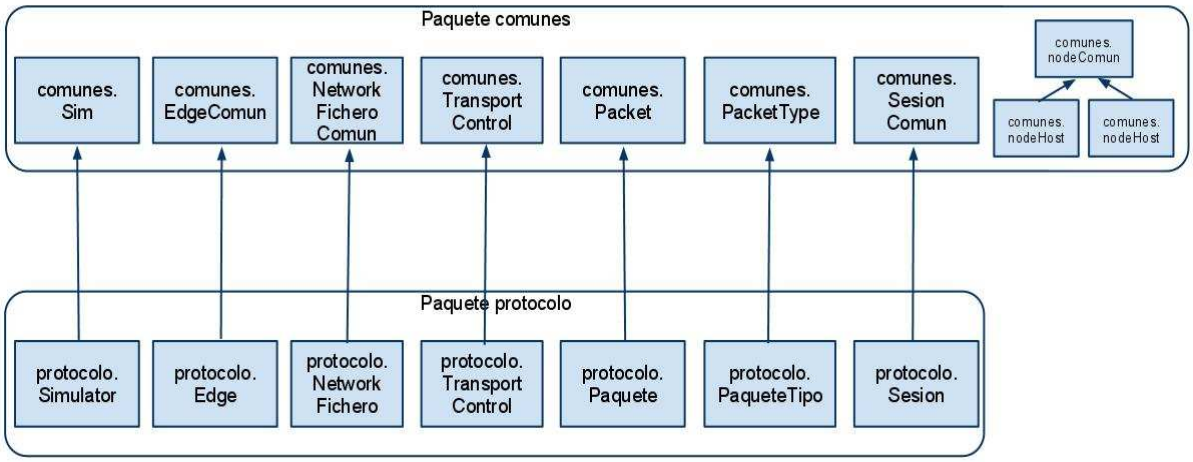

Figura 2 - Herencias

Todos estos paquetes del SPBA son agrupados en el archivo *SPBA_fat.jar* para su fácil uso y distribución. Este archivo comprimido es generado gracias a un plug-in del entorno de desarrollo *Eclipse* llamado "Fat Jar Eclipse Plug-In".

2.4 Tabla de rutas y enrutamiento

Para lograr la dirección de los paquetes se usa una tabla de rutas única para todos los enrutadores para lograr una rápida simulación. Ésta es distinta para cada topología.

Si en la ejecución no se proporciona el fichero de tabla de rutas necesario para la topología a simular, se genera automáticamente uno nuevo y se guarda para simulaciones futuras.
La clase "TablaRutas.java" contiene una tabla de "unsignedShort" de un tamaño "numRouters x numRouters":

```
private short caminos[numRouters][numRouters];
```

Esta clase es Serializable para que un objeto "TablaRutas" puedar ser guardado en un fichero y usado en el futuro sin tener que ser generada la tabla en cada simulación, puesto que su tiempo de creación es largo.
Para la creación del fichero de rutas ("<topología>.rutas") se puede usar el algoritmo de Dijkstra (ejecutado una vez por cada nodo de la red) o el algoritmo de Floyd (ejecutado una sola vez).

Para este proyecto han sido implementados ambos métodos, pero el usado finalmente es Floyd ya que, al tener menor complejidad, es mucho más rápido en ejecución para calcular caminos de todos los nodos a todos los nodos.

De la ejecución del algoritmo no es necesario guardar los caminos de todos los nodos a todos los nodos. Lo que se guarda es el "siguiente nodo" para llegar al nodo destino, el enlace de salida que debe usarse, para llegar al nodo indicado.

Cuando un enlace (edge) quiere enviar un paquete, este paquete es pasado al Nodo actual (el nodo que contiene ese enlace de salida) y éste se lo pasa al enlace de salida correspondiente (que puede ser el mismo que lo intentaba enviar). Los paquetes son enrutados desde los *hosts* por su único enlace de salida y desde los enrutadores (routers) por el enlace indicado en la tabla de rutas.

2.5 Envío de paquetes y colas de salida

Para hacer más reales los tiempos de simulación se han creado en los enlaces colas de salida. Cada vez que se programa un paquete para ser enviado se agrega a esa cola, comprobando si hay actualmente alguno enviándose para enviarle ya o esperar. Cuando se realiza el envío efectivo del paquete se programa un temporizador (insertándolo en el montículo) para que al finalizar el envío (en un tiempo de transmisión *TX*) se intenten enviar más paquetes de la cola si existieran. El paquete llega al destino cuando acabe el tiempo de transmisión más el tiempo de propagación (*TX*+*TP*). El tiempo de transmisión es calculado usando el tamaño del paquete partido del ancho de banda del enlace *(size/bw)*. El tiempo de propagación depende de la configuración "LAN" o "WAN".

La lista de paquetes de salida tiene un tamaño máximo. Si llega un paquete y la cola está llena este será descartado.

2.6 GT-ITM y topologías

Con la ayuda del programa de modelado de topologías grandes, GT-ITM, se ha creado una serie de redes de distintos tamaños para ejecutar las simulaciones.

GT-ITM [1 y 8] es una colección de rutinas para generar y analizar gráficos utilizando una amplia variedad de modelos para la topología de interconexión de redes. Los gráficos se generan en

formato *Don Knuth SGB*, y hay rutinas simples para convertir a un formato que puede ser más fácil de analizar.

Con GT-ITM se crean redes con dos tipos de nodos. Los nodos *transit*, que forman parte del centro de la red, y los nodos s*tub* que están en los flancos de la red generada y de los que cuelgan los nodos host finales.

El programa genera, con un cierto grado de aleatoriedad indicado, un conjunto de dominios *transit*, cada uno de ellos con una cantidad de nodos *transit* dentro de un margen marcado. De cada nodo *transit* saldrán dominios formados por la cantidad señalada de nodos *stub* interconectados entre ellos.

En la "Figura 3 - Redes GT-ITM" se pueden ver el formato de las redes creadas, indicando cuales serían los dominios *transit* y los dominios *stub*.

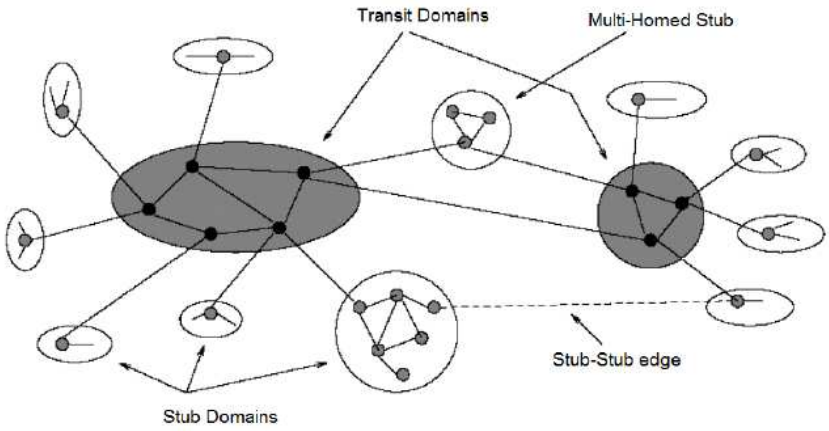

Figura 3 - Redes GT-ITM

En la "Figura 4 - Redes GT-ITM" se puede apreciar como los nodos Host cuelgan de los router *stub* (y no de los router *transit*).

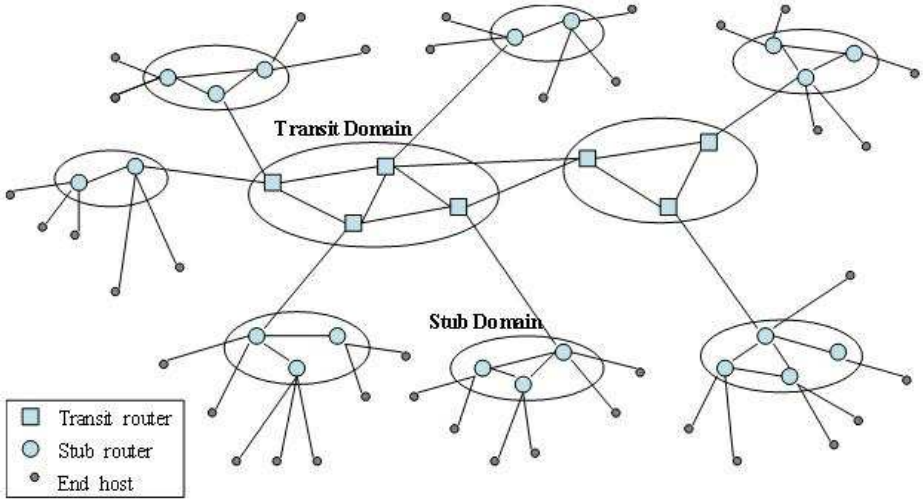

Figura 4 - Redes GT-ITM

Un ejemplo gráfico de una red con un sólo dominio *transit* y 12 dominios *stub* puede ser el de la "Figura 5 - Redes GT-ITM" (sin incluir los nodos host).

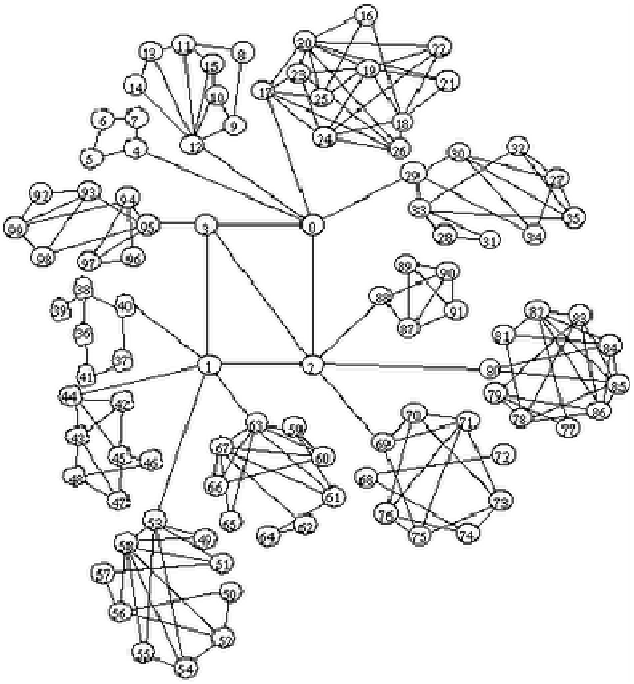

Figura 5 - Redes GT-ITM

Los ficheros de definición de redes contienen sólo los router (tanto *stub* como *transit*). En tiempo de ejecución se calculan y unen a cada nodo *stub* los host indicados en la configuración de la simulación. En la "Tabla 1" se pueden ver los nodos de cada tipo que contienen las redes incluidas en el paquete "SPBU". La "Figura 6 - Red Básica" representa la topología "basic" con 3 *routers transit* y 12 *routers stub*, sin incluir los *hosts.*

Tabla 1 - Topologías

Tipo	Routers	Stub / Transit
basic	15	12/3
small	110	100/10
medium	1100	1000/100
big	11000	10000/1000
verygreat	19500	18750/750
giant	36900	36000/900

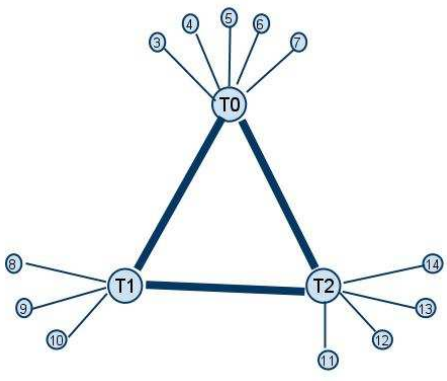

Figura 6 - Red Básica

El número de hosts y el número total de nodos (routers más hosts) se puede calcular en función del valor de configuración "numhost" que indica cuántos hosts cuelgan de cada router *stub*.

Los cálculos a realizar para obtener el número de host son:

<NºHosts>=<NºRoutersStub>*<numhost>

Para cálcular el número total de nodos se realiza la operación:

<NºNodos>=<NºHosts>+<NºRoutersStub>+<NºRoutersTransit>=
=<NºRoutersStub>*<numhost>+<NºRoutersStub>+<NºRoutersTransit>=
=<NºRoutersStub>*(<numhost>+1)+<NºRoutersTransit>

2.7 Creación de la red

Crea la red según el fichero de topología indicado. Para ello hace uso del fichero generado "<topologia>.alt".

Se hace una lectura secuencial del fichero, leyendo cada una de sus partes. Se van creando los nodos Router y enlazándolos como se define en el fichero. Estos Router podrán ser *transit* o *stub*. Más información sobre las topologías en la sección 2.6 *"GT-ITM y topologías"*. De este modo se crean objetos "NodoRouter.java", y se crean objetos "Edge" direccionales de dos en dos (uno en cada dirección) para lograr la bidireccionalidad de todos los enlaces.

Ejemplo:

Como se puede observar en la "Figura 7", el enlace bidireccional entre el "nodo 1" y el "nodo 2" está en realidad formado por dos objetos tipo "Edge". El primero identificado con "origen"=1 y "destino"=2 y perteneciente al "nodo 1"; y el segundo identificado con "origen"=2 y "destino"=1 y perteneciente al "nodo 2".

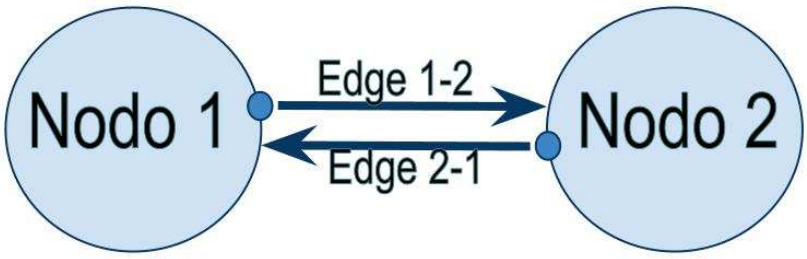

Figura 7 - Edge

Los nodos *host* son creados a continuación (en el número indicado en el parámetro configurable "numHost") estando unidos a nodos de tipo s*tub*.

Cada nodo de tipo *host* tendrá un solo enlace de salida guardado en su atributo "myEdge" (y por tanto un sólo enlace de entrada apuntándole). Y cada nodo de tipo Router tendrá una lista dinámica con sus enlaces de salida guardados en su atributo "myEdges" (En java: *ArrayList<EdgeComun> myEdges*), y tendrá ese mismo número de enlaces de entrada apuntándole.

2.8 Acciones

Para ejecutar las simulaciones se usa un fichero de "acciones" estructurado por líneas con este formato:

```
<time> <tipo> <nodoOrigen> <nodoDestino>
```

Así se realizarán en los momentos oportunos, según la etiqueta "time", el inicio o cierre de la sesión entre los nodos expresados por "nodoOrigen"-"nodoDestino". El "tipo" de acción puede ser "o" (open) o "c" (close).

Un ejemplo puede ser:

```
720 o 648 4088
```

Esta línea expresaría: en el microsegundo 720, el nodo *host* 648 inicia sesión (open) con el nodo *host* 4088.

GenerarAcciones.java

Esta clase tiene un procedimiento principal "ejecutar" que crea acciones en el "ficheroAcciones" indicado en la cabecera. Se le debe indicar cuántas sesiones abrir, cuántas sesiones cerrar, en qué topología y en cuántos pasos:

```
public boolean ejecutar(String ficheroAcciones,int sesAbrir, int
sesCerrar, String ficheroTopologia, int numPasos){
```

Entre las opciones de configuración están:

```
boolean unicas=true; //Si está a true solo una sesión de salida por Host.
boolean reabrir=false; //Permite reabrir una sesión cerrada
```

GenerarAcciones2.java - Esta clase permite la creación de acciones divididas en etapas. Se usa para ver cómo se comporta una red con muchas sesiones estabilizadas cuando se introducen nuevas sesiones. Mediante tres *arrays* se expresa cuantas sesiones se inician, cuantas se cierran y a partir de qué momentos.

```
static int tiemposInicio[]={0,100000,200000,300000};
```

Expresa el instante en que comienza cada ronda de entrada/salida de sesiones. Por tanto indican el paso a partir del cual se abren o cierran una etapa de sesiones.

```
static int numSesionesOpen[]={100000,0,10000,40000};
```

Expresa cuantas sesiones se inician en cada etapa concreta.

```
static int numSesionesClose[]={0,20000,0,40000};
```

Expresa el número de sesiones que se cierran en la etapa concreta. Ese número de sesiones cerradas serán de las abiertas en etapas anteriores y no en la etapa actual. El primer valor debe ser "0" puesto que no hay sesiones abiertas de otras etapas anteriores.

2.9 Algoritmo Centralizado y comparación

Para comprobar que los resultados ofrecidos por los protocolos en las simulaciones son correctos, se comparan con los resultados de un Algoritmo Centralizado.
La ejecución de este algoritmo creará un fichero de resultados para su posterior comparación.

Este algoritmo ofrece el resultado final que debe tener la red, tras la apertura y cierre de todas las sesiones. Para su cálculo analiza el fichero de acciones calculando cuales serán las sesiones que queden abiertas tras finalizar la ejecución (que será cuando se compararán los resultados).

Con esas sesiones calculadas, carga la red con sus *routers* y *hosts*, carga el fichero de rutas y carga las sesiones en todos los nodos por los que pasa.
De este modo queda un registro en cada *router* de cada sesión que pasa por él.

Aquí comienza la ejecución del algoritmo centralizado:

```
Quitar enlaces vacíos
Mientras (quedan sesiones){
      Asignar "bw" equitativo a todas las sesiones de todos los
      enlaces
      Asignar el mínimo de cada sesión a todos los enlaces de su
      camino
      Localizar enlaces saturados y borrar todas las sesiones que
      pasen por esos  enlaces.
```

```
        Quitar enlaces vacíos
    }
```

Tras esto se crea un fichero de resultados ("results_algcentral.txt") con líneas que tienen el siguiente formato:

```
Fin: Host xx-yy ==>zzz.zz
```

Siendo *xx* el nodo origen, *yy* el nodo destino, y *zzz.zz* el ancho de banda con 2 decimales.

En un principio se guardaban los enlaces en un ArrayList. Se comprobó que esto era motivo de una gran complejidad en tiempo de ejecución. Por eso se creó otra estructura auxiliar para ordenar los enlaces en función del nodo origen.

```
private static NodoAlg nodosOrigen[];
```

Así cada NodoAlg contiene un ArrayList de enlaces

```
ArrayList<EdgeAlg> edges;
```

Otra mejora aplicada fue usar un solo objeto sesión común para todos los enlaces que contienen la sesión. Así se simplificaba el paso de asignar el mínimo de cada sesión a todos los enlaces de su camino; dado que todos los enlaces contienen un puntero al mismo objeto "Se" (sesión).

Tras la ejecución del algoritmo centralizado ha sido creado un fichero de soluciones que contiene los anchos de banda con el formato anteriormente indicado. Estos serán mostrados ordenados por el nodo origen, y por el nodo destino en caso de nodo origen coincidente. Todo protocolo, al finalizar su ejecución, creará un fichero de iguales características recorriendo sus nodos *host* en busca de los anchos de banda asignados a sus sesiones finales.

Para comparar los resultados de todos los protocolos, se buscan diferencias entre cada protocolo y el algoritmo centralizado. Para ello se recorren ambos ficheros de resultados secuencialmente comparando línea por línea (al estilo del comando *Linux* "diff").

Con un número de sesiones muy grande el algoritmo centralizado llegaba a tardar tiempos considerados demasiado grandes. Para ello se mejoró de la siguiente manera creando AlgCentral2.java. Se evitó recalcular en cada ronda para cada sesión el ancho de banda mínimo que se le debe asignar por cada enlace; lo que se hace es buscar el enlace que tiene un ancho de banda equitativo menor (ancho de banda / número de sesiones). Solo se encuentra un enlace

"saturado" por iteración, pero ésto permite que la complejidad del problema se reduzca drásticamente.

Además se mejoran las estructuras de datos incluyendo en cada sesión un puntero a los enlaces por los que pasa para mejorar el rendimiento del algoritmo a la hora de eliminarlos.

La parte central del algoritmo quedaría:

```
Quitar enlaces vacíos
Mientras (quedan sesiones){
      Buscar enlace con ancho de banda equitativo (bwEq) menor
      Para cada sesión de ese enlace eliminarla de todos los
      enlaces en los que aparezca (disminuyendo el ancho de banda
      disponible en cada enlace)
      Quitar enlaces vacíos
}
```

2.10 Configuración

El fichero de configuración contiene:

path= directorio de ejecución

codEje= carpeta donde se ejecutará (dentro del directorio de ejecución). Si se deja a "0" va cambiando.

numOpen= número de sesiones que se abrirán en el fichero de acciones a generar.

numClose= número de sesiones que se cerrarán en el fichero de acciones a generar.

numChange= número de sesiones que tendrán un cambio en ancho de banda máximo que solicitan.

numPasos= pasos desde el instante 0 hasta aquel en que se generarán las acciones (en microsegundos)

generarAcciones= a "true" si quieres generar un nuevo fichero de acciones.

ejecutarAlgCentral= a "true" si quieres calcular los resultados del algoritmo centralizado.

ejecutarSimulaciones= a "true" si quieres ejecutar las simulaciones de los protocolos pasados como parámetro al programa.

generarGraficos= a "true" si quieres generar estadísticas y gráficos para cada protocolo.

compararResultados= a "true" si quieres comparar los resultados de todos los protocolos ejecutados con el algoritmo centralizado.

totalPasos= número máximo de pasos que ejecuta el simulador.

cicloPasos= número de pasos que componen un ciclo (por ejemplo de actualización)

pararSiEstabilizado= a "true" si quieres que el protocolo pare la ejecución cuando esté la red estabilizada (si el protocolo lo implementa).

numSimulaciones= número de simulaciones que quieres ejecutar para todos los protocolos indicados en los parámetros de ejecución del programa. A "0" para infinitas (o hasta encontrar error).

debug= a "true" para activar las trazas de depuración.

topologia= indica la topología (debe existir el fichero ".alt" en "<path>/topologias")

numhost= indica el número de *hosts* que cuelgan de cada nodo *stub* de la topología indicada.

velHost= expresa la velocidad de los enlaces de los *hosts*.

velStub= expresa la velocidad de los enlaces que unen *routers stub* entre ellos o con *routers transit*.

velTransit= expresa la velocidad de los enlaces entre los *routers transit*.

lan_wan= a "true" configura el simulador con tiempos de propagación bajos (10 microsegundos) y a "false" configura el simulador con tiempos de propagación entre 1000 y 10.000 microsegundos.

En el apartado 4.2 "Configuración y ejecución" se puede ver un ejemplo.

Estas configuraciones son cargadas por la clase "Ejecutar.java", la cual da un valor por defecto a las variables de configuración que no localiza en el fichero.

2.11 Programa "Ejecutar.java"

Para simplificar y ordenar las ejecuciones de simulaciones se ha creado una clase en java llamada "Ejecutar.java" que, para cada simulación, usa/crea una carpeta de resultados. Este "programa" realizará las ejecuciones y comparaciones que quieras, con la configuración, y número de veces que se le indique.

Este programa debe llamarse indicándole en los parámetros la ubicación del fichero de configuración y seguidamente los protocolos que debe simular. Un ejemplo de llamada puede ser:

```
java comunes.Ejecutar
/home/usuario/Simulaciones/Ejecuciones/config.txt protocolo1
protocolo2
```

En este ejemplo se ejecuta el programa Ejecutar con el fichero de configuración "*/home/usuario/Simulaciones/Ejecuciones/config.txt"* sobre los protocolos *"protocolo1"* y *"protocolo2"*.

La clase, al iniciarse, carga el fichero de configuración (estableciendo valores por defecto a las variables que falten por configurar). Se proporciona más información en el apartado 2.10 "Configuración".

Este programa genera resultados que guarda en la carpeta "results" del directorio de ejecución. Entre otras cosas guarda un *log* general, *logs* específicos, resultados de cada protocolo, resultados del alg. centralizado y estadísticas.

En un principio, este script que se encargaba de ejecutar las simulaciones correspondientes, el algoritmo centralizado y comparar los resultados, estaba implementado en bash. Para facilitar su programación, y dado la complejidad que estaba alcanzando, fue programado en java.

2.12 Directorio de ejecución

Contiene las carpetas:

config - Están los ficheros de configuración de cada protocolo y normalmente también los ficheros para configurar las simulaciones.

Los ficheros de configuración de cada protocolo le indican qué clases debe usar para la creación de la red y sus protocolos de transporte.

Los ficheros de configuración de la simulación tienen la estructura anteriormente explicada en el apartado 2.10 "Configuración".

lib - Están las bibliotecas java necesarias incluyendo el *SPBA_fat.jar*.

results - En este directorio se guardan las carpetas con los resultados de cada simulación. El nombre de la carpeta será el indicado en el parámetro de configuración "codEje". En caso de ser "" (nulo), se generará un nombre aleatorio para cada simulación.

En las carpetas de resultados se guardará un fichero con nombre "*acciones.txt*" que contiene las acciones ejecutadas en dicha simulación. Se generarán los resultados del algoritmo centralizado (si se ha configurado para ejecutarse) y los de todos los protocolos indicados en los parámetros de lanzamiento de la simulación. Los nombres de los resultados tienen el formato "*results_*.txt*". Por ejemplo "*results_algcentral.txt*" o "*results_fdbu.txt*". Estos ficheros de resultados serán usados para comprobar que los resultados del protocolo son correctos.

También se genera un *log* general que muestra los pasos seguidos en la simulación y los errores generales encontrados; teniendo el fichero en cuestión el nombre "log.txt". A su vez se genera un fichero de *log* para cada protocolo ejecutado con el nombre "log_*.txt" siendo el "*" el nombre del protocolo.

En la carpeta de cada simulación, a su vez, se crea otra carpeta llamada "estadísticas" que contiene datos meramente estadísticos sobre "paquetes por sesión", "paquetes/tiempo", "estrategias recibidas", etc. Con estos datos se generan automáticamente gráficas (si así se ha configurado la simulación).

tmp - Por este directorio pasan archivos temporales y otros necesarios para la realización de alguna tarea repetitivamente. Se usa para la generación de los gráficos.

En este directorio hay unos scripts que interpreta "Gnuplot" y, junto con los ficheros de datos de las simulaciones, se generan imágenes que muestran distintos tipos de estadísticas.

Puede verse más información en el apartado 2.13 "Gráficas y estadísticas".

topologías - Esta carpeta contiene los ficheros de cada topología, con el nombre de la topología en cuestión y extensión ".alt". Con ellos, el entorno de simulación genera la

red con las características, nodos y enlaces indicados. Cuando se carga una topología por primera vez para ejecutar un protocolo se crea un fichero con la tabla de rutas de la red; éste tiene el nombre de la topología y extensión ".rutas".
Más información en el apartado 2.6 "GT-ITM y topologías".

2.13 Gráficas y estadísticas

Se generan estadísticas automáticas sobre los paquetes/tiempo generados (siendo tiempo el intervalo entre estadísticas indicado en el fichero de configuración con el parámetro "cicloPasos"). Estos datos estarán en el fichero con el nombre "paq_time.dat".
También se generan estadísticas sobre paquetes/sesión/tiempo. Estos serán guardados en el fichero "paqses_time.dat"
Ambos ficheros de estadísticas tienen un formato como el siguiente:

```
<paso> <paquetes totales> <paquetes tipo 1> <paquetes tipo 2> …
                          <paquetes tipo 20>
```

donde el tipo del paquete es dependiente del protocolo en cuestión, pudiendo haber un máximo de 20 tipos de paquetes con estadísticas.
Ejemplo de una línea del fichero:

```
500 34 8 12 9 2 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Otro tipo de estadísticas que se generan son relativas a los anchos de banda asignados a cada sesión. En el fichero "estrategias.dat" se guarda cada vez que se recibe en un *host* una estrategia, una línea indicando el paso actual y el error entre el ancho de banda indicado y el ancho de banda final (indicado por el algoritmo centralizado).
Tienen un formato sencillo:

```
<paso> <error en la estrategia>.
```

Ejemplos:

```
136 0.0
138 -1.0661157024793388
```

Usando scripts de "Gnuplot", al final de la ejecución del protocolo, el simulador genera automáticamente estadísticas con los datos recopilados.
Se generan cinco gráficos automáticamente:

- paq_time.jpg
- paq_time_tipos.jpg

- paqses_time.jpg
- paqses_time_tipos.jpg
- errorEstrategias.jpg

Estas gráficas son generadas en el directorio "tmp" con los datos obtenidos de la simulación. Luego se copian a la carpeta de "estadísticas" de la simulación concreta y se cambia el nombre de los ficheros de datos y a los gráficos poniéndoles como prefijo el nombre del protocolo en cuestión. Así, si en una simulación se lanza más de un protocolo estarían guardados todos los datos y todos los gráficos con distintos nombres identificativos.

A continuación se muestran algunos ejemplos de gráficas que se generan en la ejecución de cada protocolo:

- paq_time.jpg

  En la "Figura 8" se muestran los paquetes totales que se generan por intervalo de tiempo.

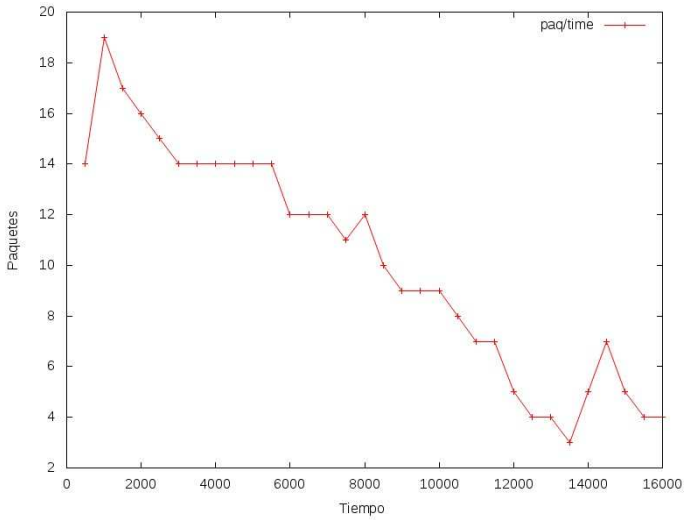

Figura 8 - Paq/Time

- paq_time_tipos.jpg

  En la "Figura 9" se muestran los paquetes que se generan de cada tipo por intervalo de tiempo. Cada tipo de paquete es una línea en el gráfico hasta un máximo de 20 tipos.

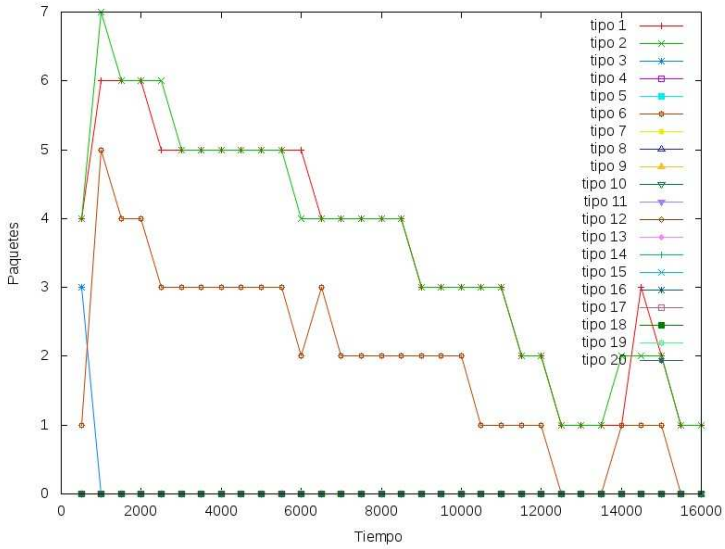

Figura 9 - Paq/Time tipos

- paqses_time.jpg

  En la "Figura 10" se muestran los paquetes por sesión totales que se generan por intervalo de tiempo.

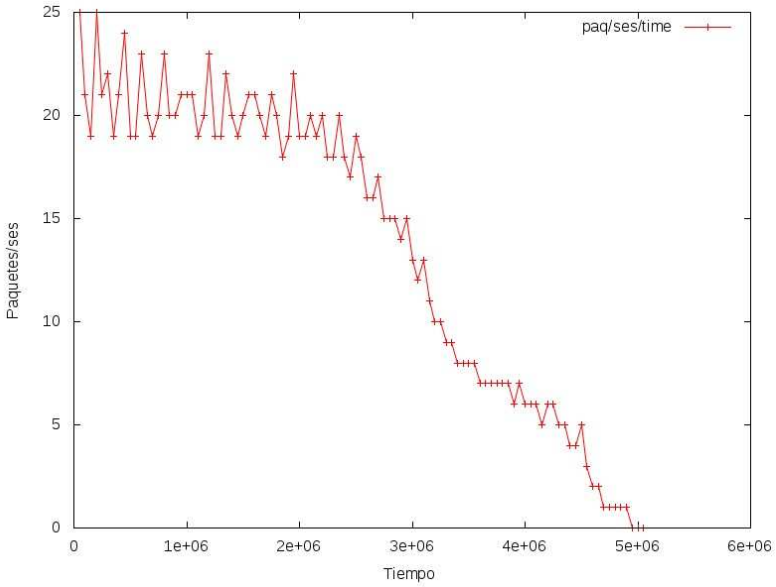

Figura 10 - Paq/Ses/Time

- paqses_time_tipos.jpg

  En la "Figura 11" se muestran los paquetes por sesión de cada tipo que se generan por intervalo de tiempo. Cada tipo de paquete es representado por una línea hasta un máximo de 20 tipos de paquetes.

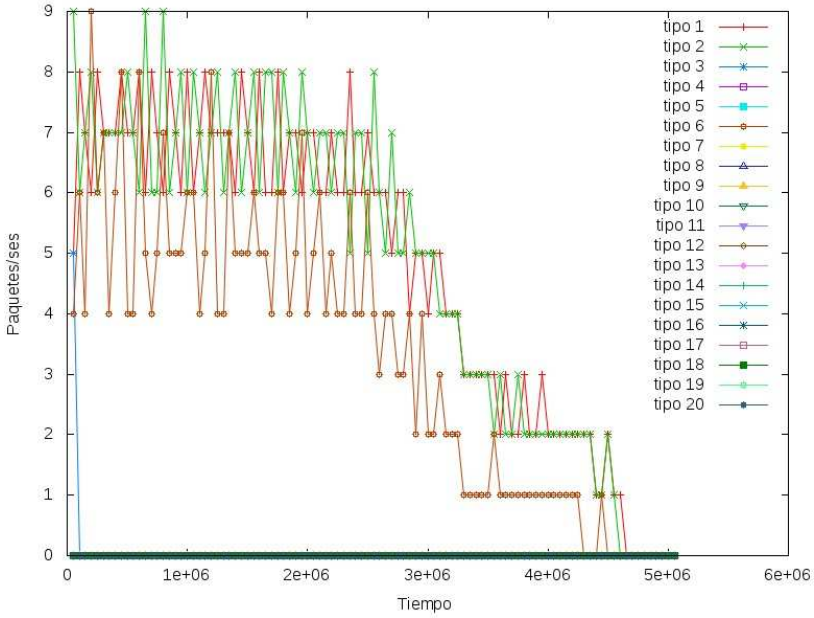

Figura 11 - Paq/Ses/Time tipos

- errorEstrategias.jpg

  En la "Figura 12" se muestran los puntos en el tiempo (eje *x*) en que se reciben estrategias de asignación de ancho de banda en un nodo *host*. En el eje *y* se expresa el error respecto de esa estrategia indicada y el valor real final de la red estabilizada.

  Los puntos en esta gráfica tienden a la línea del 0. Un protocolo es bueno si sus puntos tienden rápidamente a 0 y no da valores muy desviados del valor final.

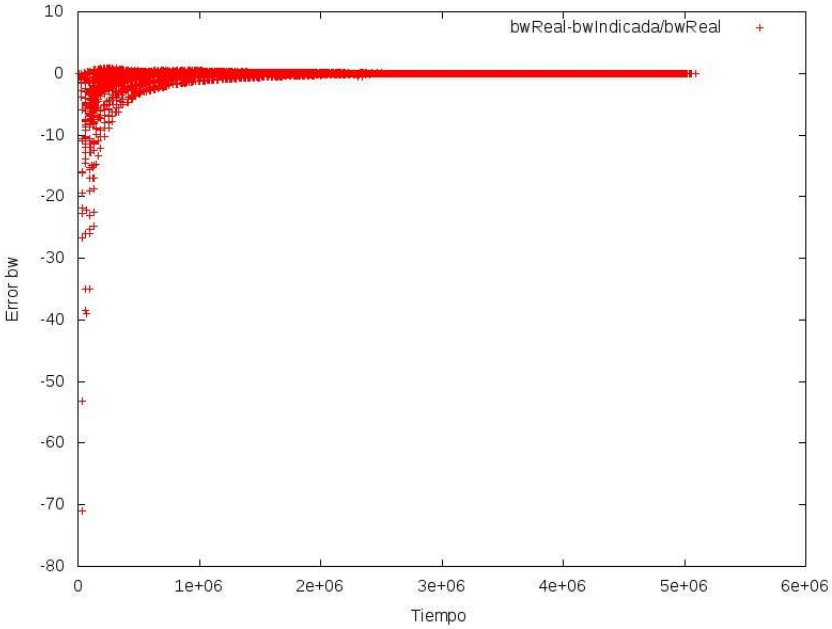

Figura 12 - Error estrategias

El entorno de simulación está preparado para que se puedan generar estadísticas propias para cada protocolo. Para ello se llama a ciertos procedimientos reimplementables cada "cicloPasos" y al final de la ejecución por si en el protocolo se quiere ejecutar algún tratamiento o recopilación de datos.

CAPÍTULO 3

DISEÑO E IMPLEMENTACIÓN DEL PROTOCOLO FDBU

En esta sección se presenta el protocolo FDBU (Fair Distribution by Updating), en español "Reparto Equitativo Mediante Actualizaciones", un protocolo distribuido que permite asignar un *max-min fair rate* en la red.

3.1 Descripción

El protocolo FDBU, desarrollado para este proyecto, trata de mantener las sesiones con el ancho de banda justo sin generar ningún paquete de control mientras no haya cambios en la red. Esto permite que en redes con pocos cambios la distribución sea óptima, únicamente generando paquetes para el reajuste cuando es necesario por algún cambio en la red (apertura de sesión, cierre de sesión, cambio de petición).

El protocolo está implementado para permitir más de una sesión por *host*. Esto quiere decir que los enlaces de los *hosts* también hacen reparto de ancho de banda.

Para ello el protocolo hace uso de los siguientes tipos de paquetes:

Join - Paquete que se envía al arrancar la sesión para notificar a todos los routers del camino su existencia. Cada router crea la sesión indicada cuando le llega, y reenvía el paquete hacia el destino. El *host* destino ignora el paquete.

Leave - Paquete que se envía al cierre la sesión para notificar a todos los routers del camino su finalización. Cada router elimina la sesión de su lista de sesiones cuando le llega, y reenvía el paquete. El *host* destino acaba con el paquete (ignorándolo).

RateRequest - Paquete que envía el *host* para preguntar qué ancho de banda se le puede asignar. Cada router compara la limitación indicada en el paquete con su "mmfr" (ver apartado 3.2 "Pseudocódigo"), asignando el mínimo, y reenvía el paquete. El destino crea un SetRate para ser enviado a la fuente indicándole el ancho de banda mínimo entre todos los routers.

SetRate - Paquete que envía el destino al recibir un RateRequest contestando con el ancho de banda que se asigna a la sesión actual. Cada router y el *host* destino comprueban que el paquete sigue siendo válido (no obsoleto). Guarda el valor final asignado a la sesión en cuestión, y comprueba que ninguna otra sesión necesite generar un "UpdateRate" para recalcular su SetRate por ser obsoleto su valor actual. Si el receptor es un router, reenvía el paquete por su camino hasta el *host* que acaba con él. Si ninguna sesión del router/host en cuestión tiene actualizaciones en proceso, se generan los paquetes UpdateRate pendientes.

UpdateRate - Paquete enviado hacia el host origen cuando es necesario recalcular/recomprobar el ancho de banda asignado a una sesión. Los routers lo reenvían si la sesión sigue activa. El host origen lo procesa si no tiene ninguna actualización en progreso. Si la hay, lo bloquea hasta que se resuelva la actualización (RateRequest-SetRate) en progreso. Sólo guarda el último UpdateRate bloqueado para procesarlo cuando pueda. Cuando el origen lo procesa crea un RateRequest nuevo para comprobar una vez más su ancho de banda asignado.

Ejemplo de inicio de sesión sencillo:

Trabajamos sobre la red "básica" con 3 nodos host por router *stub*, y las siguientes velocidades de ancho de banda en los enlaces, quedando la configuración:

topologia=basic

numhost=3

velHost=110

velStub=160

velTransit=215

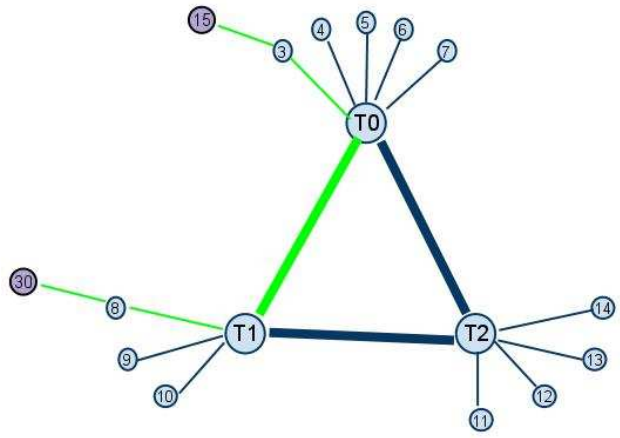

Figura 13 - FDBU ejemplo 1

Iniciamos una sesión entre el host número 15 y el host número 30.

Acción: *"1 o 15 30"*

Tenemos velocidades finales asignadas:

Fin: Host 15-30 ==>110.0

Los paquetes que se han generado/movido en la red han sido:

Join - generado al iniciar la sesión (en primer lugar).

15 --> 3 --> T0 --> T1 --> 8 --> 30

RateRequest - generado al iniciar la sesión (en segundo lugar).

15 --> 3 --> T0 --> T1 --> 8 --> 30

SetRate - generado como respuesta al RateRequest.

30 --> 8 --> T1 --> T0 --> 3 --> 15

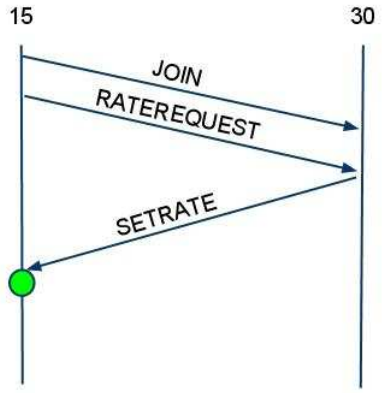

Figura 14 - FDBU ejemplo 1

Ejemplo de generación de paquete "*Update*":

Trabajando sobre el escenario anterior, con la sesión 15-30 ya estabilizada, insertamos la sesión 16-50.

Acciones: *"1 o 15 30" y "100 o 16 50*"

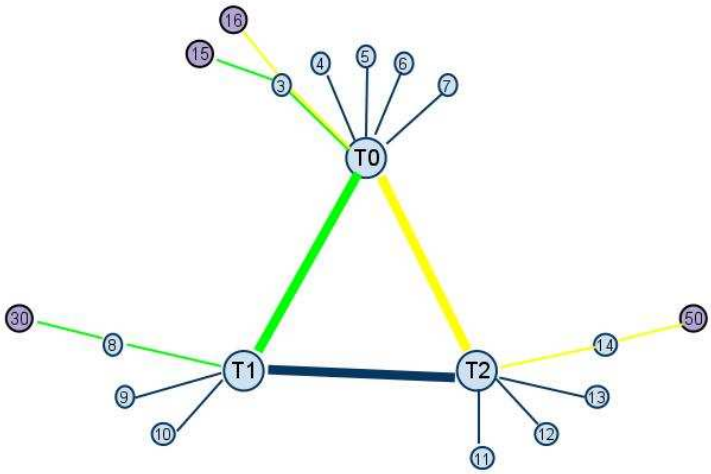

Figura 15 - FDBU ejemplo 2

En el diagrama se ve la red básica con el camino de la sesión 15-30 en verde y el de la sesión 16-50 en amarillo.

Tendremos velocidades finales asignadas:

Fin: Host 15-30 ==>80.0

Fin: Host 16-50 ==>80.0

Al igual que en el escenario anterior con la entrada de la sesión 15-30, con la llegada de la sesión 16-60 se generarán un Join y un RateRequest. El rateRequest será contestado con un SetRate, pero este SetRate al pasar por el router 3, y comprobar su enlace de salida (en sentido de la sesión) se dará cuenta de que es necesario generar un UpdateRate para la sesión 15-30 (anteriormente estabilizada).
Este UpdateRate será enviado al Host 15 quién generará de nuevo un "RateRequest" que tendrá su "SetRate" de respuesta.

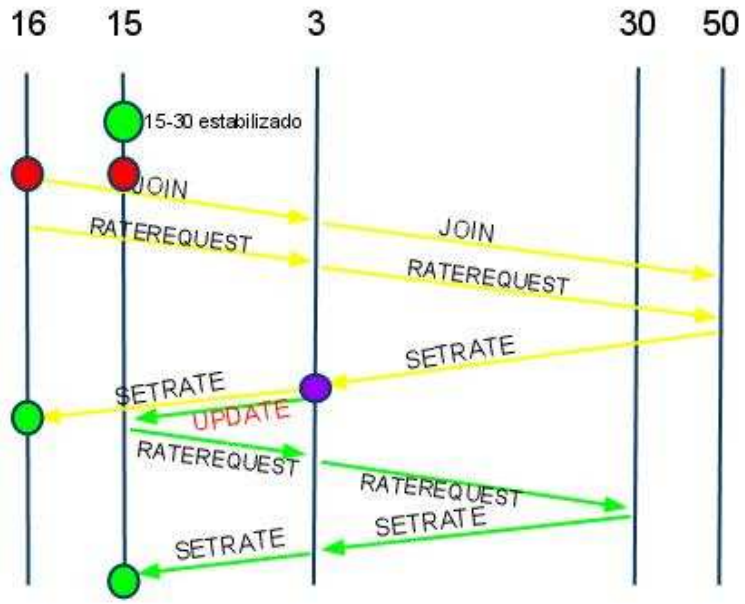

Figura 16 - FDBU ejemplo 2

En el diagrama de la "Figura 16" los puntos verdes indican una sesión estabilizada, los rojos una sesión desestabilizada y el morado la generación de un nuevo "UpdateRate". Las líneas en amarillo corresponden a paquetes referentes a la sesión 16-60 y las líneas verdes a los de la sesión 15-30. Por simplificarlo sólo se han dibujado los 4 host implicados y el nodo router que genera el "UpdateRate".

Como se puede apreciar en el diagrama, el "SetRate" de la sesión 16-50 genera un "UpdateRate" para la sesión 15-30 (en el punto morado).

3.2 Pseudocódigo

//Notation

$\lambda_m(s) = \max_{s \in S}\{s.bw\}$.Es el máximo ancho de banda asignado a una sesión en el enlace "e".

$\lambda_t(S) = \sum_{s \in S}\{s.bw\}$. Es el total de ancho de banda consumido por las sesiones del enlace e.

s - Representará a la sesión del conjunto "S" que se está tratando

//Atributos de Edge

Be - es el ancho de banda del enlace e.

S - es el conjunto de sesiones que atraviesa el enlace e.

//Atributos de Sesión "s"

bw - es el ancho de banda asignada a la sesión s.

η - código de actualización de la sesión s.

waitClose - boolean que indica si está en proceso de cierre la sesión s.

srcSes - origen de la sesión.

dstSes -destino de la sesión.

//Atributos del paquete "p"

bw - ancho de banda indicado por el paquete p.

η - código de actualización del paquete p.

s - identificador de la sesión en cuestión

src - origen del paquete

dst - destino del paquete

//Interfaz origen

```
procedure Join(s){
      send downstream Join(s)
      send downstream RateRequest(s,s.bw)
}
```

```
procedure Leave(s){
      send downstream RateRequest(s,0)
      s.waitClose ← true
      wait for SetRate(s,0)
      send downstream Leave(s)
}

procedure Change(s,bw){
      send downstream RateRequest(s,bw)
}

//Interfaz edge (núcleo del algoritmo)
function mmfr(S, be){
      var
            S': {set of sessions}
      begin
            S' ← φ
            while ∃ s ∈ S : s.bw < (be−λt(S'))/(|S|+1) do
                  S ← S − s
                  S' ← S' ∪ {s}
            end while
            return (be−λt(S'))/(|S|+1)
      end

when received Join(p) do            //s no puede pertenecer a S
      if is p.dst then drop paquet endif
      S ← S ∪ {s}
      s.bw ← 0
      send downstream Join(p)
end when

when received Leave(s, dst) do            //s debe perteneciar a S
      if is p.dst then drop paquet endif
      S ← S − {s}
      send downstream Leave (s)
end when
```

```
when received RateRequest (p) do
      if is p.dst then send upstream SetRate (s, p.bw, p.η)
      else
            if p.bw=0 then s.waitClose ←true
            else if s.waitClose then drop paquet endif
            s.η ← p.η
            p.bw ← min (p.bw, mmfr(S{s}, be))
            send downstream RateRequest(p)
      endif
end when

when received SetRate(p) do
      if s.waitClose and p.bw ≠ 0 then drop paquet endif
      if s.η ≠ p.η then drop paquet endif
      if p.bw ≤ mmfr(S − {s}, be) then
            if s.bw ≠ p.bw then
                  bwAnt ← s.bw
                  s.bw ← p.bw
                  for each s' ∈ (S − {s}) do
                        if s.η = ∅  or bwAux > p.bw or
                        (bwAux < p.bw and s'.bw > mmfr(S − {s'}, be))
                        then
                              s.pendingUpdate ← true
                  end for
            end if
      else
            p.bw ← ∞
      endif
      if !srcSes then send forward SetRate ( s, p.bw, p.η) endif
      s.η ← ∅
      if ∀ s' ∈ S : s.η = null then
            for each s' ∈ S do if(s'.actualizacionPendiente)
                  s.pendingUpdate ← false
                  send upstream UpdateRate ( s' )
      endif
end when
```

```
when received UpdateRate(p) do
      if s.waitClose then drop packet endif
      if p.dst then
            block updateRate until s.η = ∅
            //Sólo se mantiene el último updateRate bloqueado por sesión
            send downstreamRateRequest(s, ∞)
      else
            send upstream UpdateRate(p)
      endif
end when
end.
```

3.3 Implementación

A continuación se hará una breve descripción de la implementación de este protocolo viendo por encima el funcionamiento y uso de cada clase:

fdbu.Edge

Clase que representa un enlace. Esta contiene el código general del protocolo. Basada en el pseudocódigo expresado en el apartado 3.2 "Pseudocódigo".

El método "procesaPk" despacha los paquetes que le van llegando enviándolos al método correspondiente según el paquete: pLeave, pRateRequest, pSetRate, pUpdateRate o pJoin.

Los procedimientos "addSesion" y "closeSesion" son sólo usados por los enlaces salientes de los Host para iniciar y cerrar sesiones.

El código fuente de esta clase puede verse en el "ANEXO A".

fdbu.NetworkFichero

Clase obligatoria para todo protocolo que hereda de "comunes.NetworkFicheroComun" e indica qué tipo de nodos usar para los *hosts* y los *routers,* y la clase que implementa los enlaces.

fdbu.Paquete

Clase que hereda de "comunes.Packet" añadiendo a los parámetros básicos de un paquete (sesión, origen, destino, tipo y tamaño) el ancho de banda y el código de actualización.

fdbu.PaqueteTipo

Esta clase contiene los cinco tipos de objetos "PaqueteTipo" que pueden enviarse con este protocolo que son definidos por un número:

```
0: SETRATE
1: RATEREQUEST
2: JOIN
3: LEAVE
4: UPDATERATE
```

fdbu.Sesion

Esta clase hereda de "comunes.SesionComun", y existe un objeto identificando cada sesión. Contiene todos los parámetros que es necesario guardar de la sesión como pueden ser su velocidad asignada actual, variables *booleanas* para el control de actualizaciones y estados, y métodos para generar estadísticas de las estrategias recibidas.

fdbu.Simulator

Clase que extiende a "comunes.Sim" indicándole el nombre del protocolo concreto; en este caso "fdbu".

fdbu.TransportControl

Esta clase es obligatoria y hereda de "comunes.TransportControlComun". Implementa los métodos abstractos de comprobación de estabilización y de generación de estadísticas propias, y contiene el método encargado de mostrar las velocidades finales de la red para su almacenamiento y posterior comparación con el algoritmo centralizado.

El código contiene trazas de error, que en condiciones normales no deberían aparecer, para que la resolución de problemas sea más sencilla. Además, si el modo "debug" está activado, se muestran estadísticas cada cierto número de pasos (según configuración de la variable "cicloPasos") con el formato:

```
(87:3)(si:no) Red NO estabilizada. #s NO Stb: 3 #s Stb: 87 #%NoStb
3.33 avgDiff: 0.01 avgAbsDiff: 0.01
```

```
(90:0) Red SI estabilizada: 90 sesiones
```

En estas trazas se comprueba la estabilización de la red mostrando el número de sesiones con el ancho de banda igualado al del resultado del algoritmo centralizado. Muestra también el tanto por ciento de sesiones no estabilizadas y la diferencia de asignación de ancho de banda media tanto con signo como en valor absoluto.

Si la red está estabilizada y en la configuración se fijó como "true" la variable "pararSiEstabilizado", la ejecución de la simulación parará en ese instante.

3.4 Pruebas de funcionamiento

Se ha realizado una batería de simulaciones para comprobar el rendimiento y correcto funcionamiento del protocolo. Todas las simulaciones ejecutadas ofrecen resultados correctos comparándolas con el algoritmo centralizado, no dando ningún error considerable de asignación de ancho de banda. Se considera que el funcionamiento del algoritmo es correcto con un tiempo y número de paquetes generados aceptable, pero mejorable a la hora de evitar ciertas actualizaciones redundantes.

Las simulaciones han sido lanzadas con las topologías "small", "medium" y "big", las 3 con configuraciones "Wan" y "Lan". Se lanzan 3 ejecuciones con cada red y configuración de 10, 30, 100, 300, 1000, 3000 y 10000 sesiones, y se recopilan los datos en una hoja de cálculo Excel. En el fichero se calculan las medias de tiempo y paquetes, además de valores como los "paquetes/sesión". Las simulaciones para "LAN – Small" con 3.000 y 10.000 sesiones no se incluyen por su lenta resolución.

En las tablas de la 2 a la 7 se muestran los resultados obtenidos en todas las simulaciones para cada topología y configuración.

Tabla 2 (LAN - Small)

SES	TIEMPO 1	PAQUETES 1	TIEMPO 2	PAQUETES 2	TIEMPO 3	PAQUETES 3
10	**961**	**285**	**709**	**183**	**1.143**	**253**
30	**1.711**	**1.337**	**1.466**	**1.480**	**1.592**	**1.280**
100	**2.854**	**11.918**	**4.305**	**12.443**	**3.419**	**12.540**
300	**16.498**	**170.972**	**12.631**	**132.969**	**9.507**	**120.096**
1000	**294.694**	**5.331.803**	**435.031**	**8.798.860**	**351.776**	**7.576.356**

Tabla 3 (LAN - Medium)

SES	TIEMPO 1	PAQUETES 1	TIEMPO 2	PAQUETES 2	TIEMPO 3	PAQUETES 3
10	887	150	714	147	861	150
30	1.066	524	882	469	937	476
100	1.205	1.878	1.193	1.765	1.368	2.031
300	1.412	6.909	1.492	6.732	1.407	7.061
1000	2.504	45.762	2.421	44.019	2.190	45.150
3000	5.277	436.100	5.623	441.770	4.739	408.425
10000	47.119	16.009.790	37.416	14.164.609	44.135	15.269.408

Tabla 4 (LAN - Big)

SES	TIEMPO 1	PAQUETES 1	TIEMPO 2	PAQUETES 2	TIEMPO 3	PAQUETES 3
10	847	165	899	165	885	156
30	910	486	866	477	893	489
100	1.186	1.657	923	1.652	1.076	1.673
300	1.372	5.426	1.360	5.453	1.354	5.395
1000	2.038	22.309	2.007	20.919	1.586	20.810
3000	2.597	92.792	2.754	96.602	2.247	96.895
10000	5.832	914.136	4.962	857.956	5.566	890.431

Tabla 5 (WAN - Small)

SES	TIEMPO 1	PAQUETES 1	TIEMPO 2	PAQUETES 2	TIEMPO 3	PAQUETES 3
10	287.523	283	172.497	307	193.000	299
30	806.833	1.167	339.624	1.842	316.460	1.399
100	1.420.033	21.016	1.583.101	16.209	287.665	18.470
300	4.631.569	119.541	5.323.713	158.258	2.790.387	140.293
1000	13.323.759	1.432.589	9.681.242	1.193.495	5.095.077	1.444.901
3000	6.724.446	8.625.908	40.254.315	12.561.581	29.513.680	8.959.483
10000	86.378.254	60.028.170	62.514.495	55.517.776	29.555.455	55.149.392

Tabla 6 (WAN - Medium)

SES	TIEMPO 1	PAQUETES 1	TIEMPO 2	PAQUETES 2	TIEMPO 3	PAQUETES 3
10	30.352	150	27.378	150	74.645	150
30	250.494	522	152.718	480	99.886	489
100	148.081	1.813	157.100	1.985	204.203	1.966
300	438.141	8.061	116.740	7.695	276.889	7.861
1000	165.922	54.640	1.046.486	51.123	305.451	53.065
3000	3.243.689	526.566	1.458.781	517.279	1.318.924	490.277
10000	9.911.634	11.922.559	5.064.871	13.521.884	9.060.985	11.554.600

Tabla 7 (WAN - Big)

SES	TIEMPO 1	PAQUETES 1	TIEMPO 2	PAQUETES 2	TIEMPO 3	PAQUETES 3
10	21.271	168	35.946	159	65.282	153
30	15.555	498	45.555	468	31.648	480
100	105.015	1.704	172.007	1.671	305.621	1.672
300	82.659	5.281	588.664	5.510	484.476	5.415
1000	601.518	24.008	81.230	22.846	664.423	24.143
3000	487.553	104.913	273.266	109.406	667.700	107.921
10000	2.482.691	993.013	2.302.834	1.061.265	1.010.040	1.018.953

A continuación se muestran unas gráficas que expresan distintos aspectos del rendimiento del protocolo dependiendo del número de sesiones que se inserten en la red, del tipo de topología usada y su configuración "Lan" o "Wan".

Paquetes

En la "Figura 17" se muestran los paquetes que se generan para distintos números de sesiones en distintas topologías ("Small", "Medium" y "Big") tanto para configuraciones "LAN" como "WAN". Como se aprecia a simple vista, el número de paquetes generados será menor cuanto más grande sea la red dado que su congestión y número de interacciones entre sesiones será menor.

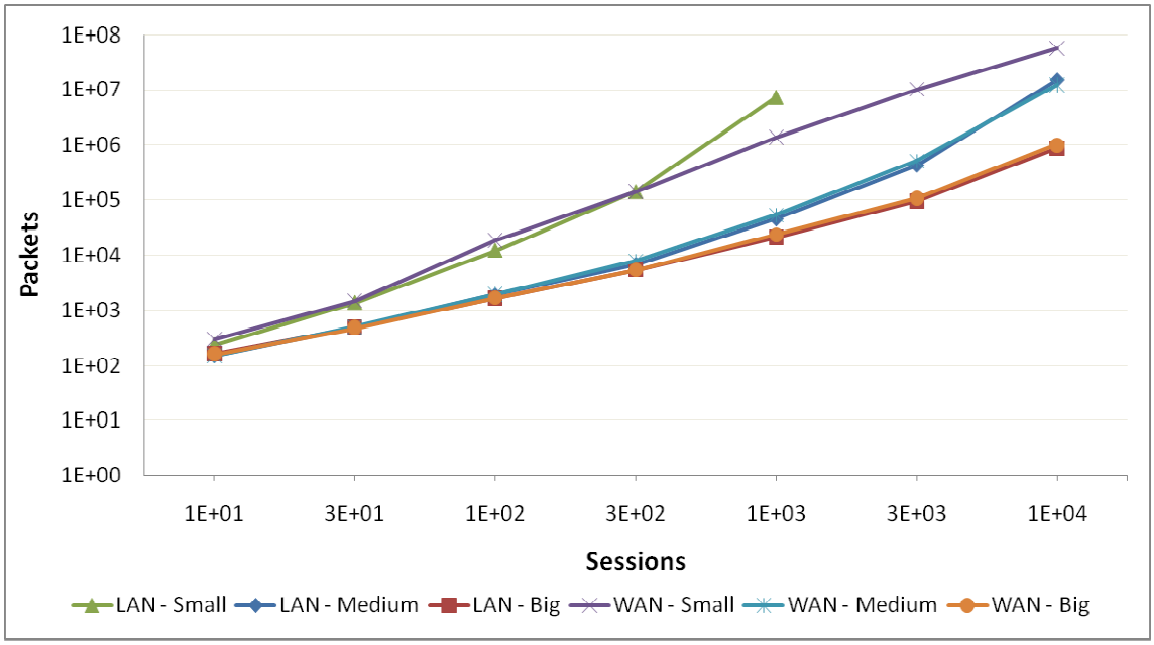

Figura 17 - FDBU Paq/Ses

Paquetes/Sesión

En la "Figura 18" se aprecia que el número de paquetes generados por cada sesión va aumentando rápidamente con el aumento de las sesiones. Esto quiere decir que cuantas más sesiones contenga la red, más paquetes de control serán necesarios para lograr la estabilidad de cada sesión.

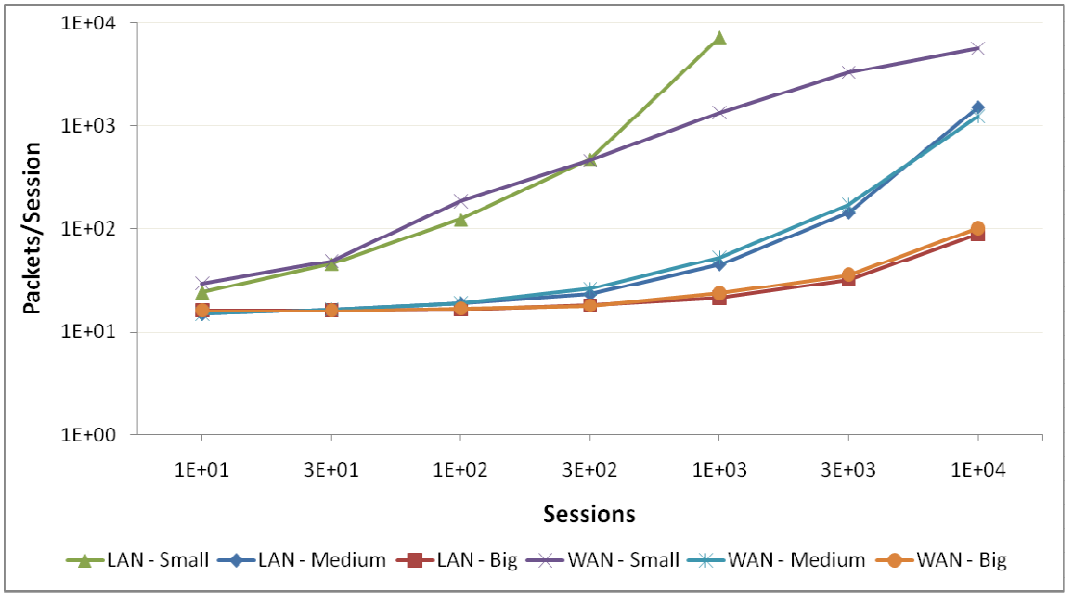

Figura 18 - FDBU Paq/Ses

Tiempo

La "Figura 19" contiene la gráfica de tiempos para todas las simulaciones ejecutadas. Se puede observar que los tiempos de resolución para configuraciones "WAN" son mucho más altos dado que sus tiempos de transmisión también son más altos. Pero en topologías como la "Small" con configuración "LAN" se dispara el tiempo con alto número de sesiones por congestión en las actualizaciones.

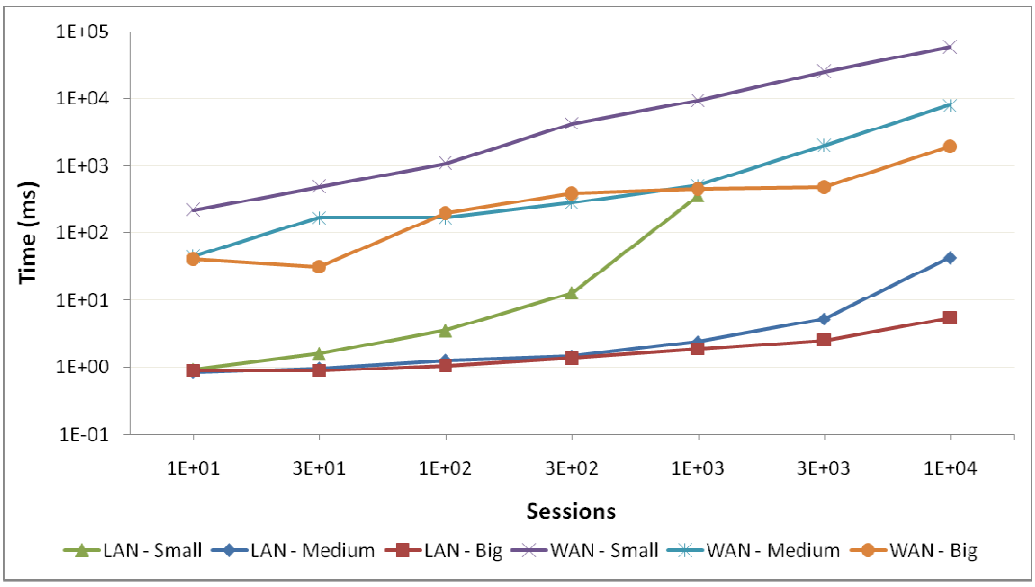

Figura 19 - FDBU Time/Ses

Es importante anotar que el tiempo total, mostrado en estas gráficas, indica el momento en el que se dejan de generar paquetes de control si no hay entradas/salidas de sesiones, y no el tiempo de estabilización que puede ser mucho menor al ajustarse los anchos de banda equitativos correctos. Pese a que ese tiempo total sea alto los valores que los hosts tienen asignados para cada sesión son altamente precisos o incluso exactos poco tiempo después de iniciar las sesiones.

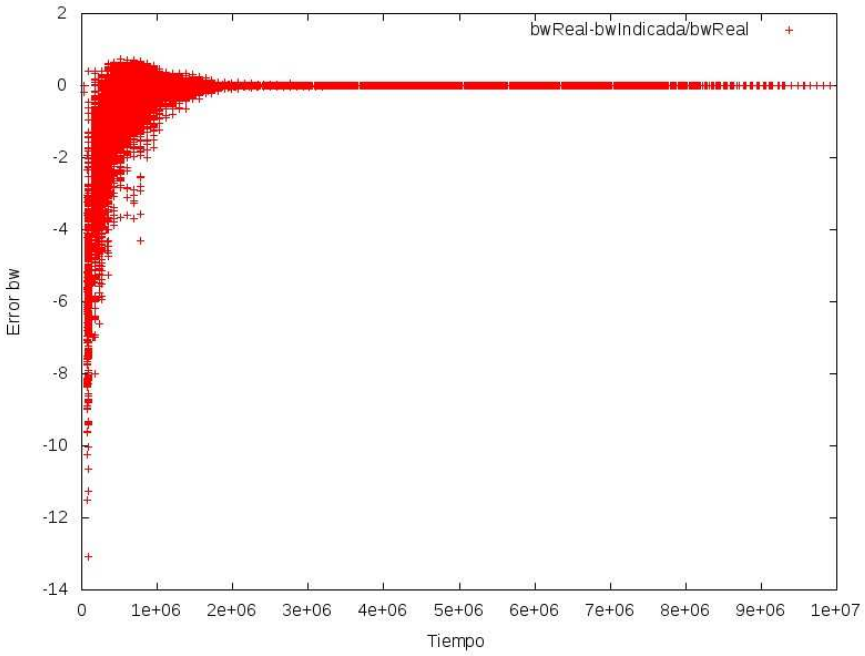

Figura 20 - FDBU Error estrategias

En la "Figura 20" se muestra una gráfica de errores en las estrategias dados en una simulación de 10K sesiones sobre la red "Medium" con configuración "Wan". Como se puede apreciar, solo en los primeros instantes se reciben errores altos de asignación de ancho de banda. Estas asignaciones erróneas son además en su mayoría a la baja, lo que hace que no se colapsen los nodos. Poco después del comienzo de la simulación se corrigen esos errores por la recepción de asignaciones correctas de ancho de banda, con error "0".

En la Figura 21 se muestra una gráfica con entradas y salidas de paquetes para comprobar el funcionamiento del protocolo cuando una red está estabilizada. En un primer momento se

introducen 1.000 sesiones. Tras cuatro milisegundos (estando ya la red estabilizada) se cierran 100 sesiones. Cuatro milisegundos más tarde se abren 100 sesiones nuevas. Y en una etapa final (otros cuatro milisegundos más tarde) se hace una entrada de 100 sesiones y salida combinada de otras 100 sesiones. Solo se cierran sesiones de las anteriormente estabilizadas y nunca de las recién abiertas.

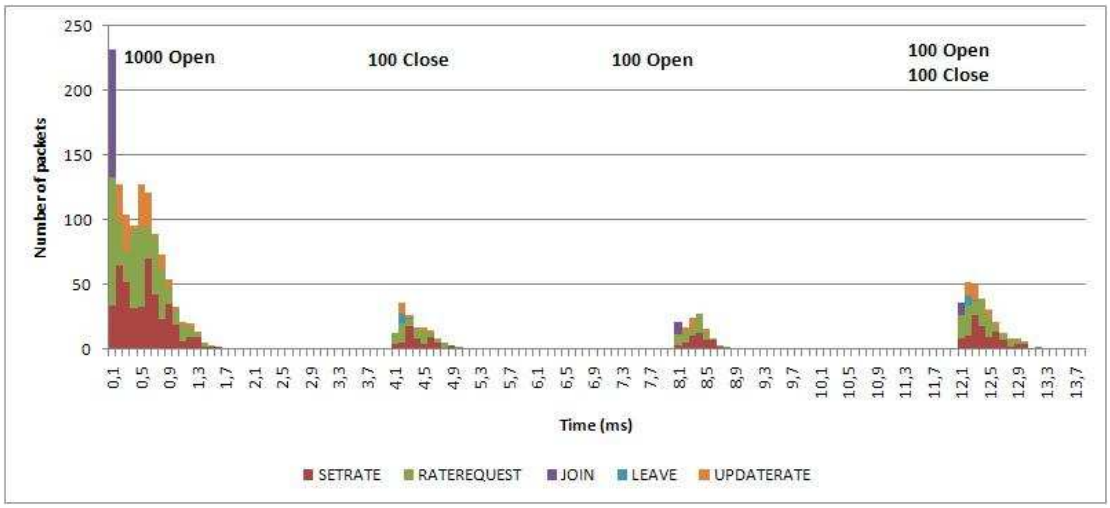

Figura 21 - FDBU E/S

Se puede apreciar que el tiempo de estabilización de la red es más o menos igual entren, salgan o lo hagan simultáneamente. Este tiempo depende más del colapso interno de las sesiones actuales de la red.

CAPÍTULO 4

MANUAL DE USUARIO

Con este pequeño manual de usuario se pretende explicar cómo instalar, configurar y ejecutar una simulación en el "Simulator protocols for bandwidth allocation" (SPBA, Simulador de protocolos de asignación de ancho de banda). Además, se muestra de forma resumida cómo se podría agregar un protocolo nuevo, dando las nociones necesarias para su implementación, ejecución y comprobación de sus resultados con el algoritmo centralizado o comparar su rendimiento con otros protocolos.

4.1 Instalar Simulador

Para la ejecución de simulaciones con el simulador SPBA se suministra un paquete "*fat.jar*" que contiene todos los ficheros ".*class*" necesarios para la ejecución del simulador. Este fichero está en "*Ejecuciones/lib/SPBA_fat.jar*". Además del código contenido en el fichero "*jar*", se requieren también las carpetas de configuración y topologías. Todo ello se coloca en el directorio deseado

dentro de la carpeta de "Ejecuciones" quedando una estructura como la indicada en el apartado 2.12 "Directorio de ejecución".

Una vez copiada la carpeta de "*Ejecuciones*" en el directorio deseado se deben hacer unos cuantos ajustes esenciales en la configuración. En los ficheros de configuración de la simulación concreta deberá indicarse el nuevo directorio en el que está la carpeta de "*Ejecuciones*". Además, para que las gráficas se generen automáticamente con la herramienta *gnuplot* es necesario cambiar las rutas de la carpeta "*tmp*" citadas en el fichero "*tmp/generarGraficosUnicos*".

Tras hacer estas modificaciones, ya se puede proceder a ejecutar una simulación con el protocolo de ejemplo incluido en el paquete.

4.2 Configuración y ejecución

Para realizar una simulación particular, se debe modificar/crear un fichero de configuración con las características y nombre deseado en el directorio "*Ejecuciones/config/*". La manera de configurar ese fichero se explica en el apartado 2.10 "Configuración"; y no debe olvidarse de indicar la ruta elegida para la carpeta "Ejecuciones".

Una de las informaciones indicada en el fichero de simulación es la topología que se usará, que debe estar en el directorio "*Ejecuciones/topologias/*".

Para ejecutar una simulación usando el ".jar" bastará con ejecutar el comando:

```
java -cp <fichero_jar> comunes.Ejecutar <fichero_configuración>
<protocolo1> <protocolo2> … <protocoloN>
```

De esta manera se usará el contenido del fichero "jar" como *classpath,* ejecutando la clase "Ejecutar" del paquete "comunes", y usando el fichero de configuración indicado para ejecutar todos los protocolos desde <*protocolo1*> hasta <*protocoloN*>.

Un ejemplo de ejecución puede ser:

```
java -Xmx9000M -cp ./SPBA_fat.jar comunes.Ejecutar
/home/usuario/workspace/SPBA/Ejecuciones/config/configA.txt fdbu
```

En este ejemplo se estaría configurando la máquina virtual *java* para que reserve un tamaño mínimo de *heap* de 9000 Megabytes mediante el parámetro "-Xmx9000M". Se estaría usando de variable *classpath* el fichero "*SPBA_jat.jar*" del directorio actual para ejecutar la clase "Ejecutar" del paquete "comunes", aplicando la configuración del fichero indicado como parámetro para ejecutar únicamente el protocolo "fdbu".

4.3 Resultados

En la ruta "*Ejecuciones/results*" se creará una carpeta con el nombre indicado en el parámetro de configuración "codEje" o con un nombre aleatorio si este parámetro es "0".
En esta carpeta estarán todos los datos obtenidos de la ejecución del protocolo. Así, la carpeta contendrá un *log* general de la ejecución, un *log* concreto de la simulación de cada protocolo, un fichero con los resultados del algoritmo centralizado, un fichero con los resultados de cada protocolo, un fichero con las acciones generadas y ejecutadas para la simulación, y una carpeta con estadísticas de distintos tipos y sus gráficas generadas automáticamente.

Un ejemplo del *log* general puede ser:

```
Comenzando ronda de simulaciones con código: PRUEBAS
Generando acciones
Ejecutando algoritmo centralizado (24 Jan 2011 09:45:45 GMT)
Fin de ejecución de algoritmo centralizado (24 Jan 2011 09:45:45 GMT)
Tiempo de ejecución del algoritmo=934ms
Buscando protocolos indicadas en los parámetros
Evaluando protocolo fdbu
Ejecutando simulación PRUEBAS del protocolo fdbu (24 Jan 2011 09:45:46
GMT)
Fin de ejecución de la simulación del protocolo fdbu (24 Jan 2011
09:48:26 GMT)
Tiempo de ejecución del algoritmo=160260ms
Resultados correctos para protocolo fdbu
Ejecutando comando: gnuplot
/home/usuario/workspace/SPBA/Ejecuciones/tmp/generarGraficosUnicos
FIN de evaluación del protocolo fdbu con código: PRUEBAS
FIN de ejecución con código: PRUEBAS
```

4.4 Añadir un protocolo (con *src*)

Entre el material incluido con el entorno SPBA se encuentra el código fuente bajo un directorio llamado "src". Para insertar un nuevo protocolo, se puede crear un nuevo paquete dentro del "src" dado. Este "src" tendrá unas clases mínimas obligatorias y todas las clases extra que se necesiten para su implementación con las siguientes restricciones:

Simulator.java – Esta clase hereda de "comunes.Sim" pero indica el nombre del protocolo concreto para la simulación.

NetworkFichero.java - Hereda de la clase abstracta "comunes.NetworkFicheroComun" e implementa como mínimo los métodos abstractos para indicar al protocolo qué clases usar para crear los *edges* los *routers* y los *hosts*.

TransportControl.java – Clase que hereda de "comunes.TransportControlComun" realizando labores como la extracción a un fichero con el formato adecuado de los resultados de ancho de banda asignados por el protocolo al final de la ejecución. Además permite realizar estadísticas y captura de valores concretos para cada protocolo como pueden ser los ciclos necesarios para estabilizarse, el porcentaje de paquetes irrelevantes o repetitivos, etc.

Los Routers serán objetos de una clase que herede de "comunes.NodoRouter" o directamente de esa misma clase si no es necesario agregarle nada. Al igual pasa con los hosts y la clase "comunes.NodoHost".

Los objetos que representan a los enlaces tendrán que heredar todos de la clase "comunes.EdgeComun", pudiendo ser de distintos tipos de clases. Esta clase será la que contenga el código principal del algoritmo de asignación.

Los paquetes usados deben heredar del paquete básico "comunes.Packet".
Los tipos de paquetes serán indicados en una clase que herede de *PacketType*.

Para representar a las sesiones se recomienda usar una clase que herede de "comunes.SesionComun" ya que ésta ofrece de antemano la representación de la sesión mediante el origen, el destino y un contador extra para diferenciar sesiones repetitivas de un mismo origen a un mismo destino.

El uso de un nuevo protocolo requiere, por supuesto, un fichero de configuración en el directorio "Ejecuciones/config/" que indique adecuadamente los nombres de las clases para la creación de la red y para el protocolo de transporte.

4.5 Añadir topología

Para usar una topología distinta de las que se ofrecen en el simulador, se puede usar la herramienta GT-ITM o se puede crear una topología con el formato adecuado a mano. El fichero de rutas será generado en cualquier caso automáticamente al ejecutar un protocolo concreto.

Para generar redes usando el programa de modelado GT-ITM se sigue este procedimiento:

-Primeramente se debe descargar y compilar el código del programa. Está preparado para SunOs 4.x y con ciertas modificaciones en el *Makefile* también para *Solaris*. Y hay otra versión disponible para *Linux*.

-Tras esto preparamos ficheros de configuración de las redes que queremos crear. Estos ficheros tienen un formato como el siguiente:

```
# <method keyword> <number of graphs> [<initial seed>]
# <# stubs/trans node> <#rand. t-s edges> <#rand. s-s edges>
# <n> <scale> <edgemethod> <alpha> [<beta>] [<gamma>]
# number of nodes = 1x4x(1+3x8) = 100
```

-Se ejecuta el programa "edriver", que genera un fichero ".gb" que contiene la definición de una red con las características del fichero de configuración indicado:

```
/bin/edriver ficheroConfiguracion -nd -hh -ll -hl
```

-Se ejecuta el comando "sgb2alt" para transformar la definición en formato ".gb" a una definición de la red más sencilla que se guardará en un fichero ".alt"

```
bin/sgb2alt redIndicada.gb redIndicada.alt
```

-Los ficheros .alt contienen 3 zonas.

La primera define las cantidades y propiedades generales de la red:

```
GRAPH (#nodes #edges id uu vv ww xx yy zz)
```

En una segunda zona se van indicando todos y cada uno de los vértices (diferenciando nodos *stub* de nodos *transit*):

```
VERTICES (index name u v w x y z)
```

En la última zona se indican los enlaces entre nodos y sus características.

```
EDGES (from-node to-node length a b)
```

Este programa sólo puede usarse en procesadores de 32 bits. No está preparado para procesadores de 64 bits. Esto dio algunos problemas durante la realización del proyecto como se indica en la sección "Historia del proyecto".

HISTORIA DEL PROYECTO

El proyecto comenzó por las necesidades de tener un entorno de simulación para probar los protocolos del proyecto de investigación para el que trabajo. Se comenzó usando el *PeerSim* y se fue adaptando para orientarlo a protocolos de asignación de ancho de banda. Además, se modificó para automatizar y facilitar el lanzamiento y configuración de las simulaciones. Se le agregó la posibilidad de usar temporizadores y se modificó el envío de paquetes para hacer los tiempos de transmisión y propagación más reales. Se pusieron colas para evitar el adelantamiento de paquetes garantizando que llegaran en orden. Se automatizó la carga de la red creada por GT-ITM y la creación de la tabla de rutas centralizada.

Poco a poco se fueron adaptando y mejorando partes del simulador para que la codificación de un protocolo fuera lo más sencilla posible mediante el uso de herencia. Se automatizó la ejecución de un algoritmo centralizado y la comparación de sus resultados con los obtenidos en los protocolos codificados.

Con este entorno se han codificado tanto protocolos de terceros, como protocolos diseñados por el grupo de investigación. Como ejemplo del entorno, se ha desarrollado el protocolo FDBU y se han generado extensas estadísticas y gráficas sobre su funcionamiento. Durante la codificación y prueba de los protocolos se iban agregando mejoras al entorno de simulación de tal manera que estuvieran disponibles para todos los protocolos, como pueden ser la información de estado de la red cada cierto tiempo y las estadísticas finales.

Fases del proyecto

- Familiarización con el entorno de desarrollo de *PeerSim*.
- Familiarización con los protocolos max-min fair.
- Diseño de las mejoras en el entorno con la creación del SPBA.
- Desarrollo e implementación del protocolo FDBU.
- Pruebas de funcionamiento y mejoras sobre el protocolo FDBU.
- Obtención de resultados, estadísticas y gráficas.
- Redacción de la memoria.

Las mejoras en el entorno han sido creadas durante todas las fases, a medida que se han necesitando para la creación y prueba de los distintos protocolos; haciendo esas mejoras comunes a todos los protocolos.

Problemas encontrados

Entre los problemas encontrados durante la realización del proyecto se encuentran los siguientes:

El GT-ITM no funcionó en los tres primeros PCs en que fue probado, mostrando errores un tanto extraños y poco descriptivos. Tras mucho evaluar el problema, se dio con la solución al ejecutar en un PC de 32 bits en lugar de uno moderno con 64 bits.

También se han encontrado problemas de memoria pese a que el PC usado para las simulaciones tenía 16 GB de RAM. Para evitar el uso de tanta memoria se redujo tanto como se pudo la tabla de rutas quitando información redundante de nodos hosts con un solo enlace de salida, y reduciendo los tipos de datos a "unsigned" (forzado en java).

El tiempo de ejecución del algoritmo centralizado era muy alto para simulaciones con muchas sesiones y redes muy grandes. Se mejoró su ejecución con varias optimizaciones e incluso se creó un segundo algoritmo centralizado muchísimo más rápido por tener menor nivel de complejidad.

Pese a que la tabla de rutas solo había que crearla una vez para cada red, puesto que se guardaban los resultados, este tiempo era altísimo por usar el algoritmo de "Dijkstra". Se solucionó parcialmente adaptando el algoritmo de "Floyd optimizado".

Algunos protocolos daban errores por adelantamiento de unos paquetes a otros. Se evitó haciendo uso de colas de salida y de entrada en cada enlace. Muchos de los protocolos de terceros fueron imposibles de hacer funcionar bien pese a que son publicados como "correctos". Ejemplo de ello pueden ser los protocolos de Churny [3] y Cobb [4].

CONCLUSIONES

Bajo el entorno de simulación "SPBU" se han ido codificando muchos protocolos, ofreciendo la mayoría de ellos resultados erróneos para casos muy concretos. Dado que el entorno ofrece la posibilidad de lanzar simulaciones automatizadas hasta que se encuentra un error, lo usual es lanzar simulaciones de cierto número de sesiones durante un plazo de tiempo, y si no da error con ninguna de las acciones generadas, ir aumentando el número e ir cambiando la topología.

Pese a que en un principio puede parecer que un protocolo funciona correctamente, en muchos casos aumentando la complejidad de interacciones entre sesiones el reparto se vuelve más complejo y muchos de los protocolos no funcionan. Es curioso cómo algunos protocolos publicados en revistas "importantes" no funcionan correctamente con simulaciones con cierta complejidad.

Con este proyecto he aprendido el funcionamiento de los protocolos de red a un nivel más concreto, he entendido cómo de importantes son las simulaciones para la comprobación del funcionamiento de estos protocolos y he afianzado mis conocimientos de programación en lenguaje Java. Además, todo ha sido desarrollado desde *Linux* dándome una gran experiencia de uso de este sistema operativo.

BIBLIOGRAFÍA

[1] CALVERT, Ken; DOAR, Matt; ZEGURA, Ellen W. *Modeling Internet Topology*. IEEE Communications Magazine, EEUU: 1997. [consulta: 20 de octubre de 2010] Disponible en Web: <http://www.cc.gatech.edu/projects/gtitm/>

[2] BOUDEC, Jean-Yves, *Rate adaptation, Congestion Control and Fairness: A Tutorial.* Ecole Polytechnique Fédérale de Lausanne (EPFL), Suiza: 2008. [consulta: 14 de septiembre de 2010]

[3] CHARNY, Anna. *An Algorithm for Rate Allocation in a Packet-Switching Network with Feedback*. Massachusetts Institute of Technology, EEUU: 1994. [consulta: 3 de diciembre de 2010]

[4] COBB, Jorge Arturo; G. GOUDA, Mohamed. *Stabilization of max-min fair networks without per-flow state.* In Sandeep S. Kulkarni and André Schiper, editors, SSS, volume 5340 of Lecture Notes in Computer Science, EEUU: 2008. [consulta: 27 de noviembre de 2010]

[5] DELFINO, Adrián; RIVERO, Sebastián; SAN MARTÍN, Marcelo. *Ingeniería de Tráfico en Redes MPLS* [PFC]. Instituto de Ingeniería Eléctrica, Facultad de Ingeniería de la República, Méjico: 2005. [consulta: 7 de enero de 2011]

[6] JESI, Gian Paolo; JELASITY, Márk; MONTRESOR, Alberto; VOULGARIS, Spyros. *PeerSim P2P Simulator* [en línea]. Italia: 2009, [consulta: 15 de junio de 2010]. Disponible en Web: <http://peersim.sourceforge.net>

[7] MOROTÓ OSÉS, Daniel. *Quality of Service* [material gráfico proyectable]. UPNA, EEUU: 2008/2009, [consulta: 18 de junio de 2010]. Disponible en web: <https://www.tlm.unavarra.es/~daniel/docencia/nsri/nsri08_09/slides/Tema1-QoS.pdf>

[8] THOMAS, Megan; EDWARDS, Elizabeth; BHATTACHARJEE, Samrat. *Modeling Topology of Large Internetworks (Gt-Itm)* [en línea]. Georgia: 2007, [consulta: 30 de junio de 2010]. Disponible en Web: <http://www.cc.gatech.edu/projects/gtitm/ >

[9] ZEGURA, Ellen W.; CALVERT, Ken; S. Bhattacharjee. *How to Model an Internetwork*. Proceedings of IEEE Infocom '96, EEUU: 1996, [consulta: 21 de octubre de 2010]. Disponible en Web: <http://www.cc.gatech.edu/projects/gtitm/papers/howto.ps.gz>

[10] ZEGURA, Ellen W.; CALVERT, Ken; DONAHOO, M. Jeff. *A Quantitative Comparison of Graph-based Models for Internet Topology*. IEEE/ACM Transactions on Networking, EEUU: 1997, [consulta: 30 de junio de 2010]. Disponible en Web: <http://www.cc.gatech.edu/projects/gtitm/papers/ton-model.ps.gz>

Siguiendo las normas ISO 690-1987 y ISO 690-2

APÉNDICE

ANEXO A: *Código fuente "fdbu.Edge"*

A continuación se expone el código fuente de la clase "Edge", escrito en *java*, de cuya instanciación surgen los objetos enlaces unidireccionales para el protocolo FDBU.

```
package fdbu;
import java.util.ArrayList;
import comunes.NodeComun;
import comunes.NodeHost;
import comunes.Packet;
import comunes.Timer;
public class Edge extends comunes.EdgeComun {
	public ArrayList<Sesion> sessions;
	public Edge(long vel, int nodoA, int nodoB) {
		super(vel, nodoA, nodoB);
		sessions=new ArrayList<Sesion>();
	}
	public static double mmfr(ArrayList<Sesion> STotal, double be){
		ArrayList<Sesion> S2 = new ArrayList<Sesion>();
		ArrayList<Sesion> S1 = new ArrayList<Sesion>();
		Sesion s;
		S1.addAll ( STotal );
		while (S1.size()>0 && ( (s=getMenorVel(S1)).bw <
		(be/(S1.size()+1)))){
			S2.add(s);
			S1.remove(s);
			be -= s.bw;
		}
		return be/(S1.size()+1);
	}
	private static Sesion getMenorVel(ArrayList<Sesion> sessionsx) {
		Sesion s;
		Sesion min;
		min = sessionsx.get(0);
		for (int i=1; i<sessionsx.size(); i++ ){
			s = sessionsx.get(i);
			if ( s.bw < min.bw )
				min = s;
		}
		return min;
	}
	private double bwMax(ArrayList<Sesion> S) {
		double bMaxAux = 0;
		for(int i=0;i<S.size();i++){
			if(S.get(i).bw>bMaxAux)
				bMaxAux=S.get(i).bw;
		}
		return bMaxAux;
	}
	public void closeSesion(int nobj){ //SOLO PARA EDGES DE NODOS HOSTS
		if(((NodeComun) esteNodo()).isHost()){
			if(numSesionesEntre(esteNodoId(),nobj)>1){
				error("Hay varias sesiones con ese nodo. Posible
				sesión anticuada en
				proceso de cierre");
				return;
```

```
                }
                Sesion s = buscarSesion(esteNodoId(),nobj);
                if(s==null){
                        debug("No existe la sesión a cerrar");
                        return;
                }
                //s.esperandoSetRateClose=true;
                int este = esteNodoId();
                Paquete paq=new Paquete(este, nobj, este, nobj,
                getnSes(este,nobj), 0,
                PaqueteTipo.RATEREQUEST);
                this.procesaPk(paq);
        }else{
                error("No es un nodo Host el que intenta cerrar una
                sesión");
        }
}
public void addSesion(int nobj) { //SOLO PARA EDGES DE NODOS HOSTS
        if(((NodeComun) esteNodo()).isHost()){
                Sesion s = buscarSesion(esteNodoId(),nobj);
                if (s!=null){
                        if(!s.esperandoSetRateClose){
                                error("Se esta intentando añadir una sesión que
                                ya existe (no
                                esta ni en proceso de cierre)");
                                return;
                        }else{
                                debug("Se añadido la sesión aunque ya existía
                                una en proceso
                                de cierre entre esos nodos");
                        }
                }
                NodeHost este = (NodeHost) esteNodo();
                long nSes=este.sigCont();
                Paquete paq=new Paquete(este.getID(), nobj, este.getID(),
                nobj, nSes, 0,
                PaqueteTipo.JOIN);
                this.procesaPk(paq);
                Paquete paq2=new Paquete(este.getID(), nobj, este.getID(),
                nobj, nSes,
                Float.MAX_VALUE, PaqueteTipo.RATEREQUEST);
                this.procesaPk(paq2);
        }else{
                error("No es un nodo Host el que intenta agregar una nueva
                sesión");
        }
}
@Override
public void procesaPk(Packet paq) {
        Paquete paquete = (Paquete) paq;
        debug("procesando paquete "+paquete);
        debug("sesiones antes: " +sessions);
        if(paquete.tipo==PaqueteTipo.LEAVE){
                this.pLeave((Paquete) paquete);
        }else if(paquete.tipo==PaqueteTipo.RATEREQUEST){
                this.pRateRequest(paquete);
        }else if(paquete.tipo==PaqueteTipo.SETRATE){
                this.pSetRate(paquete);
        }else if(paquete.tipo==PaqueteTipo.UPDATERATE){
```

```
                this.pUpdateRate(paquete);
        }else if(paquete.tipo==PaqueteTipo.JOIN){
                this.pJoin(paquete);
        }else{
                error("Paquete desconocido: "+paquete.tipo);
        }
        debug("sesiones despues: " +sessions);
}
private void pRateRequest(Paquete p) {//Procesar RateRequest
        if(esteNodoId()!=p.dst){ //Si no eres el destinatario (dst
        siempre)
                Sesion s = buscarSesion(p.srcSes,p.dstSes,p.nSes);
                if(s==null){
                        debug("RateRequest obsoleto");
                        return;
                }
                if(p.bw==0)s.esperandoSetRateClose=true;
                if(s.esperandoSetRateClose && p.bw!=0)return;
                s.codActualizacionActual=p.codActualizacion;
                //s.bwMoe=p.bw;
                double mmfr=mmfr(sesionesSin(sessions,s),bw);
                if(mmfr<p.bw){
                        p.bw=mmfr;
                        //p.limitador=esteNodoId();
                }
                enviarPaquete(p); //downStream_RateRequest
        }else{
                //Rebotar con SetRate
                if(esteNodoId()!=p.dstSes)error("El destino de un paquete
                RateRequest no es el destino de la sesión");
                Paquete paq=new Paquete(esteNodoId(), p.srcSes, p.srcSes,
                p.dstSes, p.nSes,
                p.bw, PaqueteTipo.SETRATE,p.codActualizacion);
                //paq.limitador=p.limitador;
                enviarPaquete(paq);
        }
}
private void pSetRate(Paquete p) { //Procesar SetRate
        if(esteNodoId()!=p.dstSes){
                Sesion s = buscarSesion(p.srcSes,p.dstSes,p.nSes);
                if(s==null){
                        error("Sesión indicada por setRate no existe o es
                        anticuado.");
                        return;
                }
                if(s.esperandoSetRateClose && p.bw!=0){
                        debug("Se estaba esperando setRate de cierre.
                        Descartado.");
                        return;
                }
                if(s.codActualizacionActual!=p.codActualizacion && p.bw!=0){
                        debug("SetRate con codActualizacion no aceptado");
                        return;
                }
                double mmfr=mmfr(sesionesSin(sessions,s),bw);
                if(p.bw<=mmfr){
                        //if(p.limitador!=esteNodoId())s.bwMoe=p.bw;
                        if(s.bw!=p.bw){
                                boolean aumentada=s.bw<p.bw;
```

```
                double maxAux= bwMax(sesionesSin(sessions,s))-
                0.1;
                s.bw=p.bw;
                ArrayList<Sesion> sessionsAux = new
                ArrayList<Sesion>();
                sessionsAux.addAll(sessions);
                for(int i=0; i<sessions.size();i++){ //Si alguno
                tiene
                      asignado de mas
                      Sesion s2=getMenorVel(sessionsAux);
                      sessionsAux.remove(s2);
                      if(s2!=s){
                            if( (s2.codActualizacionActual!=0)
                            || (aumentada &&
                            s2.bw>mmfr(sesionesSin(sessions,s2),
                            bw) )
                            || (!aumentada))// &&
                            (s2.bw<s2.bwMoe || s2.bw>=maxAux ) ))
                            s2.actualizacionPendiente=true;
                      }
                }
          }
    }else{
          debug("Paquete ignorado por ser obsoleto. Velocidad
          reducida a
          Double.MIN_VALUE");
          p.bw=Double.MAX_VALUE;
    }
    s.codActualizacionActual=0;
    if(p.dst!=esteNodoId()) enviarPaquete(p);
    //forward_setRate(s,s.bw)
    if(!actualizacionesPendientes()){
          for(int i=0; i<sessions.size();i++){ //Si alguno tiene
          asignado de mas
                Sesion s2=sessions.get(i);
                if(s2.actualizacionPendiente){
                      s2.actualizacionPendiente=false;
                      generarUpdateRate(s2);
                }
          }
    }
    s.codActualizacionActual=0;
    if(s.updateRatePendiente!=null){
          if(!esteNodo().isHost()) error("Nodo no host con
          updateRatePendiente");
          debug("Procesando UpdateRate pendiente");
          procesaPk(s.updateRatePendiente);
    }
    if(s.esperandoSetRateClose && p.bw==0 &&
    esteNodoId()==s.srcSes){
          if(!esteNodo().isHost()) error("Nodo no HOST generando
          paquete
          Leave");
          s.esperandoSetRateClose=false;
          Paquete paq=new Paquete(esteNodoId(), s.dstSes,
          esteNodoId(),
          s.dstSes, s.nSes, 0, PaqueteTipo.LEAVE);
          this.procesaPk(paq);
    }
    if(esteNodo().isHost() && s.generarEstadisticas){
```

```
                        if(p.bw!=Double.MAX_VALUE) s.generarEstadisticas();
                }
        }else{
                error("NodoHost dstSes procesando paquete SetRate");
        }
}
private boolean actualizacionesPendientes() {
        for(int i=0; i<sessions.size();i++){ //Si alguno tiene asignado de
        mas
                if(sessions.get(i).codActualizacionActual!=0) return true;
        }
        return false;
}
private void pLeave(Paquete p) { //Procesar Leave
        //System.out.println("close: "+sessions);
        if(esteNodoId()!=p.dst){
                Sesion s = buscarSesion(p.srcSes,p.dstSes,p.nSes);
                if(s==null){
                        error("Sesion indicada por close no existe.");
                        return;
                }
                sessions.remove(s);
                enviarPaquete(p); //downStream
        }
}
private void pUpdateRate(Paquete p) { //Procesar Leave
        if(esteNodoId()==p.dstSes)error("El nodo destino de la sesión ha
        recibido un paquete Update");
        Sesion s = buscarSesion(p.srcSes,p.dstSes,p.nSes);
        if(s==null){ //Si existe la sesion indicada en AvailableRate
                debug("No existe la sesión indicada en el UpdateRate");
                return;
        }
        if(s.esperandoSetRateClose)return;
        if(esteNodoId()==p.dst){ //== srcSes
                if(s.codActualizacionActual!=0){
                        debug("UpdateRate bloqueado. Pendiente de procesar
                        hasta acabar
                        actualizacion "+s.codActualizacionActual+".");
                        s.updateRatePendiente=p;
                        return;
                }
                s.updateRatePendiente=null;
                Paquete paq=new Paquete(esteNodoId(), p.dstSes, p.srcSes,
                p.dstSes, p.nSes,
                Double.MAX_VALUE,
                PaqueteTipo.RATEREQUEST,p.codActualizacion);
                procesaPk(paq);
        }else{
                enviarPaquete(p);
        }
}
private void pJoin(Paquete p) { //Procesar Leave
        if(esteNodoId()!=p.dst){
                Sesion s = buscarSesion(p.srcSes,p.dstSes,p.nSes);
                if(s!=null){ //Si existe la sesion indicada en AvailableRate
                        error("La sesión indicada por el Join ya existe");
                        return;
                }
                s=new Sesion(p.srcSes,p.dstSes,p.nSes,p.bw);
```

```
                s.bw=0;
                sessions.add(s);
                enviarPaquete(p);
        }else{
                debug("Nodo destino de la sesión ignorando paquete Join");
        }
}
//Otras funciones auxiliares:
public void generarUpdateRate(Sesion s){
        Paquete paq=new Paquete(esteNodoId(), s.srcSes, s.srcSes,
        s.dstSes, s.nSes,
        Double.MAX_VALUE, PaqueteTipo.UPDATERATE);
        if(esteNodoId()!=s.srcSes)enviarPaquete(paq);
        else processEvent(paq);//Util.aleatorio(1,20)
}
public static ArrayList<Sesion> sesionesSin(ArrayList<Sesion> S, Sesion
s){
        ArrayList<Sesion> SAux = new ArrayList<Sesion>();
        SAux.addAll(S);
        SAux.remove(s);
        return SAux;
}
private Sesion buscarSesion(int nA, int nB, long nCont){
        for(int i=0; i<sessions.size();i++){
                Sesion s = sessions.get(i);
                if(s.equal(nA,nB,nCont)){
                        return s;
                }
        }
        return null;
}
private int numSesionesEntre(int nA, int nB) {
        int cont=0;
        for(int i=0; i<sessions.size();i++){
                Sesion s = sessions.get(i);
                if(s.srcSes==nA && s.dstSes==nB){
                        cont++;
        }
}
return cont;
}
private long getnSes(int nA, int nB) {
        Sesion s = buscarSesion(nA,nB);
        if(s==null)return 0;
        else return s.nSes;
}
private Sesion buscarSesion(int nA, int nB) {
        for(int i=0; i<sessions.size();i++){
                Sesion s = sessions.get(i);
                if(s.srcSes==nA && s.dstSes==nB){
                        return s;
                }
        }
        return null;
}
@Override
public void apliTimeout(Timer t) {
        error("No es usado apliTimeout");
}
@Override
```

```
	public void changeSesion(int nodeDst, double bwMax) {
		error("Cambio de sesiones no implementado");
	}
}
```

aquí acaba la memoria

Insturcciones/notas (para la edición):

edición final memoria:

Imágenes y tablas en hojas distintas dejando muxo espacion en blanco NOOO!

Capítulos, bibliografía y apendice empiezan en hoja impar

Actualizar indices de contenido, tablas y figuras

poner al índice "capitulo n. " en los capítulos.

Sangrar tabla

quitar marca de agua (diseño de pagina/marca de agua/qitar

edición final código:

quitar paquetes peerSim innecesarios

quitar bneckv6

limpiar todo el código de código antiguo/comentado

poner private metodos posibles

poner comentarios generales

quitar imports sobrantes

revisar warnigs (amarillos)

Encuadernar:

Portada azul celeste

Adjuntar: excel de gráficas, workspace (src, bin), directorio Ejecutar, memoria