Ingeniería de Tráfico en Redes MPLS

Adrián Delfino
Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay
a-delfino@adinet.com.uy

Sebastián Rivero
Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay
sebarivero@mercurio.com.uy

Marcelo San Martín
Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay
msanmar@adinet.com.uy

RESUMEN

El rápido crecimiento de la Internet y de los distintos servicios ofrecidos (como ser VoIP, videoconferencia, etc.) estimula la necesidad de los ISPs de poseer sofisticadas herramientas de gestión de redes de manera de lograr un uso óptimo de los recursos de la red, al mismo tiempo que se debe determinar con cuidado por dónde dirigir el tráfico de cada demanda de manera de no ver comprometidos los SLAs. El presente trabajo introduce una herramienta de software que permite analizar las distintas prestaciones que se pueden obtener al aplicar algoritmos de Ingeniería de Tráfico sobre una red de computadoras basadas en MPLS. Ofrece una interfaz gráfica para el diseño de la topología de la red, distintas herramientas de visualización del estado actual de la misma y varios algoritmos para el establecimiento de LSPs. Se pudo comprobar el correcto funcionamiento de los algoritmos en base a pruebas y ensayos y comparando contra ejemplos ya conocidos. Se comprobó además con éxito el funcionamiento de la carga automática de la red en la red multiservicio del IIE de la Facultad. Todo esto le permite a un proveedor de red el experimentar con cambios en la configuración de la red en un ambiente simulado, en vez de en una red operacional, basándose en una plataforma para investigaciones del tipo "what-if" de ingeniería de tráfico.

1. INTRODUCCIÓN

La Ingeniería de Tráfico (TE) es una disciplina que procura la optimización de la performance de las redes operativas. La Ingeniería de Tráfico abarca la aplicación de la tecnología y los principios científicos a la medición, caracterización, modelado, y control del tráfico que circula por la red (Ver [1]). Las mejoras del rendimiento de una red operacional, en cuanto a tráfico y modo de utilización de recursos, son los principales objetivos de la Ingeniería de Tráfico. Esto se consigue enfocándose a los requerimientos del rendimiento orientado al tráfico (tiene como prioridad la mejora de los indicadores relativos al transporte de datos, como ser minimizar la pérdida de paquetes, minimizar el retardo, maximizar el throughput, etc), mientras se utilizan los recursos de la red de una manera fiable y económica (el objetivo es en este caso el optimizar la utilización de los recursos de la red de manera que no se saturen partes de la misma mientras otras permanecen subutilizadas).

Todo lo anterior apunta a un objetivo global, que es minimizar la congestión. Un reto fundamental en la operación de una red, especialmente en redes IP públicas a gran escala, es incrementar la eficiencia de la utilización del recurso mientras se minimiza la posibilidad de congestión.

Claramente, la congestión es un fenómeno nada deseable y es causada por ejemplo por la insuficiencia de recursos en la red. En casos de congestión de algunos enlaces, el problema se resolvía a base de añadir más capacidad a los enlaces. La otra causa de congestión es la utilización ineficiente de los recursos debido al mapeado del tráfico. El objetivo básico de la Ingeniería de Tráfico es adaptar los flujos de tráfico a los recursos físicos de la red. La idea es equilibrar de forma óptima la utilización de esos recursos, de manera que no haya algunos que estén sobre-utilizados, creando cuellos de botella, mientras otros puedan estar subutilizados.

La componente del packet forwarding (ver [2]) de TE contiene a Multi Protocol Label Switching (MPLS), responsable de dirigir un flujo de paquetes IP a lo largo de un camino predeterminado a través de la red. La clave detrás de MPLS es el mecanismo de asignación e intercambio de etiquetas en que se basa. Esas etiquetas son las que permiten que se establezcan las rutas que siguen los paquetes entre dos nodos de la red. Esa ruta a seguir se la conoce como ruta conmutada de etiquetas (LSP).

Los Label Edge Routers (LER) operan en los extremos de la red MPLS y se encargan de interconectar a ésta con la red de acceso, asignando una etiqueta a cada paquete de información entrante. Los paquetes, una vez etiquetados por el LER, viajan por la red MPLS a través de los routers de conmutación de etiquetas (LSRs). Estos se encargan básicamente de dirigir el tráfico en el interior de la red, según sea la etiqueta que contenga el paquete y dirigiéndolo por el LSP correspondiente.

Un conjunto de paquetes que comparten los mismos requerimientos para su transporte, pertenecen a la misma FEC (Forwarding Equivalence Class). Todos los paquetes que pertenezcan a la misma FEC seguirán el mismo LSP para llegar a destino.

Como vemos, es muy importante el establecimiento de los LSPs, cómo elegirlos y dónde ubicarlos de manera de cubrir de la mejor manera posible las demandas. Con tal motivo, desarrollamos una herramienta de software que tiene cuatro distintos mecanismos para ofrecer al usuario, al momento de elegir cómo y por dónde ubicar a los LSPs. En primer lugar, dentro de los algoritmos que podríamos considerar "online" tenemos al Ruteo Explícito y al

Constraint Shortest Path First (CSPF). En ambos se considera que las demandas van llegando una a la vez, de manera que el asunto es ver la manera de alojar tal demanda por determinado LSP considerando las restricciones exigidas por el usuario en términos del ancho de banda y el estado actual de la red (posibles LSPs ya existentes que ocupan ya recursos de la red). Por otro lado tenemos a los algoritmos "offline" que son el Minimum Interference Routing Algorithm (MIRA) y los de Fairness. En este caso, se consideran todas las demandas existentes hasta el momento de manera conjunta y se determina la mejor manera de alojarlas en la red, haciendo un aprovechamiento inteligente de los recursos de manera que la mayoría de ellos vean sus requerimientos satisfechos.

Otra importante funcionalidad en la que se hizo hincapié fue la de implementar un mecanismo automático de carga de la red, que consiste básicamente en descubrir todos los routers presentes en la red, que estén intercambiando información de ruteo mediante el protocolo OSPF. Ésta es una manera muy práctica de poder levantar la topología de la red en caso que el usuario no tenga conocimiento de la misma, o en caso que se trate de una red de gran tamaño en cuyo caso se consumiría mucho tiempo en crearla manualmente.

Finalmente, es importante destacar que en todo momento el usuario tiene la posibilidad de visualizar el estado actual de la red, como ser información sobre utilización de los enlaces, LSPs establecidos, enlaces que conforman los LSPs, etc. Esta información es desplegada en forma numérica. Pero en caso de también desearlo, existe una manera gráfica de poder visualizar en pantalla la utilización de los enlaces, por medio de la diferenciación por colores de los enlaces.

Los conceptos básicos sobre CSPF, MIRA, y Fair Networks, son explicados en la sección 2 del presente documento. En la sección 3 presentaremos los principales casos de uso y en la sección 4 trataremos la arquitectura del software. Finalmente en una quinta sección se presentarán las conclusiones.

2. CSPF, MIRA y FAIR NETWORKS

Veremos a continuación los principales conceptos encerrados detrás de estos tres diferentes mecanismos de establecimiento de LSPs, de manera que se distingan las diferencias principales entre unos y otros.

2.1 CSPF

La computación de caminos para protocolos de ruteo intradominio se basa en un algoritmo que optimiza (minimiza) una métrica escalar en particular. Para el caso del protocolo Open Shortest Path First (OSPF) un administrador de red asigna a cada enlace en la red una métrica administrativa. Dada la opción de múltiples caminos a un destino dado, OSPF usa el algoritmo de Dijkstra del camino más corto (SPF) para computar el camino que minimiza la métrica administrativa del camino.

Para el caso de algoritmos de ruteo basados en restricciones (CBR) se requiere la habilidad de computar un camino de manera tal que sea óptimo respecto a alguna métrica escalar y que no viole un conjunto de restricciones.

La computación de un camino se puede lograr usando un algoritmo de shortest path first (SPF). El algoritmo SPF plano, computa un camino que es óptimo con respecto a cierta métrica escalar. Entonces, para computar un camino que no viole restricciones, todo lo que necesitamos es el modificar el algoritmo de manera tal que pueda tomar en cuenta esas restricciones. Nos referiremos a tal algoritmo como Constraint Shortest Path First (CSPF) (Ver [3]). Observar la Figura 2.1.

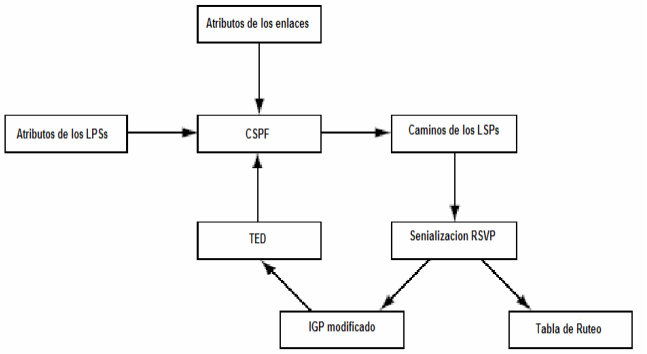

Figura 2.1: Proceso de computación del CSPF.

El algoritmo CSPF requiere que el router que realiza la computación del camino tenga información sobre todos los enlaces en la red. Esto impone una restricción en el tipo de protocolo de ruteo que podemos usar restringiéndonos a protocolos de estado de enlace como IS-IS u OSPF. Como comentario sobre uno del resto de los mecanismos restantes necesarios para soportar CBR, la capacidad de ruteo explícito necesaria es provista por MPLS.

2.2 MIRA

El objetivo es minimizar la "interferencia" que provoca el establecimiento de un nuevo LSP a potenciales nuevos LSPs que son desconocidos (Ver [5] y [6]). Mapear un LSP en la red puede reducir el máximo ancho de banda disponible entre algunos pares de nodos ingreso-egreso críticos en la red, dependiendo de por dónde se dirija el mismo. Este fenómeno es conocido como "interferencia".

Una solución es que los caminos que reducen la cantidad de ancho de banda disponible entre otros nodos ingreso-egreso sean evitados.

Para ello, se asocia a cada enlace de la red un peso y luego se aplica SPF a la red. Este peso es proporcional a cuán crítico es el enlace para el establecimiento de LSPs entre pares de nodos ingreso-egreso. De esta manera futuras

demandas entre estos nodos tendrán baja probabilidad de ser rechazadas.

La implementación del algoritmo se explica a continuación.

4. Primero asociamos un peso a cada enlace. Nos encontramos así con los nodos Ingreso-Egreso, que se deben a que desde ciertos nodos ingreso pueden ser permitidos LSPs a ciertos nodos egreso solamente.

Calculamos el máximo ancho de banda entre cada par de nodos y lo llamamos $\theta^{s'd'}$. Se calcula para cada enlace qué proporción del máximo ancho de banda entre (s', d') pasa por el enlace: $\frac{f_l^{s'd'}}{\theta^{s'd'}}$ para el enlace *l*. En este caso, $f_l^{s'd'}$ representa la cantidad de tráfico del flujo máximo de red, que pasa por el enlace *l*.

Tenemos también que tener en cuenta el ancho de banda residual en cada enlace: $\frac{f_l^{s'd'}}{\theta^{s'd'} R(l)}$ que corresponde al peso que se asocia debido al par de nodos (s', d').

Luego asignamos a cada enlace el peso total debido a las contribuciones de todos los pares de nodos ingreso-egreso (*s', d'*):

$$w(l) = \sum_{(s',d')\in L} \frac{f_l^{s'd'}}{\theta^{s'd'} R(l)}, l \in E$$

4. Se eliminan todos los enlaces que no cumplen con la restricción de ancho de banda del nuevo LSP.
4. Corremos el SPF a una topología reducida.
4. Establecemos el LSP en la red.

2.3 FAIR NETWORKS

El determinar cuánto tráfico de cada flujo debe ser admitido por la red y por dónde rutear al mismo una vez que ingresa, satisfaciendo los requerimientos de alta utilización de la red y garantizando justicia a los usuarios, es uno de los retos más grandes en el diseño de las redes de telecomunicaciones de hoy en día.

La pregunta que nos debemos hacer es la siguiente: ¿qué principio debemos seguir al alojar las demandas entre los recursos que la red tiene pare ofrecer, de manera de cumplir con algún criterio de justicia?

Una posible respuesta es utilizar el esquema de asignación de recursos llamado Max-Min Fairness (MMF). La idea básica de Max-Min Fairness es incrementar lo máximo posible el BW de una demanda sin que sea a expensa de otra demanda (Ver [7]).

En nuestro software, implementamos cuatro distintos algoritmos de MMF: Max-Min Fairness básico para caminos fijos, Max-Min Fairness para caminos fijos acotados, Max-Min Fairness con múltiples caminos y Max-Min Fairness con múltiples caminos acotados.

2.3.1 Max-Min Fairness básico para caminos fijos

Consideremos una red con enlaces de capacidades fijas y con caminos prefijados únicos asignados para transportar los flujos de las demandas (o sea que en caso de existir varios caminos posibles calculados por CSPF entre determinado par de nodos, nos quedamos solamente con uno sólo de ellos, pudiendo éste ser elegido por el usuario a su gusto). El usuario tiene como opciones de métrica para calcular los caminos en CSPF, por peso (se le asigna a cada enlace un determinado peso administrativo) o minhop (se le asignan a los enlaces un peso igual a uno y se calcula el camino en base a la cantidad de saltos).

Básicamente la idea es incrementar de a pasos el BW de cada demanda hasta que satura al menos un enlace por el que pasan una o varias demandas. El valor de dicho paso se calcula de la siguiente manera:

Paso = mínimo {capacidad/cantidad de damandas que pasan por él}

Nuestro objetivo es hallar el vector de asignación de recursos $x^* = (x_1^*, x_2^*, ..., x_D^*)$ donde D es la cantidad de demandas. Dicho vector indicará cuánto BW llevará cada demanda.

Los pasos para resolver este algoritmo se resumen en los siguientes:

Paso 0: Poner $x^* = 0$

Paso 1: $t := \min\left\{ c_e / \sum_d \delta_{ed} : e = 1, 2, ..., E \right\}$

Paso 2:

$$c_e := c_e - t\left(\sum_d \delta_{ed}\right) \quad para \quad e = 1,2,...,E; \quad x_d^* := x_d^* + t \quad para \quad d = 1,2,...,D$$

En donde, *d* = 1, 2,…, D son las demandas y *e* = 1, 2,…, E los enlaces. Por otro lado, las constantes se definen de manera tal que δ_{ed} = 1 si el enlace *e* pertenece al camino fijo de la demanda *d* y 0 en otro caso; y c_e es la capacidad del enlace *e*.

Removemos todos los enlaces saturados (todos los enlaces *e* con c_e=0). Para cada enlace removido *e*, remover todos los caminos y correspondientes demandas que usan ese enlace removido (todas las *d* con $\delta_{ed} = 1$).

Paso 3: Si no queda ninguna demanda más, entonces detenerse. En caso contrario ir al paso 1.

2.3.2 Max-Min Fairness básico para caminos fijos con cotas

Ahora seguiremos considerando el caso del MMF para caminos fijos, pero agregándole ahora pesos a las demandas y cotas superior e inferior a los flujos asignados a ésas demandas.

Agrega a cada demanda un enlace virtual de capacidad igual al BW deseado multiplicado por la prioridad.

En este caso, los índices *d* y *e* siguen señalando a las demandas y enlaces respectivamente. A su vez, las constantes δ_{ed} y c_e se definen de igual manera que

anteriormente. Las nuevas constantes son w_d que indica el peso de la demanda *d; h_d* que indica el límite inferior para el flujo de la demanda *d* y H_d que corresponde al límite superior para el flujo de la demanda *d.*

Los pasos para resolver este algoritmo son los siguientes:

Paso 0: Poner $x_d^* = h_d$ para $d = 1, 2,\ldots,$ D y

$$c_e := c_e - \sum_d \delta_{ed} w_d h_d \quad para \quad e = 1,2,...,E;$$

Incrementar de a pasos el bw de cada demanda hasta que satura al menos un enlace por el que pasan una o varias demandas o alcanza el bw deseado por la demanda. Remover todos los enlaces saturados (todos los *e* con c_e = 0). Para cada enlace removido *e* remover todos los caminos y correspondientes demandas que usan el enlace removido (todas las *d* con $\delta_{ed} = 1$).

Paso 1: Calculamos

$$t = \min\{c_e / \sum_d w_d \delta_{ed} \; ; \; e = 1,2,...,E\}$$

Paso 2:

$$c_e = c_e - t(\sum_d w_d \delta_{ed}) \; ; \; e = 1,2,...,E \; ; \; x_d^* = x_d^* + t \;\; con \; d = 1,2,...D$$

Remover todos los enlaces saturados (todos los *e* con c_e=0). Para cada enlace removido *e* remover todos los caminos y correspondientes demandas que usen el enlace removido (todas las *d* con $\delta_{ed} = 1$).

Paso 3: Si no quedan más demandas entonces detenerse. En caso contrario ir al paso 1.

La implementación de este algoritmo consiste básicamente en agregar un enlace y nodo ficticio para cada demanda, de manera de asegurarse que por dicho enlace pase solamente ésa demanda en particular. O sea que cada demanda tendrá un enlace y nodo ficticio propio de ella. La capacidad de dicho enlace ficticio será igual al ancho de banda requerido para la demanda en cuestión. De esta manera, nos aseguramos que el ancho de banda no se aumente más que el ancho de banda requerido por el usuario. Es una manera de poner un tope superior al ancho de banda.

En las próximas secciones consideraremos un problema más general para MMF, que involucrará optimización simultánea de caminos y flujos. Nos referimos a este tipo de problemas como problemas de caminos flexibles, en contraste con los problemas de caminos fijos vistos hasta ahora. En particular, el problema analizado considerará ahora el caso donde múltiples caminos son posibles candidatos en la lista de ruteo asignada a cada demanda. Esto implica por ende que los flujos que satisfacen el volumen de cada demanda son divididos entre los posibles caminos.

2.3.3 Max-Min Fairness para múltiples caminos

Calcula todos los caminos posibles para cada demanda, sin considerar métrica alguna. Esto permite separar cada demanda en varias sub demandas.

Tomemos x_{dp} como el flujo (BW) asignado al camino *p* de la demanda *d,* y a *Xd* como el flujo total (BW) asignado a la demanda *d*, $X = (X_1, X_2,\ldots,X_D)$. Para este algoritmo aparece el índice $p = 1, 2,\ldots, P_d$ para indicar los caminos candidatos para la demanda *d.* Dentro de las constantes tenemos ahora $\delta_{edp} = 1$ si el enlace *e* pertenece al camino *p* de la demanda *d* y 0 en caso contrario. Y finalmente, dentro de las variables tenemos a x_{dp} para representar el flujo (ancho de banda) asignado al camino *p* de la demanda *d;* y a X_d como el flujo total asignado a la demanda *d*, $X = (X_1, X_2,\ldots, X_D)$.

Los pasos para resolver este algoritmo son los siguientes:

Paso 0: Resolvemos el problema de optimización lineal (LP) siguiente: **maximizar** *t*

sujeto a

$$X_d = \sum_p x_{dp} \qquad d = 1,2,...,D$$

$$t - X_d \le 0 \qquad d = 1,2,...,D$$

$$\sum_d \sum_p \delta_{edp} x_{dp} = c_e \qquad e = 1,2,...,E$$

con todos los $x_{dp} \geq 0$

Observar que el objetivo en el que estamos interesados es en realidad:

maximizar **min** $\{X_d : d = 1, 2,\ldots, D\}$

Ahora nombremos (t^*, x^*, X^*) a la solución óptima del mismo. Pongamos $n := 0$, $Z_0 := \varnothing$, $Z_1 := \{1, 2,\ldots,D\}$, y $t_d := t^*$ para cada $d \in Z_1$.

Paso 1: Pongamos $n := n + 1$. Empecemos considerando las demandas $d \in Z_1$ una por una, para chequear si el volumen total de asignación X_d^* puede hacerse mayor a t^*, sin decrementar las máximas asignaciones ya encontradas $t_{d'}$ para el resto de todas las demandas *d'*. El chequeo es llevado a cabo por el test de no bloqueo siguiente:

maximizar X_d

sujeto a

$$X_{d'} = \sum_p x_{d'p} \qquad d' = 1,2,...,D$$

$$t_{d'} - X_{d'} \le 0 \qquad d' = 1,2,...,D \qquad (t_{d'} - const)$$

$$\sum_{d'} \sum_p \delta_{ed'p} x_{d'p} \le c_e \qquad e = 1,2,...,E$$

con todos los $x_{d'p} \geq 0$

Si no hay demandas que bloqueen en Z_1 (demanda $d \in Z_1$ se considera que bloquea, si X_d no puede incrementarse más) entonces ir al Paso 2. En caso contrario, cuando la primer demanda que bloquee se encuentre, digamos demanda *d*, agregar *d* al conjunto Z_0 y borrarla del conjunto Z_1 ($Z_0 := Z_0 \cup \{d\}$, $Z_1 := Z_1 \backslash \{d\}$). Si $Z_1 = \varnothing$, entonces detenerse el vector

$X^* = (X_1^*, X_2^*, ..., X_D^*) = (t_1, t_2, ..., t_D)$ es la solución para este problema. En caso contrario proceder al Paso 2.

Paso 2: Solucionar el problema LP siguiente para mejorar las mejores asignaciones totales actuales:

maximizar t

sujeto a

$$X_d = \sum_p x_{dp} \qquad d = 1, 2, ..., D$$

$$t - X_d \le 0 \qquad d = Z_1$$

$$t_d - X_d \le 0 \qquad d = Z_0 \qquad (t_d - const)$$

$$\sum_d \sum_p \delta_{edp} \qquad e = 1, 2, ..., E$$

$$x_{dp} \le c_e$$

con todos los $x_{dp} \ge 0$

Nombremos *(t^*, x^*, X^*)* a la solución del último problema LP. Pongamos $t_d := t^*$ para cada $d \in Z_1$ y vayamos entonces al Paso 1 nuevamente.

Observar que puede suceder que la solución óptima de lo anterior no incremente al t^* porque puede que haya más demandas que bloqueen además de la detectada en el Paso 1.

La salida del test de no bloqueo es positiva, lo cual significa que la demanda *d* es no bloqueadora si el óptimo X_d es estrictamente mayor que t^*. En otro caso, la demanda considerada resulta ser bloqueadora en realidad.

2.3.4 Max-Min Fairness para múltiples caminos acotado

La idea detrás de este algoritmo es la siguiente. Es básicamente el mismo algoritmo que el expuesto anteriormente; la principal diferencia es que se detiene una vez que se llega a obtener el ancho de banda requerido por el usuario.

Como vemos entonces, los pasos para resolver el algoritmo 4 son los mismos que los usados en el algoritmo 3. La única diferencia es que se hace un tratamiento de los resultados arrojados por el algoritmo 3 de manera de, en caso de ser suficiente el ancho de banda que porta cada demanda, detenerse en cuanto se llega al valor del ancho de banda requerido.

3. PROBLEMAS A RESOLVER

Veremos ahora cuáles son los principales problemas planteados en el proyecto y cuáles fueron las distintas soluciones creadas para cada uno de ellos dentro del software NET-TE.

Se distinguen tres grandes problemas u objetivos trazados dentro de NET-TE: construcción de la topología de la red de trabajo, establecimiento de los LSPs por los cuáles pasará el tráfico de cada demanda y visualización del estado actual de la red.

3.1 CONSTRUCCIÓN DE LA TOPOLOGÍA

Para empezar a trabajar, lo primero que necesita hacer el usuario es construirse la topología de la red sobre la cual va a trabajar. NET-TE ofrece dos maneras de realizar esto: una manual y otra automática.

En el caso del método manual de construcción, el usuario dispone de dos posibles objetos para crear su topología: routers y links. Los routers que se ofrecen son de dos tipos: LERs y LSRs.

Los campos que ofrece NET-TE para configurar los enlaces son los siguientes: ancho de banda, peso administrativo y afinidad**.** La opción del uso de pesos administrativos es especialmente útil en los casos en los que el usuario desea darle más prioridad a ciertos enlaces sobre otros. Mediante el uso de la Afinidad, le da al usuario la posibilidad de crear grupos de enlaces que se diferencien unos a otros de acuerdo al tipo de tráfico que pasa a través de ellos.

NET-TE ofrece también un **mecanismo automático**, mediante el cual, tras ingresar determinados parámetros obligatorios comienza a iterar hasta descubrir completamente la red. Para NET-TE, el cargar la topología se entiende como descubrir todos los routers presentes en la red que estén intercambiando información de ruteo mediante el protocolo OSPF. Los parámetros obligatorios a ingresar son la dirección IP de algún router de la red (en general es la del router a la que la computadora está directamente conectada), la versión SNMP que se desea ejecutar (NET-TE ofrece las versiones 1 y 2) y el "community" o password usado por SNMP para permitir sólo el acceso al router a personas con permiso. También se ofrecen parámetros opcionales, que el usuario es libre de modificar, como el puerto (NET-TE usa el 161 por defecto), número de reintentos y timeout. Una vez cargada la topología en forma automática, el usuario vuelve a ser libre de poder modificarla a su gusto, tal como lo hace si la cargara manualmente.

3.2 ESTABLECIMIENTO DE LOS LSPs

En MPLS el tráfico para determinada demanda sigue determinados LSPs desde que entra a la red MPLS hasta que sale. Por eso es tan importante el establecimiento de los LSPs, cómo elegirlos y dónde ubicarlos de manera de cubrir de la mejor manera posible las demandas. NET-TE tiene por ende cuatro distintos mecanismos para ofrecer al usuario, al momento de elegir cómo y por dónde ubicar a los LSPs.

3.2.1 Ruteo Explícito

Se le ofrece al usuario una ventana en la cual, a partir de la elección del nodo de origen, se le van desplegando los posibles enlaces para que pueda ir creando salto a salto el LSP de manera explícita de origen a fin. En NET-TE, la demanda se expresa en términos del ancho de banda.

Esta es una manera que como vemos no utiliza algoritmo alguno, sino que sólo se basa en la decisión que tome el

usuario y depende exclusivamente del camino que éste desee.

3.2.2 CSPF

NET-TE le muestra al usuario cuales son todos los posibles caminos por los cuales puede rutear su tráfico, asegurándose que cumplan con el BW solicitado por el cliente, además de un conjunto pre-definido de restricciones que puede él mismo ingresar. Finalmente, será decisión del usuario el elegir el camino que más le convenga, dentro de toda la gama de soluciones.

Dentro de los parámetros obligatorios a ingresarse se encuentran el nodo de origen, el nodo destino y el BW requerido por el cliente. En caso que se desee buscar soluciones sólo por aquellos enlaces que soportan cierto tipo de tráfico se incorporó al NET-TE la posibilidad de elegir la Afinidad, como parámetro opcional. Se incorporaron también otros dos parámetros opcionales a elegir, que son: Enlace Presente y Enlace Ausente. Los mismos le aseguran al cliente que las soluciones a mostrar (en el caso que existan) cumplirán con estas restricciones.

Ahora bien, ya que el CSPF se basa en el algoritmo Dijkstra, se debe determinar cuál es la métrica a usar para elegir el camino "más corto". NET-TE ofrece 4 diferentes tipos de pesos a asignar a los enlaces: Ruteo Mínimo por Pesos Administrativos, Ruteo por Mínima Cantidad de Saltos, 1/(BWreservado) y 1/(BWlibre).

El primero de todos es básicamente basarse en los pesos que fueron pre-definidos por el usuario para cada enlace. El segundo, es simplemente establecer la cantidad de saltos como la métrica elegida. La tercera, tal como lo indica su nombre, usa pesos que equivalen al inverso del BWreservado en cada enlace (Ver [4]). Es una manera de procurar seguir usando los enlaces que ya están siendo más utilizados por otros LSPs, y no tocar los que están más libres. Ahora, supongamos que el usuario quiere que el nuevo LSP a crearse tienda a pasar por aquellos enlaces que están más libres en la red, no tocando aquellos que ya tienen recursos consumidos o LSPs pasando por ellos; para ello NET-TE incorporó el cuarto peso.

Por último, NET-TE le ofrece al usuario la posibilidad de usar o no determinado criterio de TE sobre los caminos encontrados: Elegir manualmente, No saturación enlaces críticos y Minhop.

El primer criterio simplemente le indica a NET-TE que despliegue todas las soluciones posibles que encontró Dijkstra, tomando en cuenta las restricciones y pesos seleccionados. Con el segundo criterio, NET-TE lo que hace es comparar las soluciones encontradas y se queda sólo con aquella cuyos enlaces que la componen, tienen el mayor ancho de banda disponible. Finalmente, el último criterio de TE es el de Minhop, el cual, tal como su nombre lo indica, de todas las soluciones encontradas, se queda sólo con aquellas que tienen la menor cantidad de saltos de origen a destino.

3.2.3 MIRA

La idea básica de este tipo de algoritmos es la de minimizar la interferencia que provoca el establecimiento de un nuevo LSP a potenciales nuevos LSPs que son desconocidos de modo de reservar recursos para demandas a las que considero más importantes.

Este es un algoritmo que tiene como entrada los puntos (localidades, nodos, etc.) que el usuario considera críticos y mapea así los caminos de Internet por aquellos enlaces que no sean críticos para las demandas a las que se les quiere dar prioridad.

Con NET-TE, el usuario podrá cargar todas las demandas que corresponden a caminos de acceso a Internet y mapearlas en la red corriendo el algoritmo MIRA.

3.2.4 Fairness

Supongamos ahora que el usuario ya no tiene que preocuparse por ver cómo colocar determinado LSP para cada demanda nueva entrante. Ahora, lo que desea es, teniendo una matriz de tráfico que contiene a todas las demandas que desean establecerse sobre la red (la cual ya puede contener viejos LSPs establecidos), ver cómo satisfacerlas a todas ellas de una manera lo más "justa" posible. Básicamente la pregunta que nos estaríamos haciendo es la siguiente: ¿Cuántos recursos me puede ofrecer la red en su estado actual, para cada uno de los caminos que satisfacen las demandas de los clientes?

Así, en un principio usamos al CSPF para encontrar todos aquellos caminos posibles que cumplen con la restricción del ancho de banda. Para todas las demandas, el objetivo es ver cómo se asignan los recursos de la red a cada una de ellas, de manera que se obtenga el máximo aprovechamiento posible de la red.

NET-TE brinda cuatro algoritmos a usar distintos: fairness básico, acotado, con múltiples caminos y con múltiples caminos acotado.

El primero y el tercero brindan información sobre cuál es la cantidad máxima de BW que puedo tomar de la red para cada una de las demandas. La única diferencia es que el primer algoritmo sólo me considera un camino solución fijo para cada demanda, mientras que el tercero toma todos los caminos solución posibles. El segundo considera también sólo un camino solución fijo por demanda, pero le brinda al usuario la opción de ingresar prioridades a cada una de ellas y el elegir una cota inferior de BW para cada demanda.

El cuarto algoritmo es similar al tercero. La diferencia es que en vez de detenerse el cálculo cuando ya no quedan más recursos que la red pueda ofrecer, se detiene cuando se llegó a cumplir con el ancho de banda requerido para cada demanda. Obviamente que en caso de saturar primero la red antes que se llegue al ancho de banda requerido, también se detendrá el cálculo.

Para el tercer y cuarto algoritmo, se tiene la posibilidad de apreciar cada una de las sub-demandas o caminos que conforman las demandas principales en forma separada, pudiendo ver cuánto ancho de banda rutean cada una de ellas, así como apreciarlas en forma gráfica.

3.3 VISUALIZACIÓN DEL ESTADO ACTUAL DE LA RED

La mencionada visualización consiste en los métodos que tiene el usuario con el software NET-TE de poder ver el estado de la red. Lo puede hacer de dos formas posibles: por medio del uso de la función Estadísticas o bien por la de Utilización.

Con tan sólo apretar un botón, al usuario se le desplegará una pantalla donde figurarán todos los nodos que conforman la red, los enlaces, así como los LSPs establecidos hasta ahora. Podrá ver las características de cada uno de ellos, así como también por cuál o tal nodo o enlace pasa cierto LSP, entre otros valores.

Al usuario también le podría interesar el poder clasificar a los enlaces de la red, de acuerdo al porcentaje de utilización que tienen y visualizar esa clasificación de una manera rápida y sencilla en pantalla. NET-TE permite realizar eso por medio de la herramienta Utilización. Con tan sólo ingresar dos valores de porcentajes, una cota inferior y otra superior, NET-TE pintará de distintos colores cada uno de los tres niveles de utilización para cada enlace.

Esta es una manera sencilla de visualizar en pantalla qué zonas de la red están más saturadas que otras, especialmente práctico para redes de gran tamaño donde ver valores numéricos no es tan intuitivo.

4. ARQUITECTURA DE SOFTWARE

Esta sección presentará la estructura en la que fue creado el software NET-TE. Se presentan los principales packages utilizados y las tareas realizadas por las clases más importantes que los conforman.

Primeramente se comentará el Package correspondiente a la topología de la red. Se pasará luego al Package Programa, encargado de la interfaz gráfica de la aplicación. Luego el Package Cargar Red, encargado de implementar la funcionalidad de cargar la topología de la red de manera automática. Finalmente se presentarán el Package CrearLSPs, el cual implementa el algoritmo CSPF y el Package MT, encargado de crear o cargar la matriz de tráfico de la red.

4.1 EL PACKAGE TOPOLOGÍA

En el package Topología se encuentran las clases con las que representamos los elementos de una red MPLS, que cuentan con los atributos necesarios para poder aplicar algoritmos de TE.

4.1.1 La clase Elemento

Esta clase define un objeto Elemento del cual heredarán los objetos LER, LSR y Link y define los atributos comunes a ellos.

4.1.2 Las clases LER y LSR

Los LERs y LSRs representan dentro de nuestra arquitectura de red un elemento de poca complejidad, ya que cada instancia de ellos sólo lleva registros de los enlaces que en él convergen y los LSPs que por él pasan. Aunque las clases LER y LSR son muy similares, al tomar nuestro proyecto como referencia las redes MPLS es que se decidió crear dos clases por separado y no una sola, logrando por ejemplo que la representación en la interfaz gráfica sea distinta.

4.1.3 La clase Link

La clase *Link* es el elemento fundamental de nuestra arquitectura de red. Esta clase concentra la mayoría de los atributos necesarios para hacer TE como ser capacidad, capacidad reservada, pesos administrativos, afinidades, etc. Cada instancia de *Link* representa una conexión bidireccional entre dos instancias de *LER o LSR* que recibe en su construcción y lleva un registro de los LSPs que por él pasan. La clase *Link* cuenta con métodos que permiten determinar si es un posible candidato para la creación de un nuevo LSP basado en el ancho de banda que tiene disponible mediante el ruteo explícito o corriendo alguno de los algoritmos desarrollados en el software.

4.1.4 La clase LSP

La clase *LSP* modela los caminos virtuales, que se pueden establecer en las redes MPLS y que permiten dirigir un flujo agregado de tráfico. Cada LSP consta de un vector de links, a la cual se le agrega el *Nodo* desde el cual se origina el LSP y el ancho de banda requerido, atributos necesarios a la hora de correr los distintos algoritmos de **SNMP** que en el software se implementan.

4.2 EL PACKAGE PROGRAMA

Las clases que contiene el package Programa definen las funcionalidades de la Interfaz de Usuario, y concentra todas las tareas referentes al despliegue del programa a través de una interfaz gráfica mediante el uso de elementos como ser ventanas, menús, diálogos, etc.

4.2.1 La clase Principal y VentanaConf

La clase Principal genera el marco principal que contiene las barras de menú y herramientas con el que se pueden crear, abrir y modificar distintas topologías de red.

La clase VentanaConf implementa los dos paneles principales del software que son el panel gráfico en el que se representan los elementos que componen la red y el panel que contiene los botones que implementan todas las funcionalidades del software como ser, crear elementos de red, ver estadísticas o correr distintos algoritmos de ingeniería de tráfico.

4.2.2 La clase Intérprete

Esta clase permite al usuario generar un archivo de salida, que guarda en forma de texto todos los elementos de la red y sus atributos. De esta manera el usuario podrá simular varias topologías y guardar los diferentes estados de la red en un archivo .txt que luego podrá abrirlo para que la clase *Intérprete* lo convierta en elementos gráficos de una forma sencilla.

4.2.3 La clase Cargar Topología

Esta clase implementa el marco en el cual el usuario podrá ingresar los parámetros necesarios para poder "descubrir" la topología en estudio.

4.2.4 Las clases Estadísticas y Propiedades

Estas clases son las que permiten al usuario conocer los atributos y estados de los elementos que componen la red en estudio. La clase propiedades es una ventana que muestra los atributos configurables de cada elemento de la red así como por ejemplo los LSPs que pasan por el elemento, o en el caso de los enlaces su utilización. Por otro lado la clase Estadísticas genera un marco más general en el que se puede ver el estado actual de la red como ser los atributos de los elementos de la red, ver los enlaces críticos y ver detalles de los LSPs establecidos en la red.

4.2.5 La clase Utilización

La clase genera un marco al que el usuario puede ingresar valores que se toman como referencia para definir rangos de utilización. Luego se llama a un método que recorre todos los enlaces de la red y los pinta de un color que depende de la utilización actual del enlace.

Veamos a continuación, en la Figura 4.1, el diagrama de clases del Package Programa.

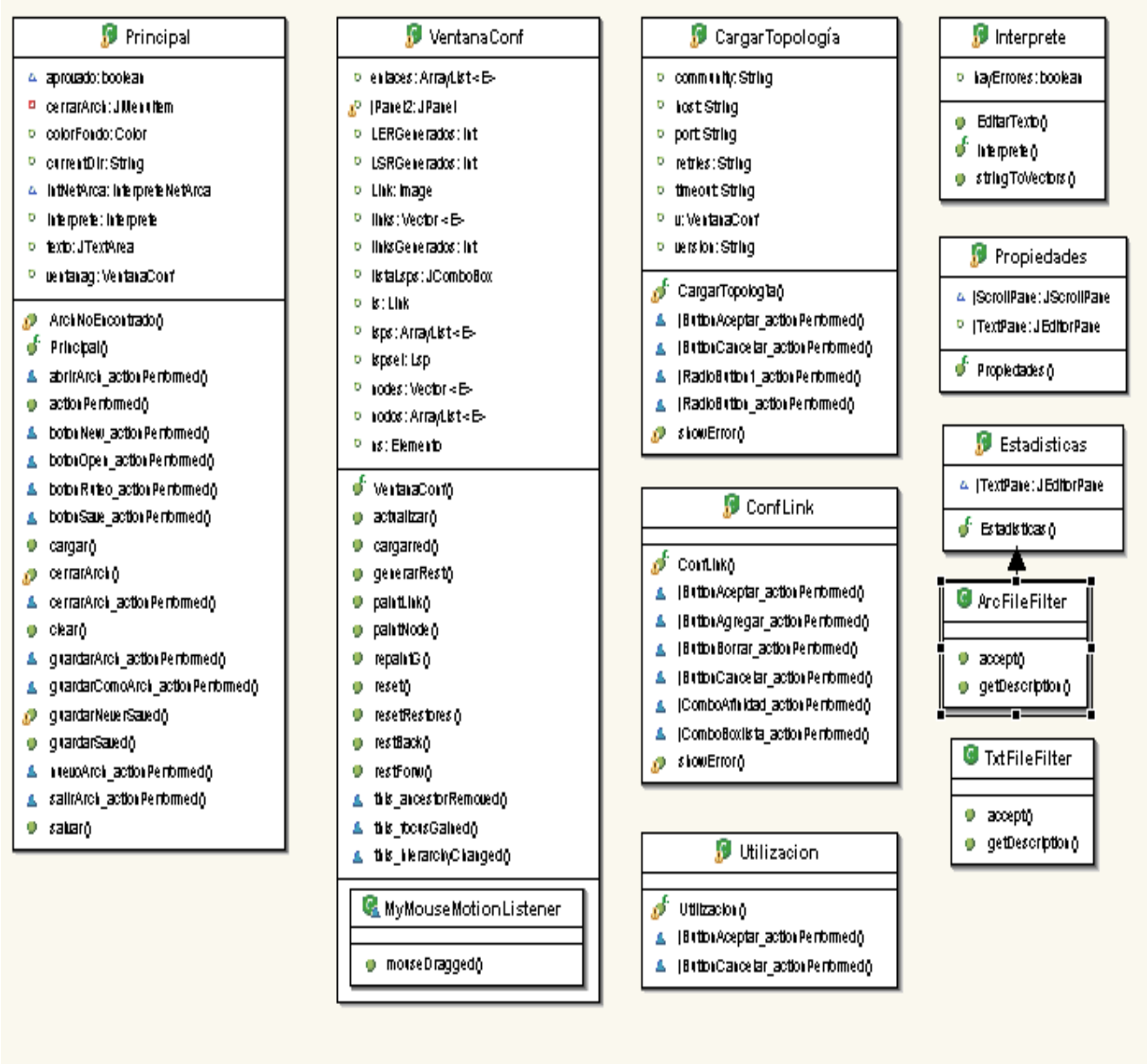

Figura 4.1: Diagrama de clases del Package Programa.

4.3 EL PACKAGE CARGAR RED

Este package es el encargado de la funcionalidad de extraer la información necesaria de los routers para poder cargar la topología automáticamente, y no de manera manual tal como lo podría hacer el usuario al crear una topología desde cero o cargar una ya existente.

4.4 EL PACKAGE CREARLSPs

Este package esta conformado por 5 clases: Ruteo Explícito, CSPF, Algoritmo, CarConf y MIRA.

4.4.1 La clase Ruteo Explícito

Esta es la clase usada cuando el usuario de la red desea establecer los LSPs de manera manual, señalando enlace a enlace, cuál es el camino que desea conforme el LSP.

4.4.2 La clase CSPF

Esta clase lo que básicamente hace es recolectar toda la serie de parámetros ingresados por el usuario, necesarios para determinar cuáles son las restricciones que desea cumpla el o los LSPs solución y qué criterio de ingeniería de tráfico desea usar. Posteriormente, de acuerdo a las opciones elegidas, pasa a llamar a la clase Algoritmo que es donde reside el método que implementa el Dijkstra y los demás métodos encargados de ejecutar el criterio de ingeniería de tráfico, si es que se eligió alguno.

4.4.3 La clase Algoritmo

En esta clase se encuentra la implementación del algoritmo Dijkstra y los de TE**.**

En cuanto al algoritmo de Dijkstra, éste es implementado por medio del método *shortestPath(),* el cual es encargado de calcular todos los caminos solución del origen a destino, con la métrica más chica.

Otro método a destacar es el *podar2()*. Este es encargado de aplicar el segundo criterio de TE (no saturación de enlaces críticos). Básicamente lo que procura es descartar a aquellas soluciones que poseen los enlaces más críticos.

Por último, se debe destacar el método *podar3(),* el cual se encarga de filtrar también las soluciones arrojadas por el Dijkstra, quedándose sólo con las más cortas en cuanto a lo que número de saltos se refiere.

4.5 EL PACKAGE MT

El package MT concentra todas las tareas referentes a calcular los mejores caminos posibles para la topología de red cargada y una matriz de tráfico dada.

En este package se encuentran las clases con las que representamos los caminos encontrados para cada ruta y cuenta con los atributos necesarios para poder asignar el ancho de banda a cada camino de forma justa. La ruta indica entre qué nodos, qué ancho de banda y qué afinidad deberá tener el LSP que se quiere establecer y el camino indica por los enlaces deberá pasar el LSP.

4.5.1 La clase Caminos

La clase *Caminos*, modela el camino virtual que se puede establecer para una ruta dada.

4.5.2 La clase InterpreteMT

Esta clase permite al usuario generar un archivo de salida, que guarda en forma de texto una lista de rutas con sus atributos que representan la matriz de tráfico. De esta manera el usuario podrá simular para una topología dada y posibles variantes de ésta, como quedan los LSPs para la misma matriz de tráfico.

4.5.3 La clase GeneroMT

Esta clase implementa el marco en el cual el usuario podrá crear el archivo que representa la matriz de tráfico. Al crear cada ruta se debe configurar sus atributos como ser el *nombre* (tiene por defecto *Ruta* seguido de un número que se incrementa al crear una ruta nueva), *ancho de banda deseado*, *nodo origen*, *nodo destino* y *afinidad* (es opcional, el camino sólo puede pasar por los enlaces con esa afinidad).

4.5.4 La clase FairnessNetwork

Esta clase permite al usuario correr los distintos algoritmos Fairness, comparar sus resultados para después crear los LSPs para las distintas rutas. Primero se llama al método que carga el archivo de matriz de tráfico y se llama al método *shortestPath()* de la clase *Algoritmo* que calcula todos caminos posibles con la menor métrica para las distintas rutas de la matriz de tráfico.

Después de calcular todos los caminos posibles para cada ruta se elige uno de los cuatro algoritmos Fairness. Se selecciona la métrica a utilizar y se calcula el ancho de banda final de cada camino. En caso de elegir uno de los dos primeros algoritmos, si existen varios caminos posibles para cada demanda, se le ofrece la posibilidad al usuario de elegir el que desee.

4.5.5 La clase OfflineMira

Esta clase implementa el marco en el cual el usuario podrá correr el algoritmo MIRA para una matriz de tráfico previamente configurada y guardada en un archivo .txt. Luego de haber cargado la matriz de tráfico el usuario tiene que elegir cuáles son los pares de nodos que se van a tomar como referencia como los futuros generadores de LSPs. El usuario puede optar entre tres opciones: todos los nodos de la red, sólo los LERs o elegirlos manualmente. Paso siguiente se corre el algoritmo.

5. CONCLUSIONES

Presentaremos a continuación los supuestos que se plantearon en el proyecto, enumeraremos los objetivos más destacados y cerraremos la sección con las conclusiones.

5.1 Supuestos y Objetivos

El único supuesto que podemos considerar importante al momento de instalar el software, es que se debe hacer sobre una red MPLS cuyos routers manejen el protocolo de ruteo OSPF. Este supuesto es necesario en caso de desear utilizar NET-TE para levantar la red en forma automática y desplegarla en pantalla.

La hipótesis sobre la red MPLS no afecta ninguna otra funcionalidad del software, ya que es principalmente una herramienta de trabajo offline, usada por el usuario para analizar distintos métodos de búsqueda de caminos para el establecimiento de LSPs.

Llegado el final de este proyecto y mirando hacia atrás, podemos ahora distinguir claramente cuatro distintos objetivos. El primero fue el desarrollar una herramienta que permita la creación de una red, visualizar su estado actual y poder modificarla de acuerdo a las necesidades del usuario. El segundo fue el ofrecer varios algoritmos que permitieran al usuario distintas posibilidades de por dónde rutear el LSP que cubrirá a cada nueva demanda entrante a la red, de manera que se ajuste de la mejor manera a sus necesidades. Como tercer objetivo se tuvo el implementar otros algoritmos que permitan, a diferencia de los anteriores, el poder considerar todas las demandas de la red en forma conjunta y ubicarlas de la mejor manera posible en la red, de tal manera que se vean todas cubiertas, o en caso de no ser posible, el cubrirlas de la manera más "justa" a todas ellas. Finalmente, como cuarto y último objetivo, se planteó la incorporación de una funcionalidad que le permitiera al usuario el poder levantar la red a la que está conectado de manera automática y visualizarla en pantalla.

Con respecto a la validación de los datos que son devueltos al aplicar los distintos algoritmos de Fairness, se comprobó su correcto funcionamiento basándose en distintas pruebas y ensayos realizados por nuestra parte. También se comparó con ejemplos de los cuales ya se conocían de manera confiable los resultados y viendo así que obteníamos los valores esperados.

Finalmente, en cuanto a la escalabilidad de los algoritmos a medida que la red crece de tamaño, pudimos corroborar en la práctica, que el tercer y cuarto algoritmo de fairness son los que consumen más tiempo de operación debido a la gran cantidad de iteraciones que se realizan. De cualquier manera, probamos con topologías de 30 nodos aproximadamente, y vimos que el tiempo que demora en desplegar los resultados es prácticamente al instante y no considerable (poco más de un par de segundos cuando máximo).

5.2 Conclusiones

Se logró implementar una interfaz gráfica sencilla y fácil de usar por cualquier usuario, la cual le otorga una amplia flexibilidad al momento de diseñar o modificar su red.

Además, se le brindó al usuario varios mecanismos para poder visualizar el estado actual de la red, pudiendo ver información de elementos en particular o de la red en general.

El problema del establecimiento de los LSPs fue solucionado utilizando el algoritmo CSPF. Realizamos varias pruebas sobre distintas topologías y situaciones, y pudimos apreciar el correcto funcionamiento del mismo.

También pudimos comprobar la gran utilidad que ofrece el algoritmo MIRA. Se hicieron pruebas de chequeo sobre topologías en las cuales había claramente ciertos enlaces que eran críticos para futuras posibles demandas, y pudimos apreciar como el algoritmo tendía a evitar dichos enlaces críticos.

En cuanto a los algoritmos de fairness pudimos apreciar con éxito como se lograba ubicar esa demanda sobre la red, cosa imposible de realizar usando el CSPF. Además, comprobamos con éxito como NET-TE lograba asignar los recursos de una manera justa en aquellos casos donde no era posible satisfacer por completo a todas las demandas.

Con respecto a la carga automática de la red, se pudo comprobar con éxito su funcionamiento en una red multiservicio desde el IIE de la Facultad

Es por todo lo anterior que creemos que NET-TE es una útil herramienta que puede ser usada por cualquier usuario que desee realizar pruebas y crear distintos escenarios de Ingeniería de Tráfico, sin temor de afectar de alguna manera la red real, ya que se está en todo momento trabajando sobre escenarios de prueba y no sobre la red en cuestión.

6. REFERENCIAS

[1] **NetScope**: Traffic Engineering for IP Networks. Paper de AT&T Labs. Autores: Anja Feldmann, Albert Greenberg, Carsten Lund, Nick Reingold y Jennifer Rexford. Marzo 2000.

[2] **Request for Comments (RFC): 3272.** Visión y Principios de la Ingeniería de Tráfico en Internet. Autores: D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, X. Xiao. Mayo 2002.

[3] **MPLS: Technology and Applications**. Capítulo 7: Constraint-Based Routing. Autores: Bruce Davie y Yakov Rekhter. Año 2000.

[4] **Data Networks**: Routing, Security, and Performance Optimization. Capítulo 8: Quality of Service. Autor: Tony Kenyon. Año 2002.

[5] **Minimum interference routing with applications to MPLS traffic engineering**. Proceedings of International Workshop on QoS. Pennsylvania. Autores: M. Kodialam y T.V.Lacksham. Junio 2000.

[6] **A New Bandwidth Guaranteed Routing Algorithm for MPLS Traffic Engineering**. Autores: Bin Wang, Xu Su y C.L. Philip Chen. Noviembre 2002.

[7] **Routing, Flow and Capacity Design in Communication and Computer Networks**. Capítulo 8: Fair Networks. Autores: Michal Pioro y Deepankar Medhi. Año 2004.