Prefacio

El presente documento constituye la documentación final del Proyecto de Fin de Carrera titulado "Ingeniería de Tráfico y Calidad de Servicio en Redes MPLS", para el Instituto de Ingeniería Eléctrica de la Facultad de Ingeniería, Universidad de la República. Los integrantes del grupo de trabajo son Darío Buschiazzo Malveira, Andrés Ferragut Varela y Alejandro Vázquez Paganini. Con este trabajo los autores obtienen el título de Ingeniero Electricista, opción Telecomunicaciones.
El proyecto se llevó a cabo en el período comprendido entre el mes de Junio de 2003 y el mes de Mayo de 2004 en Montevideo, República Oriental de Uruguay, bajo la tutoría del Ing. Pablo Belzarena.
El presente proyecto analiza el problema de brindar garantías de Calidad de Servicio (QoS), así como realizar Ingeniería de Tráfico sobre redes de datos. Se desarrolló una herramienta de software que permite relevar el funcionamiento de una red dada, estimando parámetros de QoS permitiendo predecir el comportamiento de la red en presencia de carga y evaluar el impacto de diferentes políticas en dicho funcionamiento, así como conocer las garantías de QoS que la red está en condiciones ofrecer.
El trabajo se divide en cuatro partes, cada una de las cuales procederemos a describir brevemente. En la primera parte se presentan las técnicas actuales de ingeniería de tráfico y calidad de servicio, como ser las herramientas teóricas para el análisis de performance de redes y las diferentes tecnologías disponibles para su implementación. En la segunda parte se profundiza en aquellas herramientas que se consideraron apropiadas para cumplir con los objetivos trazados. A continuación, en la tercera parte se describe la implementación en software de los algoritmos y técnicas discutidas en la parte anterior. En la cuarta y última parte, se analizan los resultados obtenidos contrastando los mismos con simulaciones. Se describen también algunas aplicaciones de la herramienta desarrollada y se realizan las conclusiones finales del proyecto.
Con la presente documentación se adjunta un disco compacto conteniendo:

- El instalador del software que se encarga de dejar el programa listo para ser utilizado.
- Documentación del código de software (JavaDoc).
- Archivos de ejemplo que permitan familiarizarse con el programa.
- Un archivo pdf con este documento.

Agradecimientos

- A nuestras familias, por el apoyo brindado a lo largo de toda la carrera.
- A Alejandra y Natalia, por su aliento y apoyo durante todo el proyecto.
- A nuestro tutor, Pablo Belzarena, quién nos acercó un proyecto de trabajo interesante y aportó su experiencia para ayudarnos en nuestra tarea.
- A los integrantes del grupo ARTES, a quienes debemos buena parte de la discusión y entendimiento de las herramientas teóricas.
- Al Instituto de Matemática y Estadística Rafael Laguardia.
- A otras personas consultadas, en especial:
  - Al Ing. Alfredo Piria, por su guía y aportes en los temas de optimización
  - Al Dr. Ravi Mazumdar, por su gentileza en ayudar a la comprensión de las herramientas teóricas principales en las que se basa el proyecto.
  - A Mauricio García, Gastón Madruga y Victor Paladino, cuyo proyecto de fin de carrera sobre simuladoción de redes fue consultado para la realización del presente proyecto.

Resumen Ejecutivo

La principal motivación del presente proyecto consiste en realizar un estudio de la performance de redes de datos actuales, haciendo especial hincapié sobre aquellas arquitecturas de mayor proyección a futuro. Es por esto que nuestro enfoque se orienta a redes de caminos virtuales (en particular MPLS), las cuales han cobrado gran relevancia en los últimos años.

Tomando como base el estudio de la performance de una red, es posible dar un salto cualitativo que permita implementar técnicas orientadas a brindar Calidad de Servicio (QoS) mediante la utilización de herramientas de Ingeniería de Tráfico.

Este proyecto pretende acompañar la actual migración hacia redes que buscan adaptarse a los nuevos requerimientos de los usuarios, desde una primera aproximación teórica a dicho problema. En definitiva, el proyecto busca proporcionar una herramienta que desde la base teórica antes mencionada, permita asistir en el diseño de las nuevas redes. Pensando a más largo plazo, el aporte de este trabajo de investigación pretende ser un primer paso hacia la posibilidad de ofrecer herramientas para practicar Ingeniería de Tráfico en tiempo real.

Dado que no es posible pensar en brindar Calidad de Servicio sin considerar la red como un todo, resulta indispensable, contar con herramientas que proporcionen resultados en todos los puntos de la misma. La teoría de colas tradicional prueba ser eficaz en el análisis el borde de la red, pero carece de generalidad, en lo que a modelos de tráfico se refiere, si se pretende obtener resultados teóricos en el interior de la misma. Es por esto que nuestro trabajo se basa en la teoría de Grandes Desviaciones, que permite proporcionar resultados teóricos de gran generalidad (se admite una mayor variedad de modelos de tráfico) y en todos los puntos de una red, incluso en sus enlaces interiores.

Como resultado de la teoría utilizada se obtienen estimaciones de la *probabilidad de overflow* en cualquier enlace de la red lo que constituye un índice de performance a partir del cual es posible implementar Ingeniería de Tráfico. Para un enlace cualquiera, la probabilidad de overflow se puede interpretar como el porcentaje de tiempo que el mismo permanece saturado por el tráfico que cursa, con la correspondiente pérdida de información que esto conlleva.

Para esto se desarrolló una herramienta de software que a partir de mediciones del tráfico que ingresa a la red en estudio y la topología de la misma, permite obtener una estimación asintótica de la probabilidad de overflow en un enlace cualquiera de la red. El hecho que la estimación sea asintótica implica que los resultados adquieren mayor precisión a medida que aumenta significativamente el número de fuentes generadoras de tráfico que ingresa a la red, lo que hace que esta herramienta sea idónea para redes de gran porte.

Otra opción para obtener resultados a cerca de la performance en el interior de una red es la de realizar simulaciones de las mismas. Esto, para redes de gran porte, demanda una gran inversión de tiempo lo que lo hace poco atractivo a la hora del diseño de una red, donde siempre es necesario variar una gran cantidad de parámetros lo que

implica realizar una nueva simulación cada vez. Tampoco las simulaciones ofrecen una alternativa viable para la implementación de ingeniería de tráfico en tiempo real por las mismas razones antes esgrimidas.

Lo anterior constituye una apreciable ventaja de nuestra herramienta frente a la alternativa de las simulaciones, ya que los tiempos de cálculo insumidos son significativamente menores, y que generalmente rondan unos pocos segundos.

Para probar la veracidad de los resultados obtenidos, se contrastaron las estimaciones proporcionadas por la herramienta contra simulaciones llevadas a cabo mediante la utilización del difundido simulador de la Universidad de Berkley, el NS-2 (Network Simulator).

Los resultados obtenidos, si bien carecen de la precisión alcanzable mediante simulaciones, proporcionan una buena aproximación al comportamiento real de las redes estudiadas con una excelente relación precisión-tiempo de cálculo. Estos resultados dejan abierta la posibilidad de pensar en implementar técnicas de ingeniería de tráfico en tiempo real, como por ejemplo control de admisión de conexiones.

Índice general

Parte I

Presentación del Problema

Capítulo 1

Introducción

En el presente capítulo se presentan los objetivos planteados al inicio del proyecto. A su vez, se introducen los conceptos elementales en los que se funda este trabajo, y los motivos que llevan a su realización. Por último, se realiza un resumen de cómo se presentan en este documento las diferentes tareas llevadas a cabo.

1.1. Objetivos del proyecto

El objetivo general del presente proyecto es desarrollar herramientas que permitan el análisis de performance de redes de datos. En particular, se pretende analizar el problema de brindar garantías de Calidad de Servicio (QoS), así como realizar Ingeniería de Tráfico sobre dichas redes.

Se propone desarrollar una herramienta de software que permita relevar el funcionamiento de una red dada, ajustando modelos de tráfico y estimando parámetros de QoS. En base a ello, predecir el comportamiento de la red en presencia de carga y evaluar el impacto de diferentes políticas en dicho funcionamiento, así como conocer las garantías que la red puede ofrecer.

El primer objetivo específico es desarrollar una base teórica a partir de resultados existentes en el área, para sustentar el posterior desarrollo de la herramienta de software. Dichos resultados abarcan desde estimadores de magnitudes relevantes, modelos para distintos tipos de tráfico y resultados teóricos sobre la performance de redes.

Otro objetivo específico es generar una librería de software reutilizable que provea funciones que permitan construir una topología de red genérica, a los efectos de realizar cálculos sobre la misma.

Un tercer objetivo consiste en construir a partir de dichas librerías una plataforma de análisis de redes que brinde resultados en cuanto a la performance de las mismas. Esto posibilita tomar decisiones relativas al diseño de la red, a los efectos de mejorar su desempeño. En particular, estimar las garantías de servicio que la red en estudio está en condiciones de ofrecer a sus usuarios, así como identificar problemas de diseño, sobrecargas y restricciones de servicio que puedan aparecer.

La realización de este proyecto fue promovida por el Ing. Pablo Belzarena en el marco del grupo ARTES de investigación conjunta entre el Instituto de Ingeniería Eléctrica (IIE) y el Instituto de Matemática y Estadística Rafael Laguardia (IMERL) que actualmente está dedicado al análisis teórico de estos temas. Este proyecto se enmarca en un proyecto más amplio que lleva a cabo este grupo con respaldo de la DINACYT.

1.2. Motivación

La convergencia de las diferentes redes de telecomunicaciones es una vieja aspiración de los diferentes actores del mundo de las comunicaciones. La aparición de nuevos y cada vez más variados servicios hace de esto un desafío cada vez mayor. Actualmente se busca ofrecer, sobre la misma infraestructura básica, servicios de transferencia de archivos, correo electrónico, envío de texto, mensajería, aplicaciones compartidas, telefonía, videoconferencia, audio y video a pedido, entre otros.

Los ejemplos antes mencionados dejan entrever que, para poder ofrecer diferentes servicios sobre una misma red, será necesario poder determinar los diferentes requerimientos que cada uno de ellos pueda tener. A modo de ejemplo, los servicios de transferencia de archivos son extremadamente sensibles a la pérdida o corrupción de los datos. En cambio, los servicios de *tiempo real* como son la telefonía y la videoconferencia son sensibles principalmente al retardo (y sus variaciones), aunque una alta tasa de pérdida también afecta la calidad de los datos enviados.

Lo anterior conduce al concepto de *Calidad de Servicio* (QoS), que abarca todas las formas en que una red afecta la transmisión de datos. Es decir, cada flujo de datos que atraviesa la red sufre de pérdidas, retardo, variaciones de retardo (jitter), pérdida del orden de paquetes, etc. A su vez, es sensible de forma distinta a los diferentes problemas, por lo que requerirán distintos valores de estos parámetros. Garantizar Calidad de Servicio consiste en que la red tenga la posibilidad de ofrecer un camino para los datos que permita asegurar cierto comportamiento favorable al servicio que se desea ofrecer. Por ejemplo, poder garantizar un retardo de menos de 300 ms resulta indispensable para una conversación telefónica. Si además ésta se realiza sobre paquetes, una baja tasa de pérdida será necesaria para hacer que la voz resulte inteligible.

A lo largo de las últimas décadas han aparecido diferentes tecnologías que intentan proveer las herramientas para lograr el objetivo de la convergencia de servicios. La tecnología ISDN (Red Digital de Servicios Integrados), el modelo ATM (Modo de Transferencia Asíncrono) y la arquitectura MPLS (Conmutación de Etiquetas MultiProtocolo) son algunos de los modelos propuestos. Incluso el protocolo IP (Internet) que surgió como una tecnología orientada mayormente a la transferencia de archivos y texto, hoy en día da soporte a una multiplicidad de servicios para los que no fue diseñado.

En la década del 90 el grupo IETF comenzó a estructurar un modelo general, independiente de la tecnología, para describir la estructura que debe tener una red convergente, y que además brinde garantías de QoS a su tráfico. El primer modelo propuesto es el de *Servicios Integrados* (IntServ), cuya principal política es la de implementar control de acceso y reserva de recursos para cada flujo que atraviesa la red. Esto requiere que cada conexión gestione un contrato de reserva donde se establecen las garantías de servicio que la red le ofrece.

Sin embargo, el modelo IntServ no resulta escalable, debido a que requiere reserva de recursos para cada flujo. Si bien existen tecnologías como ATM y MPLS que permiten gestionar estas reservas, el hecho de que deban realizarse flujo por flujo se torna inmanejable en las zonas centrales de una red de área extensa (WAN), donde miles de clientes comparten recursos.

Por estas razones, el IETF propuso un nuevo modelo para solucionar este problema. El modelo, conocido como *Servicios Diferenciados* (DiffServ), se basa en la clasificación de los diferentes tipos de tráfico en clases diferenciadas según sus requerimientos. Esto permite trabajar con agregados de flujo, mejorando el problema de la escalabilidad. La idea es establecer caminos para estos agregados de tráfico que garanticen los parámetros de QoS que cada uno de ellos requiere, y que cada nodo clasifique los datos en dichas

clases y maneje cada una de ellas de acuerdo a políticas apropiadas. Las diferentes tecnologías que ya se han mencionado permiten de un modo u otro incorporar las ideas de DiffServ, siendo posible implementar prioridades y clasificación de flujos tanto en IP, como ATM y MPLS, que son las tecnologías predominantes.

Todo lo expuesto vuelve central el problema de estimar qué garantías de QoS está en condiciones de ofrecer una cierta red sometida a cierta configuración de carga. Esto permite decidir qué tipos de datos la red está en condiciones de transportar, en qué proporción, y cuáles caminos son los más adecuados. Esto conlleva a su vez el problema de estimar la cantidad de recursos que requiere cada tipo de tráfico.

El disponer de una herramienta que permita estimar parámetros de QoS en los nodos de una red a partir de datos del tráfico que ingresa a la misma constituye entonces una herramienta útil para los diseñadores, permitiendo evaluar el impacto de diferentes decisiones de diseño como la cantidad y capacidad de los enlaces o el tipo de política de los routers. A su vez, constituye una herramienta útil de Ingeniería de Tráfico, ya que puede utilizarse para el control de aceptación de nuevos flujos, evaluando el impacto que traerán los mismos a la red; o bien en la realización de ruteo y balance de carga. Por último, es de interés que estas herramientas estén basadas en mediciones directas sobre el tráfico y no en simulaciones, debido a que esto permite también la implementación de políticas de ingeniería de tráfico de la red en línea, estimando el comportamiento de los flujos a medida que estos llegan a la red.

En este proyecto se ataca precisamente este problema, buscando establecer un índice de performance relacionado con la calidad de servicio que sea posible estimar en cada uno de los nodos de la red, a partir del conocimiento de las fuentes generadoras de tráfico, que permita adoptar estrategias y tomar decisiones como las mencionadas anteriormente. A lo largo de este documento se presenta la solución propuesta, formulando específicamente las hipótesis en las que ésta es válida y los pasos seguidos para implementarla.

Para ello, en los capítulos siguientes se introducen los elementos básicos de las tecnologías que permiten implementar Ingeniería de Tráfico y Calidad de Servicio en las redes actuales, y se introducen las herramientas teóricas desarrolladas en los últimos años para el estudio de performance de dichas redes. Esta breve reseña nos permitirá concluir esta parte con un planteo detallado del problema a resolver. Las restantes partes de este documento se enfocan en la solución propiamente dicha, y en el estudio de su desempeño.

1.3. Esquema del documento

La objetivo de esta sección es la de presentar la manera en que esta organizado este documento. Se describen las diferentes tareas requeridas para la llevara cabo el mismo, así como su presentación en este documento.

En la primera etapa del proyecto, se investigaron aquellas arquitecturas de red que permiten brindar calidad de servicio a los tráficos que circulan por ellas. La intención de esta etapa es acercarse a las técnicas disponibles para elegir definir claramente las hipótesis de trabajo, así como ajustar los objetivos de diseño para aproximar la herramienta final a las aplicaciones.

Es así que se decidió estudiar la arquitectura de red MPLS, la cual provee un marco adecuado para realizar el tipo de estudios necesarios para alcanzar los objetivos. También fue necesario realizar un estudio de otras herramientas y técnicas que luego serían de utilidad. Dentro de las misma podemos mencionar los conceptos de Ingeniería de Tráfico, técnicas de selección de rutas y DiffServ entre otros. Todas estos conceptos de desarrollan en el capítulo 2, donde se realiza énfasis en aquellos que son se mayor interés.

La siguiente parte del proyecto, resumida en la parte II de este documento, constituye el análisis de todas aquellas herramientas teóricas necesarias para plantear una solución adecuada.

Luego de un análisis de las alternativas existentes, se decidió utilizar las técnicas aportadas recientemente por la Teoría de Grandes Desviaciones al estudio de performance de redes. Esto requirió comenzar el estudio de esta teoría, que se presenta en el capítulo 3. En particular se presentan en detalle las técnicas introducidas por Mazumdar et. al. en [16] que a la postre representaron la principal base teórica del proyecto. El final del capítulo 3 se dedica al análisis de la teoría expuesta en el citado artículo. Un análisis desde el punto de vista práctico de esta técnica se presenta en el capítulo 4, donde se determinan las tareas a enfrentar para llevar a la práctica las herramientas teóricas estudiadas. Dos de dichas tareas requirieron a su vez sus respectivos análisis. Es así que se dedica el capítulo 5 a todo lo relacionado con la estimación estadística de las magnitudes relevantes del tráfico, a partir de trazas del mismo. El capítulo 6 se dedica a desarrollar un algoritmo de optimización apropiado para resolver el problema central presentado en el capítulo 4, y que constituye el paso central de las estimaciones de performance.

En paralelo a lo anterior se comenzó el desarrollo de las librerías de software en que se basaría la herramienta final. La descripción de cada una de las componentes de dicha herramienta abarca la parte III del presente documento. El capítulo 7 se dedica a una descripción general del funcionamiento del software desarrollado. Los restantes capítulos se dedican a describir cada una de las componentes en detalle.

En primer lugar, fue necesario implementar una librería que permita la descripción de una red, lo más genérica posible. Incluso antes de decidir la funciones específicas de debía implementar la herramienta de software, fue necesario contar con la anterior librería, que luego serviría como base para éstas. Esto presentó un desafío no menor, ya que para poder obtener buenas estimaciones de performance es necesario tener una base robusta de librerías que permita describir las partes esenciales de una red de forma apropiada. A su vez, dicha librería debe ser capaz de describir un conjunto suficientemente amplio de redes, para poder realizar un estudio objetivo y realista de la misma. Se desarrollo una serie de clases que cumplía con los objetivos anteriores, y estaba en concordancia con la arquitectura MPLS, permitiendo describir las principales características de éste tipo de redes. Las descripción de estas librerías se presenta en el capítulo 8.

Una vez finalizada la implementación de todos los algoritmos necesarios, fue necesario generar bibliotecas de acceso al programa, que permitieran a un usuario generar una topología de red, adjuntar a la misa diversas fuentes de tráfico y por último solicitar los cálculos deseados. Para esto fue necesario decidir la manera de interactuar con el sistema de bibliotecas de cálculo. En una primera instancia se definió un lenguaje para la interacción con el programa de manera que mediante comandos sencillos sea posible realizar todas las funciones provistas por el programa. Con la intención de facilitar la interacción, se diseño una interfaz gráfica con la que el usuario define de manera sencilla la topología mediante herramientas gráficas. A su vez es posible detallar todos aquellas entidades de la red en las que se desea realizar estimaciones de performance. Tanto la implementación como las principales funciones de este conjunto de librerías, así como el manejo de los errores, se detalla en los capítulos 9 y 10.

La última parte de este documento está abocada a estudiar a presentar las conclusiones finales del proyecto.

En primer lugar se llevó a cabo la validación de los resultados que brinda la herramienta desarrollada. Para poder verificar la calidad de las estimación así como la aplicabilidad en entornos reales de redes se realizaron simulaciones mediante la herra-

mienta NS-2. El contraste entre los resultados de la herramienta de software con las simulaciones, así como las conclusiones que se desprenden de esto se presentan en el capítulo 11.

En el capítulo 12 se presentan algunos casos de interés en los que la herramienta desarrollada puede resultar de especial utilidad para llevar a cabo tareas de ingeniería de tráfico. A su vez, en el capítulo 13 se presentan algunas líneas de trabajo que escapan al alcance de este proyecto, pero que resultan de particular interés y ameritan ser profundizadas en el futuro.

Por último, en el capítulo 14 se realizan las conclusiones finales del proyecto.

Capítulo 2

Ingeniería de Tráfico

La Ingeniería de Tráfico (TE) es una disciplina que ha adquirido gran relevancia en el mundo de las telecomunicaciones en los últimos años. Ésta procura la optimización de la performance de las redes operativas. En general, se nutre de la aplicación de tecnologías y principios científicos para el modelado, dimensionamiento, caracterización y control del tráfico que circula por la red, para luego aplicar este conocimiento con el fin de alcanzar diversos objetivos de performance.

El objetivo último de la TE consiste en lograr una fácil y eficiente operación de las redes, basándose en un uso óptimo de los recursos de la misma.

Este capítulo, pretende progresivamente introducir los temas relacionados con la Ingeniería de Tráfico, con el objetivo de llegar a concluir qué técnicas nos serán de utilidad para desarrollar el proyecto. Es así que en la sección 2.1, se presentan los conceptos más generales de esta disciplina, con el objetivo de que el lector se familiarice con los mismos. Esta sección define qué se entenderá por red de aquí en más, cuáles son los orígenes de la TE y cuáles son sus principales objetivos. En la sección 2.2, se describen las técnicas que en la actualidad, son las más utilizadas para implementar TE y cuáles son sus principales falencias. Posteriormente, se brinda una descripción de la arquitectura MPLS, con la finalidad de explicar su funcionamiento, detallando sus ventajas frente a las opciones tradicionales, que llevan a su elección a la hora de implementar TE. En la sección 2.3 se realizan los comentarios finales.

2.1. Generalidades

Cuando se debe lidiar con el crecimiento y la expansión de las redes de datos, existen dos alternativas posibles a utilizar, la ingeniería de red y la ingeniería de tráfico.

La ingeniería de red consiste en manipular la red para adaptarse al tráfico. Los diseñadores deben hacer las mejores predicciones posibles sobre el tráfico que sus redes deberán cursar, para luego construir la red de manera que satisfaga las necesidades de dicho tráfico. Generalmente este tipo de ingeniería se realiza en el mediano o largo plazo (meses/años), principalmente debido a que la instalación de nuevos circuitos o equipos, es una tarea que insume mucho tiempo.

Por otro lado, la ingeniería de tráfico se ocupa de manipular el tráfico para adaptarse a la red. A pesar del tipo de predicciones que se realicen, el tráfico que la red debe aceptar, no coincidirá con las predicciones en un 100 %. En algunos casos, por ejemplo como se dio a mediados de los años 90, el crecimiento del tráfico excede todo tipo de predicciones, y no es posible lograr que las redes se adapten al mismo paso. También existen ocasiones donde algún evento en particular (un espectáculo deportivo, un sitio

web de gran popularidad, etc.) generan mucho tráfico hacia lugares donde normalmente no lo hay. A su vez, debido a la concentración del tráfico a la salida de los países o de zonas densamente pobladas, algún problema con algún enlace de gran importancia genera grandes congestiones sobre el resto de los enlaces en funcionamiento.

Por otro lado, así como grandes crecimientos de las redes, eventos de gran popularidad o problemas importantes pueden ser causantes de gran demanda de capacidad en algún lugar, es muy común tener enlaces en las redes que se encuentren la mayor parte del tiempo subutilizados. La ingeniería de tráfico es, en su esencia, el arte de mover al tráfico de una determinada red a diversos lugares, con el objetivo de lograr que el tráfico de algún enlace muy congestionado se pase a cursar por otro que se encuentre subutilizado.

Para cumplir con los lineamientos antes mencionados, la ingeniería de tráfico se subdivide en dos ramas principalmente diferenciadas por sus objetivos:

- **Orientada a tráfico**. Esta rama tiene como prioridad la mejora de los indicadores relativos al transporte de datos, como por ejemplo: minimizar la pérdida de paquetes, minimizar el retardo, maximizar el "throughput", obtener distintos niveles de acuerdo para brindar calidad de servicio, etc.

- **Orientada a recursos**. Esta rama se plantea como objetivo, la optimización de la utilización de los recursos de la red, de manera que, no se saturen partes de la red mientras otras permanecen subutilizadas, tomando principalmente el ancho de banda como el recurso a optimizar.

Ambas ramas convergen en un único objetivo global, que es minimizar la congestión, la cual generalmente es atribuible a dos fenómenos bien distintos y cuyas soluciones también lo son. El primero de ellos es la insuficiencia de recursos en la red la cual tiene como alternativa a la obvia solución de aumentar los recursos, el control de congestión. La otra causa de congestión es el pobre mapeado del tráfico a los recursos, el cual para su solución necesita de técnicas de ingeniería de tráfico, como por ejemplo, balance de carga y una correcta distribución de recursos, etc.

Antes de profundizar en las diferentes técnicas de TE se hace imprescindible definir qué se entiende por una red. En su forma mas abstracta una red consiste simplemente en un conjunto de *nodos*, interconectados por *enlaces*. Estos elementos constituyen la topología básica de una red y definen los caminos que pueden seguir los datos sobre la misma.

Si nos referimos a redes de datos, los nodos serían por ejemplo *routers*, *switches* de redes de area local, *switches* de redes de área amplia, *multiplexores add-drop*. Si se estuviera trabajando en redes de telefonía sobre IP, los nodos podrían representar *gateways de voz*, *gatekeepers*, *softswitches* entre otros elementos.

A su vez los enlaces imponen restricciones de comportamiento al tráfico que circula por ellos. Las más significativas son la capacidad y el retardo de transmisión. En los ejemplos anteriores se podrían considerar enlaces que van desde DS0 de 64Kbps a STM1 de 155Mbps o Ethernet de 10 gigabit.

2.1.1. Multiplexación

Una propiedad fundamental para todos los tipos de redes es la *multiplexación*. La misma permite que múltiples conexiones a través de una determinada red compartan los recursos de transmisión. Los dos tipos principales de multiplexación son:

- Multiplexación por división en el tiempo (TDM)
- Multiplexación estadística

TDM

La multiplexación por división en el tiempo consiste en asignar a una o varias conexiones un determinado tiempo en un circuito físico. Como los circuitos físicos generalmente tienen tasas de transmisión constante, esto se traduce en asignar un ancho de banda determinado. Un ejemplo de TDM es Jerarquía Digital Sincrónica (SDH). TDM es una tecnología sincrónica, todos los datos que entran a una red son transmitidos de acuerdo al reloj maestro, de manera que en ningún momento se dan atascos de datos esperando a ser transmitidos.

La propiedad fundamental de TDM es la asignación fija de un ancho de banda determinado para una conexión dada en todo momento. Lo anterior es una de las principales ventajas de TDM, pero a su vez puede llevar a la subutilización de la red en caso que dicho ancho de banda no sea utilizado. Por ejemplo un enlace E1 (2048 kbps) permite transmitir aproximadamente 170 gigabits en un día, por lo tanto si uno transmitiera menos de esa cantidad por día estaría pagando capacidad que no se utiliza. El compromiso surge de la disponibilidad del E1, ya que en cualquier momento que se desee utilizar el ancho de banda contratado, éste esta disponible por completo; ese es el precio a pagar.

Multiplexado estadístico

Los altos costos de TDM representan una de las principales razones por las cuales las tecnologías de multiplexado estadístico se han vuelto populares. Este tipo de multiplexación consiste en compartir ancho de banda de transmisión entre todos los usuarios de una red, sin ancho de banda garantizado para ninguno de ellos.

La mayor ventaja del multiplexado estadístico respecto de TDM es su menor costo en términos comparativos ya que es posible sobrevender más capacidad de la que la red realmente tiene, bajo la hipótesis que no todos los usuarios transmiten a su máxima tasa al mismo tiempo. Ésto permite a los usuarios acceder a servicios de red a más bajo precio, de allí su popularidad.

Existen varias tecnologías que implementan este tipo de multiplexado, pero las principales en los últimos años han sido:

- IP
- Frame Relay
- ATM
- MPLS

Todas las anteriores trabajan en base a dividir el tráfico de la red en pequeñas unidades discretas, tratando cada una de éstas por separado. En IP se llaman *paquetes*, en ATM *celdas* y en Frame Relay *tramas*. El caso de MPLS es algo diferente y se verá más adelante, ya que coexiste con alguna de la anteriores tecnologías.

La multiplexación estadística introduce una serie de elementos a tener en cuenta, que no existen en TDM. Desde el momento en que los paquetes (supongamos que se está trabajando con redes IP) entran a la red de manera asíncrona, existe la necesidad de *contención de los recursos*. En el caso que dos paquetes arriben a un router al mismo

tiempo destinados a salir por la misma interfaz, uno de ellos deberá esperar a que el otro sea procesado para poder salir.

Existe otro elemento a tener en cuenta al tratar con los paquetes que deben esperar para ser procesados. Algunas aplicaciones (como la transferencia de archivos) no presentan problemas cuando sus paquetes están a la espera en los buffers, mientras que otras aplicaciones (voz, video) sí lo hacen. Por lo tanto se deberá tomar una política de tratamiento de paquetes para poder cumplir con las demandas de los diferentes tipos de aplicaciones.

Por último, la posibilidad de que los datos se vean puestos en espera aumenta significativamente cuando la sobreasignación de recursos es grande, y en el caso que se tengan grandes tasas de transmisión es posible que los buffers se comiencen a llenar, haciendo inevitable el descarte de algunos de los paquetes.

Cada una de las tecnologías de multiplexado estadístico tiene su propia manera de tratar estos aspectos, entre los cuales podemos citar las notificaciones de congestión (FECN y BECN) de Frame Relay, los bits de DiffServ o de notificación explícita de congestión (ECN) de IP, y las diferentes clases de servicios provistas por ATM.

Multiplexado estadístico sobre multiplexado estadístico

IP esta disponible desde los principios de los años 80, mientras que Frame Relay y ATM recién aparecieron comercialmente a principios y mediados de la década del 90 respectivamente. Una de las principales complicaciones que aparecieron al reemplazar los circuitos TDM con Frame Relay o ATM fue que se estaba corriendo IP sobre éstos, o sea que se estaban implementando dos protocolos de multiplexado estadístico, uno encima del otro. Lo anterior generalmente conduce a soluciones subóptimas, ya que los mecanismos para tratar la contención de recursos en un nivel del multiplexado, generalmente no se traducen al otro.

Por lo tanto, una de dos alternativas aparecen como viables. La primera sería evitar la congestión en el protocolo de capa 2 y la segunda consistiría en mapear de alguna manera los mecanismos de contención de capa 3 a los de capa 2. Ya que no solo es imposible sino que además no es para nada atractivo desde el punto de vista financiero el evitar la congestión en la capa 2, se deberá implementar de alguna manera la segunda alternativa. Esta es una de las razones por las que MPLS esta jugando un papel preponderante en las redes hoy en día. En la secciones posteriores se verá mas información sobre esto.

Para concluir, cabe mencionar que los diferentes enfoques de multiplexado son complementarios. Las ventajas y desventajas de cada una se aplican a la solución de la telefonía tradicional, en donde conviven ambos. En lo que se conoce como *planta externa*, se aplica el multiplexado estadístico a nivel de los concentradores y en las centrales locales. Entre centrales, se utilizan conexiones TDM, con recursos reservados. Precisamente, la cantidad de recursos entre centrales está dimensionada de acuerdo a la cantidad esperable de llamadas y no pretendiendo asignar recursos fijos a todos los usuarios, lo que no sería rentable.

2.2. Ingeniería de tráfico y MPLS

2.2.1. Situación actual

La ingeniería de tráfico es totalmente independiente de la tecnología, es una práctica general a las diferentes redes existentes hoy en día. La misma puede consistir en algo tan simple como ajustar algunos parámetros de las métricas de IP en determinadas

interfaces, o algo tan complejo como implementar un algoritmo que analice la red por completo y distribuya los caminos virtuales de ATM de acuerdo a las necesidades del tráfico.

En la actualidad existen dos tecnologías de multiplexado estadístico sobre las que se realiza ingeniería de tráfico, IP y ATM.

En el caso de IP, es una práctica muy popular, pero a su vez adolesce de grandes limitaciones. Prácticamente la única alternativa para controlar el camino que el tráfico toma a través de una red IP es cambiar el costo de un enlace particular. Generalmente los protocolos de ruteo, deciden el camino que seguirá el tráfico eligiendo aquel de menor costo frente los diferentes caminos alternativos en base a algún tipo de medida del mismo (en su manera más simple puede ser la cantidad de saltos). Por lo tanto, no existe ninguna manera razonable de controlar el camino que el flujo de datos seguirá en base al origen de éstos, sino que solo se puede hacer en base a su destino. A pesar de todo, la ingeniería de tráfico en IP es viable, y existen grandes redes que la utilizan satisfactoriamente.

Por otro lado, ATM permite definir caminos virtuales permanentes (PVC), para el tráfico de un determinado origen hacia un determinado destino. Esto significa que el administrador de la red puede tener un control mucho más fino sobre los flujos de tráfico dentro de una red. Los principales proveedores de acceso a internet del mundo utilizan ATM para decidir el camino a seguir por el tráfico. Esto se implementa definiendo un conjunto completo de PVC's entre todos sus routers y periódicamente modificando y reposicionando los mismos en base al tráfico observado a través de su red.

A pesar de la notoria mejora comparativa que proporciona ATM por sobre IP, a la hora de implementar TE, la complejidad de su implementación lo hace inviable en muchos casos, especialmente si se desea implementar un algoritmo que tome decisiones en tiempo real.

El problema del pez

Uno de los problemas clásicos que ha servido como ejemplo representativo para las aplicaciones de la ingeniería de tráfico, y en particular para remarcar las limitaciones de IP en este terreno se conoce como el "problema del pez".

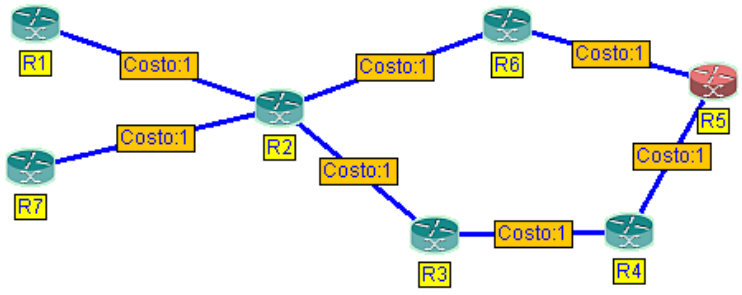

Figura 2.1: Topología del problema del pez

En la figura 2.1 se observa una topología de red en la que existen dos caminos alternativos para ir desde el router **R2** a **R5**.

1. **R2** → **R6** → **R5**
2. **R2** → **R3** → **R4** → **R5**

A su vez todos los links tienen el mismo costo (1), por lo que si se trabaja con un ruteo basado en el destino todos los paquetes provenientes de **R1** o **R7** cuyo destino sea **R5** serán enviados por el camino 1, ya que éste es de menor costo que el 2. Por lo tanto, la alta utilización del camino 1 puede llevar a congestión del mismo, generando pérdida de información, mientras que el camino 2 queda totalmente desaprovechado ya que no transporta tráfico alguno.

Trabajado con este tipo de políticas de ruteo, no existe una solución sencilla para esta problema, ya que si se desea utilizar el camino 2, y se configuraran los enlaces de manera que su costo sea menor que el del 1, el problema simplemente se trasladaría, ya que el camino 2 sería el congestionado en ese caso. En el caso de la red del ejemplo, sería sencillo configurar los dos caminos de manera que ambos tengan igual costo y el tráfico se balancearía entre ellos. Si bien esto es manejable en este caso, sería prácticamente imposible llevarlo a cabo en una red de gran porte.

Otra alternativa sería implementar algún protocolo de ruteo dinámico, el cual selecciona la ruta a seguir por los paquetes en base al nivel de utilización de los caminos posibles. En nuestro caso, al comienzo, cuando ambos caminos se encuentran sin cursar tráfico, se enviaría todo por el camino 1 debido a su menor costo. Pasado el tiempo, se detectaría la congestión antes mencionada, y el protocolo pasaría a enviar todo el tráfico por el camino 2 hasta que éste también se congestione dejando el camino 1 sin utilizar. De esta manera la congestión oscilaría entre los dos caminos sin llegar a una solución aceptable.

En caso que la red sea ATM, el problema anterior sería trivial ya que se podrían construir dos PVC que distribuyan el tráfico entre los caminos existentes de manera óptima. De todas maneras, como ya se mencionó, implementar este tipo de políticas de ruteo en ATM, en especial si la red es de tamaño considerable conlleva una gran complejidad.

En conclusión, ¿como es posible combinar la gran potencia de ATM en lo que a ingeniería de tráfico se refiere con la simplicidad de ruteo de IP? La respuesta a esta pregunta la provee MPLS.

2.2.2. MultiProtocol Label Switching (MPLS)

En los últimos años se ha visto un gran crecimiento de internet, no solo en el aumento de los usuarios, sino que además se han desarrollado una enorme diversidad de aplicaciones. Cada día aparecen nuevos tipos de tráfico que deben ser cursados, llevando Internet hacia una red convergente, donde debe convivir el tráfico de datos tradicional, con los contenidos multimedia, los cuales han hecho crecer el requerimiento de ancho de banda y de velocidad de la infraestructura actual de las redes.

El desafío que se presenta, es el de poder ofrecer distintos tratamientos a los diferentes tipos de tráfico que claramente tienen requerimientos variados. Aparece la necesidad de mejorar parámetros tales como el ancho de banda y el retardo, pero sobre todo de poder garantizarlos, ofreciendo servicios diferenciados, con diferentes calidades. Las soluciones que se ofrecen actualmente si bien permiten encarar los problemas de demanda de velocidad y ancho de banda para los paquetes atravesando las redes no permiten resolver lo anterior.

Debido a todo lo anterior, y a la aceptación de los protocolos TCP/IP como estándar de facto para la internet de hoy en día, se prevé que MPLS juegue un rol decisivo en el crecimiento y desarrollo de las actuales redes IP, brindando la posibilidad de cumplir con las necesidades de los usuarios de dichas redes, utilizando técnicas de ingeniería de tráfico y brindando calidad de servicio (QoS) y clases de servicio (CoS).

MPLS surgió como un estándar de IETF [20], y es una arquitectura de red que permite manejar distintos tipos de tráfico de manera eficiente, y puede ser implementada sobre cualquier tecnología de enlace de datos. La idea principal es que MPLS provea las bases para poder manejar los requerimientos de Internet hoy en día, ofreciendo la posibilidad de garantizar la calidad de servicio de los enlaces así como de brindar tratamientos diferenciados a los distintos tipos de tráfico. MPLS permite manejar tráficos con muy diversos requerimientos, permaneciendo independiente de los protocolos de capa 2 y de capa 3 y a su vez ofreciendo interfaces para los protocolos de reserva de recursos tales como RSVP y OSPF.

El funcionamiento de MPLS se basa en la asignación e intercambio de etiquetas, que permiten el establecimiento de caminos (LSP, Label Switched Paths) en la red. Dichas etiquetas son campos simples, de largo fijo, en las cuales de mapean las actuales direcciones IP. Cada LSP se crea en base a concatenar uno o más saltos (hops) en los que se intercambian las etiquetas, de modo que cada paquete se envía de un "conmutador de etiquetas"(LSR, Label-Switching Router) a otro, a través del dominio MPLS. Un LSR no es sino un router especializado en el envío de paquetes etiquetados por MPLS. Es de resaltar que el término especializado se utilizó simplemente en el sentido que el router pueda leer e intercambiar etiquetas, no significando que los mismos deban ser más complejos que los actuales routers IP, sino que de modo contrario, podrían utilizarse dispositivos más sencillos, que simplemente sean capaces de manejar las etiquetas, sin la necesidad de poder procesar los actuales encabezados IP. Cada vez que un paquete ingresa a un dominio MPLS a través de un router de borde (LER, Label Edge Router), se le asigna una FEC (Forwarding Equivalence Class), la cual es única para todos aquellos paquetes que comparten el mismo requerimiento en cuanto a su ruteo. En cada uno de los saltos, cada LSR asigna una etiqueta a los paquetes de una misma FEC y los envía a su próximo salto. De esta manera cada nodo de la red MPLS decide el tratamiento que le da a cada paquete en base a la etiqueta, sin necesidad de examinar el encabezado de la capa de red correspondiente.

Como ya se adelantó, dentro de una red MPLS identificamos básicamente 2 tipos de routers, de núcleo (LSR) y de borde (LER). Los equipos de borde son los encargados de interconectar distintos tipos de redes en el acceso a las redes MPLS. Los mismos juegan un papel muy importante, ya que son los que asignan a cada paquete entrante su FEC, la cual va a determinar el tratamiento y la ruta que dicho paquete seguirá a lo largo de toda la red. Aquí se presenta una de las ventajas de MPLS, el hecho que los LER pueden utilizar mucha información al asignar las clases de equivalencia, como ser el lugar de ingreso a la red, tipo de dato y otros factores que son imposibles de determinar analizando solamente el encabezado IP, el cual solo provee información acerca del destino de los datos. De todas maneras esto no aumenta la complejidad del ruteo interno de la red, ya que todos los demás LSR, no deben asignar una FEC, sino que simplemente deben leer la etiqueta y decidir por medio de ésta el tratamiento que deben darle al paquete. Otra ventaja es que los LER, mediante la asignación de la FEC, pueden forzar al paquete a tomar una determinada ruta (LSP), fijada de antemano.

Etiquetas

En su forma más simple, las etiquetas determinan el camino que los paquetes tomaran dentro de una red. Las mismas son cortas en relación al encabezado IP, de largo fijo y se transportan junto a cada paquete, encapsuladas en el encabezado de capa 2. El funcionamiento es simple, cada router examina la etiqueta de los paquetes que llegan, y determina en base a ella el próximo salto, de esta manera, a lo largo de toda la red

MPLS, el envío de paquetes es en base a la conmutación de etiquetas.

Cada paquete tiene un conjunto de éstas etiquetas, el cual es un stack "LIFO" del cual analiza la superior, la reemplaza y agrega otra(s). La utilidad de esto, entre otras cosas, es la de permitir establecer subredes, dentro de las cuales el ruteo interno se base en la etiquetas superiores, y luego que el paquete llega al borde de la misma, el ruteo sigue con las etiquetas inferiores. En la mayoría de los casos, cada router analiza solo una de las etiquetas, a lo sumo 2.

El largo genérico de las etiquetas es de 4 octetos, dentro de los cuales encontramos 20 bits que corresponden a la etiqueta en sí, mientras que los restantes se dividen en 4 campos que agregan funcionalidad (por ejemplo TTL); [20], [11].

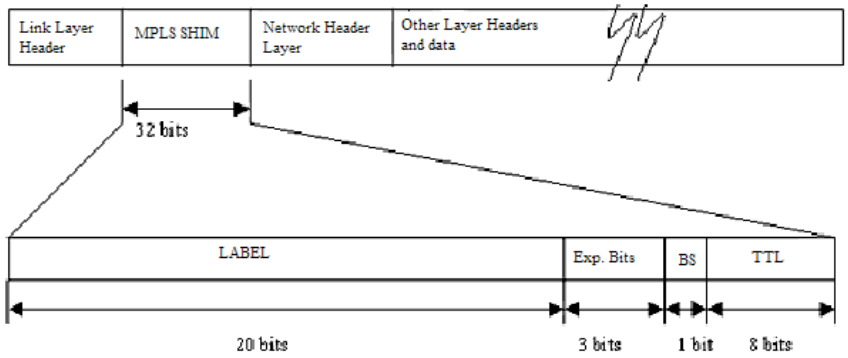

Figura 2.2: Etiqueta genérica de MPLS

La utilización de etiquetas permite un encaminamiento de los paquetes mucho más simple y veloz, no solo por su reducido tamaño, sino también por la simplicidad de las tablas de ruteo. Las etiquetas incluso pueden ser leídas y procesadas en hardware. De todas maneras, las etiquetas no fueron pensadas, ni se implementaron, para eliminar las actuales direcciones IP. La razón de esto es que hay una necesidad de poder identificar el origen de los paquetes, con fines de protección y filtrado o de ruteo mismo, cosa que necesita del encabezado de capa 3. Además, no es probable que los sistemas finales (hosts) implementen MPLS, sino que estos deberán conectarse con un dispositivo de red que pueda enviar los paquetes adecuadamente.

Para la distribución de las etiquetas se puede utilizar algún protocolo de señalización como el RSVP, o el LDP (Label Distribution Protocol), que fue definido por el grupo de trabajo de MPLS del IETF [1].

LSP

Un LSP es el circuito virtual que siguen todos los paquetes pertenecientes a una misma FEC, el cual es establecido antes del envío de la información.

Llamamos LSP de profundidad m, de un paquete P a la secuencia de routers, que comienza con un LER de ingreso, el cual coloca una etiqueta de profundidad m (en el lugar m del stack); luego sigue con todos los routers intermedios, que basan su decisión en una etiqueta de profundidad m, y que termina en el LER de egreso, en el cual la decisión se toma a través de una etiqueta de nivel menor que m, o mediante alguna otra manera (por ejemplo encabezado IP). En caso de querer establecer un nuevo LSP, en medio del camino original de profundidad m, la profundidad de las etiquetas de este nuevo LSP en el stack, debe ser mayor que m, si por ejemplo para ir del router R_i al

R_{i+1} el paquete atraviesa una subred cuyos nodos no son parte del LSP de profundidad m, se deberá generar un nuevo LSP de profundidad m+1, por ejemplo.

Comúnmente la etiqueta de nivel m se le saca al paquete en el penúltimo salto, ya que el último no basa su decisión en ella. Esto, además de ser lo lógico debido a que el último salto del LSP no basa su decisión en dicha etiqueta es más eficiente, ya que permite que todos los nodos de un LSP tomen sus decisiones en base a un único examen de la etiqueta.

Selección de las rutas

Cuando un paquete de una determinada FEC va a ser encaminado, se debe decidir cual será el LSP que seguirá. Para esto existen 2 opciones.

1. **Hop by hop routing**: cada LSR decide independientemente su próximo salto, en base a la etiqueta. Es similar a la manera en que rutea en IP.

2. **Explicit routing**. Un LSR dado (generalmente el ingreso o egreso) decide (toda) la ruta que seguirán los paquetes de la FEC. Esta variedad de ruteo puede ser *estricto* o *laxo* (loose routing). El ruteo estricto implica que el LSR determina completamente la ruta que seguirá el tráfico, mientras que la otra variedad solamente indica algunos nodos por lo que deberá pasar el mismo. La ruta para los paquetes puede ser configurada manualmente o puede ser decidida por el LSR dinámicamente a partir de alguna información de estado de la red.

La segunda modalidad es la más atractiva a la hora de aplicar ingeniería de tráfico o *policy routing*[1]. Las rutas deben ser asignadas en el momento en que las etiquetas son distribuidas, lo que tiene la gran ventaja (respecto al ruteo desde la fuente (source routing) de IP por ejemplo) que no se debe hacer paquete a paquete, o sea se le asigna la ruta a una FEC y luego todos aquellos paquetes con la etiqueta correspondiente a dicha FEC la van seguir.

En caso que un router reciba un paquete, y no tenga información de que hacer con él, puede ser peligroso tratar de rutear el mismo mediante el encabezado IP ya que la información que contiene el encabezado IP puede no ser suficiente para que el router lo encamine correctamente, lo más seguro es descartar este tipo paquete.

TTL

Cuando un paquete atraviesa una red (subred) MPLS, la cual está dentro de una red mayor (IP, por ejemplo), el encabezado de la capa de red correspondiente nunca es modificado, por lo cual, al salir del dominio MPLS, el TTL debe ser decrementado el valor que corresponda. Si las etiquetas MPLS están en un encabezado propio (MPLS-SHIM, encapsulado en el encabezado de capa 3) entonces en dicho encabezado debe haber un campo que sea el TTL, el cual deberá ser copiado del encabezado de capa 3 al ingreso y luego copiado nuevamente al encabezado de capa 3 a la salida. En caso que las etiquetas viajen dentro de otro encabezado de capa 2, y dicho encabezado no tenga un campo de TTL entonces lo anterior no será posible, y deberá haber alguna manera que el LSR de ingreso sepa el largo del LSP y pueda decrementar el valor correspondiente del TTL de capa 3. El problema de los LSP que no pueden decrementar el TTL es que no hay control de bucles, por lo que esto deberá ser implementado de alguna otra manera.

[1] En algunos casos puede ser necesario forzar a los paquetes a tomar una determinada ruta no debido a cuestiones de ingeniería de tráfico sino por motivos de otra índole, como comerciales o políticos

Label Merging

Cuando un router recibe paquetes de la misma FEC que tienen distintas etiquetas (de dos interfaces diferentes por ejemplo) es conveniente que pueda "juntar" todas las etiquetas en una sola para que todos los paquetes de la FEC salgan con la misma etiqueta.

Posibles aplicaciones de MPLS a la ingeniería de tráfico

Si bien MPLS surgió como una alternativa para acelerar la elección del próximo salto de los paquetes, al atravesar una red debido a lo pequeño de su etiqueta, hoy en día esta es la menos útil de todas sus ventajas, ya que este procesamiento se puede hacer extremadamente rápido gracias al desarrollo de los equipos y hasta incluso en hardware.

Lo que realmente hoy en día posiciona a MPLS como una arquitectura privilegiada, es la amplia gama de aplicaciones que posibilita y que son difíciles de implementar en las actuales redes IP. A modo de ejemplo recordemos el problema presentado en la sección 2.2.1 (problema del pez) donde IP y ATM adolecían de ciertas carencias.

A continuación presentamos algunas posibles herramientas que ofrece MPLS para encarar el antedicho problema y resolverlo de manera eficiente y sencilla.

Es posible enviar ciertas clases de tráfico a través de un determinado LSP prefijado (túnel LSP), el cual no tiene por que ser el mismo por el que viajarían los paquetes si fueran ruteados independientemente, lo que se conoce *Explicitly Routed LSP*. Esto se puede llevar a cabo en MPLS, siempre y cuando se configure lo siguiente:

1. Una manera de identificar cuáles paquetes deberán ser ruteados a través del túnel.
2. Una manera de establecer el túnel
3. Evitar que los paquetes que salgan del túnel vuelvan al ingreso formando un loop.

El LSR al ingreso del túnel, deberá sacar la última etiqueta del paquete, y colocar aquella que le indique el LSR de egreso. Además por encima de ésta deberá colocar la etiqueta correspondiente al próximo salto dentro del túnel.

Estos LSP requieren poco mantenimiento y permiten dirigir el tráfico por donde los recursos son los adecuados para el mismo. Por otra parte MPLS permite no solo la agregación de tráfico sino que también permite la desagregación del mismo de manera simple.

Balance de Carga Puede que un solo LSP no pueda transportar toda la carga necesaria, entonces se pude implementar balance de carga a través de LSP paralelos que no sobrecarguen los recursos de ninguno de los enlaces utilizados.

Constraint Based Routing (CBR) Permite la reserva de recursos en la base a la demanda, para esto toma como datos:

- Atributos de los agregados de tráfico
- Atributos de los recursos
- Información del estado de la topología de la red

Basado en estos atributos, la aplicación de CBR en cada nodo, automáticamente, permite definir rutas explícitas para cada flujo que se origina en el nodo. Se especifica un LSP que cumpla con los requisitos del mismo expresado en sus atributos, sujeto a las restricciones impuestas por la disponibilidad de recursos.

2.3. Resumen

El avance de la tecnología ha traído nuevos problemas al área de la transmisión de datos. Asimismo, han aparecido nuevas arquitecturas que buscan atacar la solución de los mismos. En este capítulo se ha hecho un resumen del estado actual de situación de la tecnología, en términos de lo que a Ingeniería de tráfico se refiere.

En particular, se ha enfocado en MPLS porque muestra ser una herramienta útil en el establecimiento de políticas de Ingeniería de Tráfico. Las características de esta arquitectura de red, fueron las que nos llevaron a desarrollar nuestro trabajo orientado a redes que implementen dicha arquitectura.

Parte II

Principios y Desarrollos Teóricos

Capítulo 3

Evaluación de Performance mediante Grandes Desviaciones

Las redes de datos modernas requieren poder estimar ciertos parámetros de calidad de servicio a los efectos de decidir si es posible prestar algunos servicios, como ser aquellos de tiempo real. En este aspecto, mucha teoría se ha desarrollado con distintos resultados en lo que a efectividad en la predicción y aplicabilidad se refiere.

El presente capítulo, en la sección 1, comienza por determinar que factores se considerarán relevantes a la hora de evaluar la performance de una red. Posteriormente se describe aquella teoría que por sus resultados y generalidad se escogió como pilar fundamental de este proyecto. Es así que en la sección 2, se comienza por introducir las ideas generales de la rama de la probabilidad que es la Teoría de Grandes Desviaciones y los motivos que llevaron a su elección. Posteriormente en la sección 3 se expone cómo se aplica esta teoría para estudiar la performance de una cola en el régimen conocido como de "Muchas Fuentes", para culminar en la sección 4, donde se presenta el Régimen de Muchas fuentes y Buffer Pequeño, el cual permite extender el análisis a redes completas, en particular aquellas en las que es posible utilizar caminos virtuales, como las redes MPLS. Este último proporciona el marco dentro del cual se desarrolla el proyecto en su objetivo de evaluar la performance de una red en todos los nodos de la misma.

Este capítulo es de carácter mayormente teórico con alto contenido matemático, mientras que en el siguiente se presentan de manera más aplicada aquellos conceptos que se presentarán a continuación.

3.1. Introducción

En el estudio de las redes de datos, dependiendo del tipo de aplicación en estudio, la probabilidad de pérdida de paquetes puede ser el índice de performance más restrictivo y del cual depende la viabilidad de brindar distintos tipos de servicio. Por ésto es necesario estimarla y determinar su variación frente a cambios en la estructura de la red, buffers, capacidad de los enlaces, etc.

En particular, el interés en estimar la probabilidad de pérdida se justifica por el aumento de la incidencia de aplicaciones de tiempo real en las redes modernas, lo que ha hecho aumentar el interés por el desarrollo de técnicas de estimación y predicción del funcionamiento de un enlace, desde el punto de vista de distintos parámetros de Calidad de Servicio (QoS).

Desde el punto de vista de Teoría de Colas, se trata de estudiar la probabilidad de llenado o saturación de una *"sala de espera"* o *buffer* al que llegan datos, que son

servidos según cierta política. Las técnicas clásicas de Teoría de Colas, que se aplicaron con éxito durante años en la telefonía tradicional, dejaron de ser suficientes debido a la aparición de nuevos fenómenos en el tráfico, que requieren de nuevos modelos y enfoques para su tratamiento.

Es en este contexto que se desarrolló a mediados de los 90 un nuevo enfoque para este problema. El mismo se basa en la utilización de la Teoría de Grandes Desviaciones para proporcionar una estimación de la probabilidad de pérdida. La ventaja de esta nueva técnica es que las hipótesis exigidas a la estadística del tráfico son suficientemente generales como para abarcar a la mayoría de los modelos que hoy aparecen en las redes modernas. Esto constituye un avance muy importante frente a los modelos del tipo Poisson aplicados en la teoría tradicional, y proporcionó un fuerte impulso a la investigación en esta área.

En este enfoque se desarrolla además el concepto de *ancho de banda efectivo (effective bandwidth)* [13], que representa una medida de la cantidad de recursos efectivamente utilizados por la fuente de datos. Dicha magnitud aparece estrechamente relacionada con la teoría de grandes desviaciones, y depende de la estadística del tráfico, así como de la capacidad y el tamaño del buffer, y de la interacción con las otras fuentes de tráfico que comparten el mismo.

Como ventaja adicional, los avances más recientes de la teoría se orientan a obtener resultados sobre la probabilidad de pérdida ya no solo en un enlace o cola aislado, sino en los diferentes puntos de una red. Esto permite un análisis global a nivel de, por ejemplo, todo un dominio de administración, pudiendo obtener estimaciones de lo que dicha red está en condiciones de ofrecer, y cómo diseñar para obtener la calidad de servicio deseada.

3.2. Teoría de Grandes Desviaciones

La teoría de Grandes Desviaciones proporciona un método analítico para estudiar eventos cuya probabilidad de ocurrencia es pequeña. Tuvo origen en el estudio de la probabilidad de obtener promedios de variables aleatorias alejados del valor medio de las mismas. En esta sección se introducen algunos de sus resultados. Para un enfoque más completo, se puede consultar [8].

3.2.1. Conceptos generales.

La idea básica de la teoría es que, si se tiene una sucesión de variables aleatorias reales X_n, se pueden establecer resultados asintóticos de la forma:

$$\mathbf{P}\left(X_n \in B\right) \approx e^{-nI(B)} \quad (n \to \infty) \tag{3.1}$$

siendo $B \subset \mathbb{R}$, e $I(B)$ un coeficiente que depende del conjunto B y de la distribución de las variables X_n.

A continuación definimos más formalmente estos conceptos.

Definición 3.1 *Diremos que X_n satisface un* Principio de Grandes Desviaciones (LDP) *si existe una función $I : \mathbb{R} \to \mathbb{R}^+$ tal que se cumple:*

$$-\inf_{x \in B^\circ} I(x) \leq \liminf_n \frac{1}{n} \log \mathbf{P}\left(X_n \in B\right) \leq \limsup_n \frac{1}{n} \log \mathbf{P}\left(X_n \in B\right) \leq -\inf_{x \in \bar{B}} I(x) \tag{3.2}$$

A la función I se le denomina función de tasa *o* función de velocidad *del LDP.*

Si los extremos de la desigualdad 3.2 coinciden, la ecuación puede reescribirse como:

$$\lim_n \frac{1}{n} \log \mathbf{P}\left(X_n \in B\right) = - \inf_{x \in \bar{B}} I(x)$$

Haciendo $I(B) = \inf_{x \in \bar{B}} I(x)$ se tiene, para n grande, la aproximación presentada en la ecuación 3.1.

La función de velocidad indica el orden o tasa con que la probabilidad del conjunto decae a 0. Un LDP establece que dichas probabilidades decaen exponencialmente, y que la tasa de decaimiento de cada conjunto está determinada por aquel punto que tenga una tasa de decaimiento más lenta (la menor de todas las $I(x), x \in B$).

3.2.2. El Teorema de Cramér.

Uno de los primeros resultados de la teoría es el establecer un LDP para el promedio de variables aleatorias *iid* (independientes e idénticamente distribuidas). El resultado se conoce como Teorema de Cramér y se presenta a continuación. Este teorema encuentra aplicación en el estudio de colas porque permite estudiar el comportamiento de la superposición (suma) de un conjunto de fuentes independientes, como se ve en la sección 3.3.

Es sabido que, si $\{X_n : n \in \mathbb{N}\}$ es una sucesión de variables aleatorias *iid*, tales que $\mathbf{E}\left(|X_1|\right) < \infty$, entonces por la *ley débil de los grandes números* se cumple que:

$$\mathbf{P}\left(\left|\bar{X}_n - \mu\right| > \varepsilon\right) \underset{n \to \infty}{\longrightarrow} 0 \quad \forall \varepsilon > 0 \tag{3.3}$$

siendo $\bar{X}_n$ el promedio de $X_1, \ldots, X_n$ y $\mu = \mathbf{E}\left(X_1\right)$. En particular, la ecuación anterior implica que si $B \subset \mathbb{R}$ es un conjunto tal que $\mu \notin B$ entonces se cumple que:

$$\mathbf{P}\left(\bar{X}_n \in B\right) \underset{n \to \infty}{\longrightarrow} 0$$

Establecer un LDP para la sucesión de variables $\left\{\bar{X}_n\right\}$ permite determinar la velocidad de esta convergencia. Para ello, comenzamos por introducir algunas definiciones.

Definición 3.2 *Se llama* función generatriz de momentos logarítmica *de una variable aleatoria X a:*

$$\Lambda(s) = \log \mathbf{E}\left(e^{sX}\right) \quad s \in \mathbb{R}$$

Esta función tiene algunas propiedades, a saber, es convexa, $\Lambda(0) = 0$ y $\Lambda'(0) = \mathbf{E}\left(X\right)$.

Definición 3.3 *Se llama* conjugada convexa *o* transformada de Fenchel-Legendre *de Λ a la función:*

$$\Lambda^*(x) = \sup_{s \in \mathbb{R}} \left\{sx - \Lambda(s)\right\}$$

Puede verificarse que esta función también es convexa, siendo además $\Lambda^*(x) \geq 0 \, \forall x$ y $\Lambda^*(\mu) = 0$.

El teorema siguiente responde a la interrogante planteada sobre la velocidad de convergencia.

Teorema 3.1 (Cramér) *Sea $\{X_n : n \in \mathbb{N}\}$ una sucesión de variables iid y*

$$\bar{X}_n = \frac{1}{n} \sum_{i=1}^{n} X_i$$

entonces la sucesión de variables $\left\{\bar{X}_n : n \in \mathbb{N}\right\}$ verifica un LDP con función de velocidad $I(x) = \Lambda^*(x)$.

Para ver la utilidad del teorema, consideremos un conjunto de la forma $B = [b, +\infty)$, con $b > \mu$. Como $\Lambda^*(x)$ tiene un mínimo en μ y es convexa, se tiene que

$$\inf_{x \in B^\circ} \Lambda^*(x) = \inf_{x \in \bar{B}} \Lambda^*(x) = \Lambda^*(b)$$

Por lo tanto, podemos escribir:

$$\lim_{n \to \infty} \frac{1}{n} \log \mathbf{P}\left(\hat{X}_n \geq b\right) = -\Lambda^*(b)$$

o bien, que para $n \to \infty$ se puede aproximar:

$$\mathbf{P}\left(\hat{X}_n \geq b\right) \approx e^{-n\Lambda^*(b)}$$

Observación 3.1 *Debe acotarse que la aproximación no es necesariamente precisa. Para ello, observemos que si la probabilidad anterior es de la forma:*

$$\mathbf{P}\left(\hat{X}_n \geq b\right) = C_n e^{-n\Lambda^*(b)}$$

entonces lo único que podemos afirmar es que $\log(C_n) = o(n)$ *lo cual no es una restricción exigente, pudiendo llegar a ocurrir que* $C_n \to \infty$ *con lo cual el error cometido en la aproximación puede ser grande.*

A los efectos de mejorar la estimación obtenida a partir del teorema de Cramér, se dispone de un teorema adicional que trata con conjuntos de la forma $[b, +\infty)$. Lo presentamos a continuación ya que este tipo de conjuntos son los que nos interesan al calcular las probabilidades de pérdida, debido a que interesa estudiar eventos de la forma $\mathbf{P}\left(\bar{X}_n > b\right)$.

Teorema 3.2 (Bahadur - Rao) *Sea* $\{X_n : n \in \mathbb{N}\}$ *una sucesión de variables iid y* $\bar{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i$. *Si* $\mathbf{E}(X_1) = \mu$ *(finito) y* $b > \mu$ *entonces existe* s^* *solución de:*

$$s^* : \Lambda'(s^*) = b$$

y se verifica que:

$$\mathbf{P}\left(\bar{X}_n \geq b\right) \underset{n \to \infty}{\sim} \frac{1}{s^* \sqrt{\Lambda''(s^*) 2\pi n}} e^{-n\Lambda^*(b)}$$

3.2.3. El Principio de Contracción.

Otra herramienta útil de la teoría de grandes desviaciones es el llamado *Principio de Contracción* que presentamos a continuación.

Teorema 3.3 (Principio de Contracción) *Sea* $\{X_n : n \in \mathbb{N}\}$ *una sucesión de variables aleatorias que satisfacen un LDP con función de velocidad* I_X. *Sea* $f : \mathbb{R} \to \mathbb{R}$ *una función continua. Entonces la sucesión de variables aleatorias* $\{Y_n\}$ *definidas como* $Y_n = f(X_n)$ *también satisface un LDP con función de velocidad:*

$$I_Y(y) = \inf \{I(x) : x \text{ tales que } y = f(x)\}$$

El teorema anterior permite extender los resultados de un LDP sobre una sucesión de variables a una función de las mismas, siempre que se pueda probar que dicha función es continua. Este principio está atrás de muchos resultados, para los que se dispone un LDP para la superposición de tráfico de entradas (que se pueden suponer independientes), pero interesa estudiar una función de la entrada, como puede ser el tamaño de cola. Esto permite también, como se verá más adelante, estudiar las tasas de arribo de trabajo a buffers interiores a una red, a partir de las tasas de llegada a la misma, ya que esta dependencia es continua.

3.3. El régimen de muchas fuentes

En esta sección se presenta un modelo de estudio del problema clásico de teoría de colas, en el régimen límite de que la cantidad de fuentes de trabajo que se superponen en la misma tiende a infinito. A su vez, los recursos disponibles para servir a estas fuentes se escalan de manera proporcional a la cantidad de fuentes presentes. Esto permite usar variantes del teorema de Cramér para analizar el problema.

3.3.1. Teorema de Loynes

Consideremos un nodo cualquiera de la red como una cola FIFO[1] y 2 procesos que tienen lugar en ella, uno de arribo $a(n)$ y otro de servicio $c(n)$, con $n \in \mathbb{Z}$ en ambos casos.

Si $a(n)$ es la cantidad de trabajo que arriba a la cola en el intervalo $[n, n+1]$ y $c(n)$ el trabajo procesado en dicho intervalo, entonces, la cantidad de trabajo que se agrega en la cola en el intervalo $[n, n+1]$ estará dada por la siguiente ecuación

$$x(n) = a(n) - c(n) \tag{3.4}$$

Por lo que el comportamiento de la cola estará determinado por lo que se conoce como la ecuación de Lindley

$$Q(n+1) = (Q(n) + x(n))^{+} \tag{3.5}$$

siendo $Q(n)$ la cantidad de trabajo que permanece en la cola en el instante n y $Z^{+} = \text{máx}\{Z, 0\}$.

De lo anterior está claro que la probabilidad de overflow estará directamente vinculada con el comportamiento del proceso $Q(n)$, por lo cual su estudio será necesario. De la ecuación de Lindley (ec. 3.5) se puede ver que:

$$Q(0) = \text{máx}(X_0, X_1, ..., X_n + Q(-n)) \tag{3.6}$$

donde

$$X_k = \sum_{i=-k}^{i=-1} x(i) \tag{3.7}$$

Imponiendo algunas hipótesis al proceso de entrada, puede probarse que en régimen $(n \to \infty)$ lo anterior determina la distribución estacionaria de la cola. Este resultado se conoce como el *Teorema de Loynes*.

Teorema 3.4 (Loynes) *Si el proceso x antes definido verifica que:*

- *Es estacionario*
- *Es ergódico*
- $\mathbf{E}(x) < 0$

entonces para toda condición inicial de la cola Q, se verifica que

$$\lim_{n\to\infty} \mathbf{P}(Q(n) \leq x) = \mathbf{P}\left(\sup_{n>0}(X_n) \leq x\right) \tag{3.8}$$

donde $Q_0 = \sup_{n>0} X_n$ es casi seguramente finita.

Por lo tanto, a partir de este resultado, podemos estudiar la probabilidad de overflow de un enlace mirando la distribución en régimen de la cola.

[1] First In, First Out: el trabajo que llega primero es lo primero en salir.

3.3.2. El modelo de muchas fuentes

El régimen asintótico de muchas fuentes estudia el comportamiento de una cola mediante un LDP (ver sección 3.2). Consideremos un enlace al que arriban N fuentes *iid* y que dicho enlace tiene una capacidad c y un tamaño de buffer b *por fuente*, es decir $C(N) = Nc$ y $B(N) = Nb$ como en la figura 3.1.

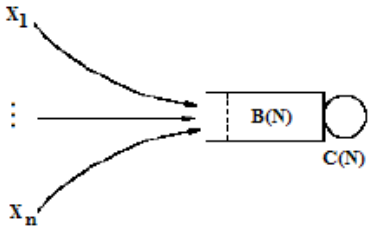

Figura 3.1: Caso de estudio

Courcoubetis y Weber, en 1996, probaron que se puede establecer un LDP para las variables Q_N (distribución de la cola cuando se superponen N fuentes) para $N \to \infty$, como se puede ver en [5].

Más exactamente, se tiene que:

Teorema 3.5 (Courcoubetis-Weber) *En el régimen anterior se cumple:*

$$\lim_{N\to\infty} \frac{1}{N} \log \mathbf{P}\left(Q_N > B\right) = -I \tag{3.9}$$

con I dado por:

$$I = \inf_t \sup_s ((ct + b)s - st\alpha(s,t)) \tag{3.10}$$

donde $\alpha(s,t)$ es el ancho de banda efectivo de la fuente que se define según la ecuación (3.11)

$$\alpha(s,t) = \frac{1}{st} \log \mathbf{E}\left(e^{sX[0,t]}\right) \tag{3.11}$$

donde X[0,t] es la cantidad de trabajo acumulado en el intervalo $[0,t]$.

Intuitivamente, este teorema está muy relacionado con los resultados de la sección anterior. El supremo en la variable s surge de estudiar la función Λ^* del tráfico de entrada en diferentes escalas de tiempo. El papel del punto b lo juega ahora el punto $b + ct$ que está relacionado con la región donde se produce el *overflow*. Por último, el ínfimo en la variable t indica que de todas las escalas de tiempo, la que domina el LDP es aquella que tiene la menor velocidad de decaimiento (es decir, el tiempo más probable de overflow).

Basados en esto estamos en condiciones de estimar la probabilidad de overflow de un enlace, a partir del ancho de banda efectivo del flujo (agregado) que arriba al mismo como:

$$\mathbf{P}\left(Q > B\right) \approx e^{-NI} \tag{3.12}$$

Con este resultado teórico, a los efectos de estimar la probabilidad de overflow resta estimar el ancho de banda efectivo de una traza y resolver el problema de optimización de la ecuación 3.10.

3.4. El régimen de muchas fuentes y buffer pequeño

El régimen de muchas fuentes introducido en la sección 3.3 presenta como principal problema el que sus resultados no son extensibles al análisis de una red completa. Los primeros intentos de extender este análisis fueron hechos por D. Wischik [21] pero la aplicabilidad de sus resultados es limitada. El principal problema de su enfoque es el requerimiento de independencia entre las entradas a cada enlace la red. La anterior situación es de esperar a la entrada, pero no sería razonable dicho requerimiento para los enlaces en los enlaces interiores de la red.

Recientemente Mazumdar et al presentan un nuevo enfoque. El principal resultado del mismo es permitir el análisis en cualquier parte de la red, con la única hipótesis de independencia en los tráficos a la entrada de la red. La principal diferencia con el modelo antes presentado es el tamaño del buffer en cada enlace de la red. Como su nombre lo indica, el buffer es "pequeño", el mismo crece con el numero de fuentes, pero a diferencia del régimen de muchas fuentes no lo hace de manera lineal, sino que mas lento.

En la sección 3.4.1 se presentan los resultados del artículo [16] en toda su formalidad. Más adelante, en la sección 4 se describe cómo se aplican los mismos, indicando aquellos principales factores a tener en cuenta a la hora de implementar esta técnica.

3.4.1. Estimación de la probabilidad de overflow en este modelo

Supongamos que se tiene una red de enrutadores y enlaces de topología conocida, sobre la que se definen flujos de tráfico que la atraviesan. Lo anterior puede representar por ejemplo tráfico circulando por los LSPs de una red MPLS como se describe en el capítulo 2.2.2.

Hipótesis sobre el tráfico: Supondremos que la velocidad de arribo de datos a la entrada de un LSP es un proceso estocástico estacionario y ergódico, superposición de N fuentes independientes y con distribución que representaremos por una variable X_m con $m = 1, \ldots, M$ siendo M el número de LSPs definidos en la red. Denotaremos $\rho_m = \mathbf{E}X_m$ a la media del tráfico. Si representamos por $X_m^N[0,t]$ el trabajo acumulado en $[0,t]$ cuando se superponen N fuentes supondremos que vale el siguiente principio de grandes desviaciones:

$$\begin{aligned} -\inf_{x\in B^\circ} I_{m,t}(x) &\leq \liminf_{N\to\infty} \frac{1}{N}\log \mathbf{P}(X_m^N[0,t]/N \in B) \leq \\ &\leq \limsup_{N\to\infty} \frac{1}{N}\log \mathbf{P}(X_m^N[0,t]/N \in B) \leq -\inf_{x\in \bar{B}} I_{m,t}(x) \end{aligned} \tag{3.13}$$

para todo $B \subset \mathbb{R}$ boreliano y siendo $I_{m,t} : \mathbb{R} \to [0,\infty]$ continua y con conjuntos de nivel compactos, denominada *función de velocidad*.

Se requiere además la siguiente hipótesis técnica sobre las funciones de velocidad:

$$\liminf_{t\to\infty} \frac{I_{m,t}(at)}{\log t} > 0 \quad \forall a > \rho_m, m = 1, \ldots, M \tag{3.14}$$

Si bien la anterior hipótesis no es adjudicable a ninguna característica física de la red, se afirma en [16] que la misma es verificada por la mayor parte de los modelos de tráfico, incluyendo aquellos con dependencias largas.

Si se puede suponer además que el flujo m es un agregado de fuentes independientes, entonces por el teorema de Cramér, la función $I(x)$ admite la expresión

$$I_{m,1}(x) \triangleq I_m(x) = \sup_s \{sx - s\alpha(s,1)\} \tag{3.15}$$

donde $\alpha(s,t)$ definido en la ecuación 3.11.

A partir de este momento nos vamos a referir a los caminos virtuales como LSP de una red MPLS, pero lo que se representa es general, no teniendo ninguna característica específica de estas redes. Cada uno de los LSP recorrerá un conjunto de enlaces $\mathbf{k}_m = (k_m^{(1)}, \ldots, k_m^{(l_m)})$.

Hipótesis sobre los enlaces: Supondremos que la red tiene K enlaces. Cada enlace $k = 1, \ldots, K$ tiene una capacidad NC_k que por lo tanto crece linealmente con la cantidad de fuentes que alimentan la red. Supondremos también que el ingreso al enlace está precedido por un buffer con política FIFO y sin prioridades. El tamaño de dicho buffer será $B_k(N)$ y se supondrá que $B_k(N)/N \to 0$ cuando $N \to \infty$ (hipótesis de buffer pequeño).

Por último, se requiere una hipótesis de estabilidad. Definiendo el conjunto de tipos de tráfico que pasan por el nodo k por $\mathcal{M}_k = \{m : k_i^m = k \text{ para algún } i\}$, se supondrá entonces:

$$\sum_{m \in \mathcal{M}_k} \rho_m < C_k \tag{3.16}$$

Esta hipótesis es fundamental para que el problema tenga sentido, ya que en caso de que la media de arribo de trabajo supere la capacidad del enlace, el mismo se verá siempre desbordado por lo que carece de sentido estudiar el problema.

Por lo tanto, bajo las hipótesis anteriores se demuestra lo siguiente:

Proposición 3.1 *Existe una función continua $g_{m,k} : \mathbb{R}^M \to \mathbb{R}$ tal que la velocidad de llegada de tráfico de tipo m al enlace k, que denotamos $X_{m,k}^N$ satisface:*

$$X_{m,k}^N/N = g_{m,k}(X_1^N/N, \ldots, X_M^N/N) + o(1) \tag{3.17}$$

Es decir, a menos de un infinitésimo, la tasa normalizada de llegada de trabajo en cualquier nodo es una función continua de las tasas de llegada de trabajo a la red para cada tipo de tráfico.

La demostración de la proposición anterior incluye la construcción explícita de la función $g_{m,k}$ para el caso de una red *feed forward*. Ésta se basa en la composición de funciones de la forma:

$$f_m\left(x_1, \ldots, x_M, C_k\right) = \frac{x_m C_k}{\max\left\{\sum_{i=1}^M x_i, C_k\right\}} \tag{3.18}$$

las cuales relacionan las tasas de entrada y salida de trabajo en un nodo.

Se tiene el siguiente:

Corolario 3.1 *Existe una función continua g_k tal que, a menos de un infinitésimo, permite calcular la tasa total de llegada de trabajo a un enlace a partir de las tasas de entrada a la red.*

Demostración:
Simplemente hay que observar que la tasa total de trabajo que llega a un enlace es $\sum_{m \in \mathcal{M}_k} X_{m,k}^N$ y por lo tanto, tomando:

$$g_k(X_1^N/N, \ldots, X_M^N/N) = \sum_{m \in \mathcal{M}_k} g_{m,k}(X_1^N/N, \ldots, X_M^N/N) \tag{3.19}$$

se tiene que g_k es continua y

$$\sum_{m \in \mathcal{M}_k} (X_{m,k}^N/N) - g_k(X_1^N/N, \ldots, X_M^N/N) \underset{N \to \infty}{\longrightarrow} 0 \tag{3.20}$$

ya que es suma de una cantidad finita de infinitésimos. ■

La existencia de tal función permite obtener la tasa de grandes desvíos del contenido del buffer, utilizando el principio de contracción (teorema 3.3). El hecho de que los procesos de entrada sean independientes implica que el vector $\frac{1}{N}\mathbf{X}^N[0,t] = \frac{1}{N}(X_1^N[0,t],\ldots,X_M^N[0,t])$ satisface un principio de grandes desviaciones en $\mathbb{R}^M$ con función de velocidades:

$$I_t(x_1,\ldots,x_M) = \sum_{m=1}^{M} I_{m,t}(x_m) \tag{3.21}$$

Se tiene el siguiente:

Teorema 3.6 *En las hipótesis anteriores se tiene que:*

$$\lim_{N\to\infty} \frac{1}{N} \log \mathbf{P}(\textit{overflow en } k) = -\mathbf{I}_k = -\inf\left\{ \sum_{m=1}^{M} I_m(x_m) : g_k(x_1,\ldots,x_M) > C_k \right\} \tag{3.22}$$

donde $I_m(x) = I_{m,1}(x)$.

La demostración de este teorema se basa en utilizar la función continua g_k para expresar, a menos de un infinitésimo, la velocidad de llegada de trabajo al enlace k. A partir de allí se calcula el proceso de cola en términos de las entradas a la red y aplicando el principio de contracción se estima la función de velocidad que permite calcular la probabilidad de overflow.

Conociendo entonces la estadística de los flujos de entrada a la red dada por las funciones I_m y la función g_k que relaciona la tasa de arribo de trabajo al nodo k con las velocidades de entrada, podemos estimar la probabilidad de pérdida del nodo k como $e^{-N\mathbf{I_k}}$, lo cuál se hará para un caso particular en el capítulo 4.

En el mismo trabajo [16] se introduce también una mejora a la aproximación anterior en el caso de que el agregado de tráfico que conforma cada flujo es también superposición de fuentes independientes. Esto es posible utilizando una generalización a dimensiones mayores a uno del teorema de Bahadür - Rao (teorema 3.2). Esta generalización, presentada por M. Iltis en [12] permite obtener el siguiente resultado:

Corolario 3.2 *En las hipótesis anteriores, e imponiendo que los agregados $X_m^N[0,t]$, $m = 1,\ldots,M$ se escriben como superposición de N procesos independientes se tiene que:*

$$\mathbf{P}\left(\textit{overflow en el nodo } k\right) \underset{N\to\infty}{\sim} C_N e^{-N\mathbf{I}_k}(1+O(N^{-1}) \tag{3.23}$$

donde C_N se escribe como:

$$C_N = \frac{d_0}{\sqrt{2\pi N \det(V(x^*))}\,|\alpha|\,(\det(|\alpha|\,(H_h(x^*) - H_f(x^*))))^{1/2}}$$

siendo x^* el punto donde se alcanza el ínfimo de la ecuación 3.22, $\alpha = \nabla I|_{x^*}$, $V(x^*) = (\,Hess(I)|_{x^*})^{-1}$, y H_h y H_f los hessianos de h y f, que son expresiones locales de la frontera de la región factible y de la superficie de nivel de la función I. d_0 es un factor de ajuste que está limitado por el orden del buffer.

Mediante esta corrección es posible mejorar la aproximación de la ecuación 3.22. Para ello, se tomó $d_0 = 1$ y se despreció el término $O(N^{-1})$ obteniéndose una estimación de la forma $C_N e^{-N\mathbf{I}_k}$.

Observación 3.2 *Elegir $d_0 = 1$ corresponde a asumir que las estimaciones realizadas se realizan para redes sin buffers. El resultado de Mazumdar permite afirmar que esta aproximación es buena para aquellas redes en que los buffers sean despreciables frente a la capacidad. Es claro que ante la presencia de buffers mayores, la probabilidad de overflow deberá reducirse, por lo que d_0 es de esperar que sea menor que 1. Sin embargo, elegir otro valor para esta constante no es posible debido a que depende de forma compleja de los buffers de toda la red, no habiendo una expresión cerrada para determinarla.*

De todas formas, la ecuación 3.22 predice que el decaimiento asintótico de la probabilidad de overflow, dado por la tasa de pérdida, es independiente del valor del buffer. Esto constituye otra alternativa de verificación, es decir, aún en aquellos casos en que el tamaño del buffer sea distinto de cero, lo que llevaría a errores en la estimación de la probabilidad de overflow, es posible verificar su decaimiento asintótico.

3.4.2. Ratio de pérdida

En la presente sección vamos a introducir una magnitud conocida como el *ratio de pérdida*, la cual se presenta como una medida de performance de un LSP. Nuestro trabajo se basó en trabajar con redes de caminos virtuales, por lo que lo anterior representa una magnitud de interés. Más aún, desde el punto de vista de la aplicabilidad del presente trabajo, cada uno de los LSP puede representar el flujo de uno o varios clientes, por lo surge la necesidad de la existencia de un parámetro con dichas características.

Como primer paso de van a definir las diferentes magnitudes de interés. Sea $Q_{k,t}^{m,N}$ la cantidad de datos del tipo m en el buffer del nodo k en el tiempo t y $Q_{k,t}^{N} = \sum_{m=1}^{M} Q_{k,t}^{m,N}$. En el tiempo t lo primero que es procesado por el buffer es lo que estaba en el mismo en el tiempo $t-1$, que se representa por $Q_{k,t-1}^{N}$. El sobrante de capacidad del buffer, $NC_k - Q_{k,t-1}^{N}$ estará disponible para el trafico que arribe en el tiempo t, cuya tasa total será $\sum_m X_{k,t}^{m,N}$. El caso de interés es aquel en que la capacidad disponible es menor al tráfico que arriba, o sea, $\left(NC_k - Q_{k,t-1}^{N}\right) < \sum_m X_{k,t}^{m,N}$ es decir, donde una fracción del tráfico que arriba al enlace deberá ir al buffer. Se representa el tamaño del buffer como $B_k(N)$

En el caso que aquella parte del tráfico que no puede ser servido inmediatamente sea menor que el tamaño total del buffer, la misma será toda procesada en dicho intervalo de tiempo. En caso contrario, los diferentes tipos de tráfico se acumularán en el buffer en función de las proporciones en que arriban. Por lo tanto, según la anterior descripción, el contenido del buffer del tipo m en el nodo k para el tiempo t va a variar según

$$Q_{k,t}^{m,N} = \gamma_{k,t} X_{k,t}^{m,N} \frac{B_k(N)}{\text{máx}\left(B_k(N), \gamma_{k,t} \sum_m X_{k,t}^{m,N}\right)}$$

donde

$$1 - \gamma_{k,t} = \frac{\left(NC_k - Q_{k,t-1}^{N}\right)}{\text{máx}\left(NC_k - Q_{k,t-1}^{N}, \gamma_{k,t} \sum_m X_{k,t}^{m,N}\right)}$$

Notar que $\gamma_{k,t}$ es la fracción del tráfico a la entrada del enlace que no puede ser procesada inmediatamente.

Ratio de pérdida Sea $\mathbf{L^{m,N}}$ el ratio de pérdida total (LR) que se define como el cociente entre el valor esperado de unidades de trabajo perdidas en todos los enlaces de un LSP m y la media del tráfico de entrada para dicho LSP.

$$\mathbf{L^{m,N}} = \frac{\sum_{k\in\mathbf{r}^m} L_k^{m,N}}{\mathbf{E}\left(X_t^{m,N}\right)} \tag{3.24}$$

donde $\mathbf{r}^m = \{k_i : i = 1, \ldots, l_m\}$ que representa el LSP m y $L_k^{m,N}$ es el valor esperado de unidades de trabajo perdidas en el nodo k que viene dado por

$$L_k^{m,N} = \mathbf{E}\left[\left(X_{k,t}^{m,N} + Q_{k,t-1}^{m,N} - NC_k - Q_{k,t}^{m,N}\right)^+\right]$$

Se tiene el siguiente resultado que caracteriza las pérdidas que se dan en cada LSP de la red.

Proposición 3.2

$$\lim_{N\to\infty} \frac{1}{N} \log \mathbf{L}^{m,N} = -\min_{k\in\mathbf{r}^m} \mathbf{I}_k$$

La demostración de la proposición se puede ver en el citado artículo [16]. El resultado anterior está de acuerdo con lo que es de esperar en la práctica, es decir, que las pérdidas para un LSP van a estar regidas por las de aquel enlace cuyo buffer está más comprometido. Esto a su vez proporciona una herramienta de gran utilidad, ya que calculando el LR, se podrán estudiar las pérdidas para el LSP en su conjunto, que por lo general es lo que interesa conocer.

3.4.3. Relación entre la probabilidad de overflow y el ratio de pérdida

Debe tenerse cuidado en distinguir claramente los conceptos de probabilidad de overflow y ratio de pérdida, ya que ambos representan parámetros de calidad de servicio distintos.

La probabilidad de overflow en un buffer de una red consiste intuitivamente en la probabilidad de encontrarlo saturado en un instante de tiempo arbitrario. Las hipótesis sobre el proceso de trabajo y el teorema de Loynes (teorema 3.4) permiten definir este concepto claramente. En particular, se define una probabilidad de overflow "en régimen" o estacionaria.

En términos temporales, la probabilidad de overflow estacionaria representa la proporción de tiempo en que el sistema se encuentra saturado (y por consiguiente perdiendo datos).

Utilizando el modelo de muchas fuentes y buffer pequeño puede estimarse con bastante exactitud esta magnitud, utilizando la ecuación 3.23. Esta magnitud reviste especial interés en la etapa de diseño de la red, donde puede desearse un cierto nivel de tiempo de sobrecarga o congestión máximo admitido. A su vez, la sobrecarga es percibida negativamente por los usuarios, afectando a todos aquellos caminos que circulan por el buffer.

La otra magnitud estudiada, el ratio de pérdida, está relacionada con la pérdida de paquetes de cada tráfico particular. Su definición puede expresarse como:

$$LR = \frac{\mathbf{E}\,(\text{Paquetes perdidos en la red})}{\mathbf{E}\,(\text{Paquetes recibidos por la red})} \tag{3.25}$$

En términos temporales puede interpretarse esta magnitud como la proporción de paquetes perdidos del total que llegan a la red, lo que representa la *tasa de pérdida de paquetes*. Bajo las mismas hipótesis de ergodicidad que antes puede probarse que asintóticamente la tasa de pérdida converge al Loss Ratio así definido.

Esta magnitud es un excelente indicador de la calidad de servicio de un LSP o camino en una red. La tasa de pérdida de paquetes influye en todo tipo de transmisión de información, y su valor debe mantenerse lo más bajo posible.

Sin embargo, las herramientas estudiadas no permiten aun obtener estimaciones precisas de la tasa de pérdida de paquetes o Loss Ratio en cualquier enlace de la red. El único resultado de que se dispone a nivel de toda una red fue presentado en el teorema 3.2 y solo hace referencia al comportamiento exponencial del mismo.

Como el objetivo central del proyecto consiste en obtener estimaciones de performance en todos los nodos de la red, se optó por concentrar los esfuerzos en estimar la probabilidad de overflow.

Además, es posible hallar una relación entre la tasa de pérdida y la probabilidad de overflow. Sea L la cantidad de paquetes que se pierden en una unidad de tiempo para cierto flujo, en estado estacionario, y X la tasa de arribo de trabajo en dicha unidad de tiempo. Sea también Z una variable que vale 1 si hay overflow en dicho instante y 0 si no ($Z = \mathbf{1}_{\{overflow\}}$). Entonces se tiene que:

$$LR = \frac{\mathbf{E}(L)}{\mathbf{E}(X)}$$

Y además, utilizando esperanza condicional:

$$\mathbf{E}(L) = \mathbf{E}(\mathbf{E}(L|Z)) = \mathbf{E}(L|Z=0)\,\mathbf{P}(Z=0) + \mathbf{E}(L|Z=1)\,\mathbf{P}(Z=1)$$

Ahora bien, $\mathbf{E}(L|Z=0) = 0$ ya que si no hay desborde, no hay pérdida de paquetes. Usando que $\mathbf{P}(Z=1)$ es la probabilidad de overflow, se puede hallar una relación entre ésta y la tasa de pérdida como:

$$LR = \frac{\mathbf{E}(L|Z=1)}{\mathbf{E}(X)}\,\mathbf{P}(\text{overflow})$$

Por lo tanto, para una cierta configuración de tráfico, la tasa de pérdida es proporcional a la probabilidad de overflow.

Sin embargo, el cálculo de la esperanza del numerador es relativamente complejo, ya que la cantidad de paquetes que se pierden en un cierto buffer depende fuertemente de la función de transferencia g que indica la tasa de arribo de paquetes en régimen al buffer en cuestión. Como la función g puede ser muy compleja y depende de la topología, no se ha desarrollado aún un método que funcione en forma general.

Para el caso de un único buffer al que arriban muchas fuentes *independientes*, Mazumdar y Likhanov ([15]) desarrollaron un método que permite estimar con exactitud la tasa de pérdida para el caso de muchas fuentes, en particular el de muchas fuentes y buffer pequeño. Este caso es útil únicamente en el buffer de ingreso a la red y no en otros, evitándose las dificultades introducidas por la función g.

En el contexto de un único agregado que llega a un buffer, la función de transferencia es la identidad, $g(x) = x$, y el ínfimo de la ecuación 3.22 se reduce a:

$$\mathbf{I} = \text{ínf}\{I(x) : x > c\}$$

siendo I la función de velocidad del tipo de tráfico que arriba y c la capacidad reservada por fuente (la capacidad total es $C = Nc$). Este problema es muy sencillo ya que la función I es creciente para valores de x mayores que la tasa media. Como c es mayor que la media, el mínimo es:

$$\mathbf{I} = I(c)$$

Si s^* es el punto tal que $\Lambda'(s^*) = c$, es decir, el valor del parámetro espacial que relaciona el ancho de banda efectivo con la función de velocidad, entonces la corrección de Bahadur-Rao permite establecer la probabilidad de overflow para un agregado de n fuentes como:

$$\mathbf{P}\left(\text{overflow}\right) \approx \frac{1}{s^* \sqrt{\Lambda''(s^*) 2\pi N}} e^{-N\mathbf{I}}$$

A partir de esta aproximación, los autores hallan la corrección para llegar al Loss Ratio como:

$$LR \approx \frac{1}{{s^*}^2 \mu \sqrt{\Lambda''(s^*)^2 2\pi N^3}} e^{-N\mathbf{I}} \tag{3.26}$$

siendo μ la tasa media del tráfico. La prueba se basa en aproximar la integral que interviene en el cálculo de $\mathbf{E}\left(L|\text{overflow}\right)$. Es de notar que el decaimiento es más rápido que en el caso de la probabilidad de overflow debido al factor N^3.

Lamentablemente, la aplicabilidad de este resultado es reducida, debido a que las hipótesis solo se cumplen en el borde de la red. El estudio de aproximaciones de este tipo que sean posibles en toda la red resulta una tarea interesante, pero que queda fuera del alcance del proyecto, debido a que las herramientas teóricas de que se disponen aún no han avanzado lo suficiente. Algunos posibles caminos para extender estos resultados se discuten en el capítulo 13.

Capítulo 4

Estudio de una red mediante grandes desvíos

En el capítulo anterior se presentó la técnica de evaluación de performance conocida como de muchas fuentes y buffer pequeño. A los efectos prácticos, la principal cualidad de dicha técnica es que permite estudiar la performance de una red dada en cualquier buffer de la misma. A su vez, dicho análisis se realiza a partir de los flujos a la entrada de la red, y posee gran generalidad en cuanto a los tipos de tráfico que verifican las hipótesis de trabajo. Esta técnica es la primera herramienta de análisis en el área de teoría de colas que reúne estas cualidades, lo que la presenta como una metodología de gran proyección a futuro. En este capítulo se realiza una descripción práctica de los resultados, pensando en las aplicaciones.

Es así que en la primera sección se divide la técnica presentada en la sección 3.4, en subproblemas más concretos. Posteriormente se pasa a describir en detalle cada uno de los problemas antes referidos, haciendo hincapié en detalles de su implementación práctica. Luego en la sección 4.2 se describe someramente los modelos de flujo considerados y las razones de su utilización. Finalmente, en la sección 4.3, mediante un ejemplo se aplican los conceptos desarrollados anteriormente.

4.1. Aplicación de la teoría de muchas fuentes y buffer pequeño

Como ya se dijo, esta teoría realiza hipótesis sobre el tráfico que ingresa a la red, para poder deducir mediante técnicas de grandes desviaciones las tasas de pérdida en todos los buffers de la red. La hipótesis más relevante que debe realizarse para que el método sea aplicable es que el tamaño de dichos buffers debe crecer más lentamente que la cantidad de fuentes que llegan a la misma. En términos prácticos, esto quiere decir que los buffers deben ser pequeños frente a la capacidad de los enlaces (que sí crecen como la cantidad de fuentes).

A su vez, el resultado que se obtiene es asintótico, por lo que se requiere una gran cantidad de fuentes ($N \to \infty$) para asegurar la validez de las estimaciones.

Ambas hipótesis están en concordancia con lo que ocurre hoy en día en las redes de gran porte, donde circula una gran cantidad de tráficos, y los buffers se mantienen pequeños para no introducir retardos de cola.

De un modo esquemático, la metodología consiste en:

- Caracterizar cada flujo de tráfico (j) por una función de velocidad $I_j(x)$.

- Para cada buffer de la red, introducir una función $g(x)$ de transferencia entre las tasas de entrada de datos a la red y la tasa total de llegada de datos al buffer.
- Resolver un problema de optimización, en el que intervienen las funciones I_j (de cada flujo) y g, para hallar la tasa de pérdida ($\mathbf{I}$) del buffer en estudio.
- A partir de la solución del problema de optimización, calcular una constante de corrección C, que depende de las funciones I_j y g en un entorno de solución.
- Estimar la probabilidad de overflow en el buffer como:

$$\mathbf{P}(\text{overflow}) \approx \frac{1}{\sqrt{N}} C e^{-N\mathbf{I}} \tag{4.1}$$

  siendo N la cantidad total de fuentes que afectan al buffer en cuestión.

La hipótesis antes introducida sobre los tamaños de los buffers produce que los resultados no dependan del tamaño del mismo.

Función de velocidad

La función $I_j(x)$ se denomina *función de velocidad* y se define en la ecuación 3.15. La misma resume las características estadísticas del tráfico j. Proviene de la teoría de grandes desviaciones estudiada en 3. Dicha función pondera las diferentes tasas de llegada de tráfico de acuerdo a su probabilidad de ocurrencia. Cuánto mayor es el valor de $I_j(x)$, menos probable es observar el valor x de la tasa de llegada. El nombre función de velocidad hace referencia a que en le teoría de grandes desviaciones se prueba que la probabilidad de observar eventos alejados de la media decae exponencialmente con una constante o velocidad de decaimiento que depende de I.

Como la anterior función depende de la estadística del tráfico, en el caso que la misma no sea conocida, deberá estimarse a partir de observaciones. Este problema ha sido abordado en diferentes trabajos, como ser [19], [10] y [6]. En concordancia con el objetivo del proyecto, es decir basar los estudios en trazas de tráfico, se utilizaron algunos resultados existentes para la estimación de la misma, lo que se discute en el capítulo 5.

Función de transferencia

La función g es el principal aporte del trabajo de Mazumdar et al La misma describe la relación entre las tasas de entrada de tráfico a la red y la tasa de arribo de trabajo a un determinado enlace a estudiar. Puede calcularse recursivamente a partir de la función f, la cual se presenta en la ecuación 3.18.

Esta función f permite hallar la velocidad de salida de uno de los tráficos de un enlace a partir de todas las velocidades a la entrada de éste. En ésta se dividen dos posibilidades según el tráfico a la entrada supere o no la capacidad del enlace, como se ve en la expresión de la misma.

Como es de esperar, si el tráfico total a la entrada no supera la capacidad, no se verá afectado por el enlace, y la velocidad a la salida será igual la entrante. En el caso contrario, es decir donde la capacidad se vea desbordada debido a los flujos a la entrada la velocidad de salida del enlace en cuestión será igual a la capacidad del mismo. En este caso las velocidades de salida de cada unos de los flujos entrantes viene dada por la proporción en que éstos ingresan al enlace.

Por lo tanto, para poder conocer las tasas de tráfico que arriban a un determinado enlace, será necesario calcular las tasas de salida de los enlaces que alimentan al mismo, y así sucesivamente hasta las entradas a la red.

Problema de optimización

La parte central del resultado de Mazumdar et al se resume en la ecuación 3.22 que permite calcular la constante **I**. Esta ecuación tiene una interpretación intuitiva, el conjunto de interés para las pérdidas es aquel donde se ve desbordada la capacidad, ya que la hipótesis de buffer pequeño permite despreciar el contenido del mismo en el análisis. En función de los valores de tasa de entrada a la red esto se da en aquellos puntos en que la función de transferencia del buffer en cuestión (g) supera la capacidad del enlace asociado.

En su caso más general, de trata de un problema de optimización no lineal en varias variables, con restricciones no lineales. A su vez, tanto la función objetivo como la función de las restricciones no tienen una expresión analítica sino que dependen de la topología red en estudio. Por último, la función objetivo depende de la estadística de las trazas y además en nuestro caso será necesario estimarla a partir de mediciones del tráfico, lo que puede redundar en una costosa evaluación de la misma.

Debido a las características del problema a resolver, el mismo representó un desafío no menor e insumió gran parte del tiempo de desarrollo del proyecto. El método de resolución que se implementó se presenta en el capítulo 6.

Coeficiente de corrección

Como se muestra en la ecuación 4.1, la estimación de la probabilidad de overflow de un determinado enlace, incluye una constante C, que surge de teoremas refinados de grandes desviaciones. Como se puede apreciar en la ecuación 3.2, la misma se calcula a partir de las derivadas de las funciones I y g por lo que no presenta dificultades en su cálculo. Las derivadas pueden ser estimadas mediante cocientes incrementales. En la sección 8.7 se muestran los detalles prácticos de su implementación en software.

Estimación de la probabilidad de overflow

Luego de llevar a cabo todos los pasos anteriores, solo resta la estimación de la probabilidad de pérdida en sí, que como ya se adelantó se basa en la fórmula 4.1.

4.2. Modelos de tráfico

Como se observó en las secciones anteriores, la información sobre el comportamiento estadístico del tráfico se condensa a través de la denominada función de velocidad (I). Si el tráfico sigue algún modelo conocido, entonces pueden deducirse ciertos parámetros de comportamiento para la función I.

El modelado del tráfico que circula por las redes actuales es un área en desarrollo del tratamiento de señales. Se han planteado diversos modelos, con diferentes tipos de distribuciones y dependencia en el proceso.

Un modelo particularmente versátil y que admite un tratamiento analítico desde el punto de vista de grandes desvíos es el modelo del Flujo Markoviano. El mismo se estudia exhaustivamente en [3] y sus aplicaciones a redes se discuten en [13] y [4]. Debido a su aplicabilidad y el que permite obtener algunas fórmulas explícitas, haremos aquí una breve descripción del mismo.

Un Flujo markoviano es un proceso de arribo de trabajo en tiempo continuo, que está modulado por una cadena de Markov. Una cadena de Markov es un proceso estocástico que toma una cantidad discreta de valores denominados *estados*. Su propiedad principal es que la estadística de la trayectoria después del tiempo t depende únicamente del estado que ocupa el proceso en t, y no de la historia previa.

Esta propiedad permite deducir que la estadística de las transiciones entre un estado y otro puede resumirse a través de una matriz Q, denominada *generador infinitesimal* de la cadena. Las entradas q_{ij} de la matriz representan la cantidad media de transiciones por unidad de tiempo entre el estado i y el estado j, y los elementos de la diagonal q_{ii} están relacionados con el tiempo medio que el proceso permanece en el estado i.

En régimen, la probabilidad de que una cadena (con cantidad finita de estados) se encuentre en el estado i es π_i, donde el vector $\pi = (\pi_1, \ldots, \pi_K)$ cumple la ecuación de balance:

$$\pi Q = \vec{0}$$

Un Flujo Markoviano es simplemente un proceso de arribo de trabajo en el cual la tasa de llegada de trabajo depende del estado de una cadena de Markov. En cada estado i, $1 \leq i \leq K$ se produce trabajo a una tasa $h_i \geq 0$ constante.

La ventaja de este modelo es que la función ancho de banda efectivo admite una expresión explícita en términos de la matriz Q, el vector π y las tasas h_i.

$$\alpha(s,t) = \frac{1}{st} \log\left(\pi \exp((Q + Hs)t)\mathbf{1}\right) \tag{4.2}$$

donde H es una matriz con los valores h_i en la diagonal y $\mathbf{1}$ es un vector columna de unos.

Un caso particular de especial aplicación es el denominado *Flujo ON OFF*. En este caso, se considera un flujo markoviano de dos estados, uno de los cuales produce trabajo a tasa constante, y el otro no produce trabajo.

Los parámetros de este modelo son λ, la tasa media de transiciones de OFF a ON, μ, la tasa media de transiciones de ON a OFF y h, la tasa de llegada de trabajo cuando está ON. En función de estos parámetros se tiene:

$$\begin{aligned} Q &= \begin{pmatrix} -\lambda & \lambda \\ \mu & -\mu \end{pmatrix} \\ H &= \begin{pmatrix} 0 & 0 \\ 0 & h \end{pmatrix} \end{aligned}$$

y de la ecuación de balance se obtiene que:

$$\pi = \left(\frac{\mu}{\lambda+\mu}, \frac{\lambda}{\lambda+\mu}\right)$$

Otra forma de interpretar este modelo es a través de los tiempos de permanencia. El tiempo de permanencia en cada estado es exponencial, de media $\frac{1}{\lambda}$ para el OFF, y de media $\frac{1}{\mu}$ para el ON. Como las transiciones solo pueden ser de ON a OFF y viceversa, el proceso consiste en una sucesión de tiempos exponenciales de ON y OFF alternados, y durante el tiempo ON se transmite a una tasa h.

A partir de la función $\alpha(s,1)$ puede obtenerse la función de velocidad $I(x)$ utilizando la definición (ecuación 3.15). Si bien no existe una expresión explícita para la misma, la solución de la ecuación puede calcularse numéricamente a partir de la expresión de α de manera rápida.

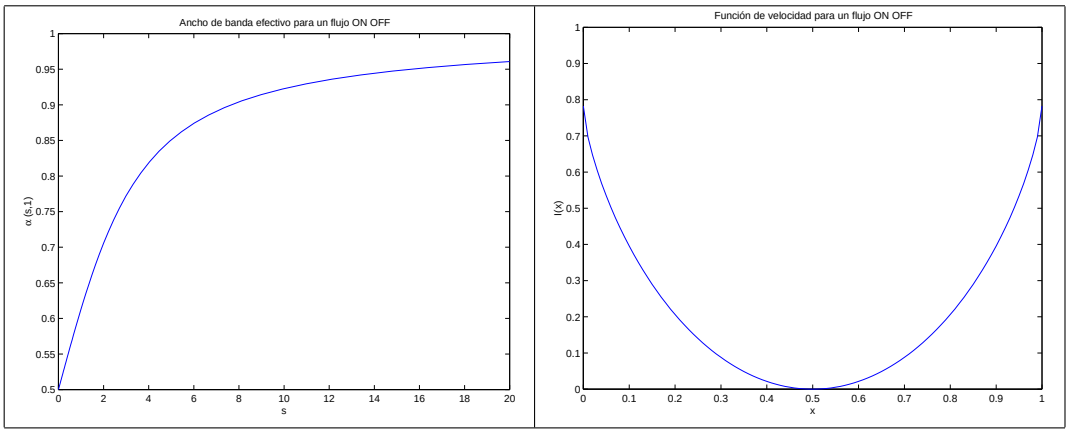

Figura 4.1: Ancho de banda efectivo y función de velocidad de un Flujo ON OFF

A modo de ejemplo, se adjuntan en la figura 4.1 las gráficas de $\alpha(s,1)$ e $I(x)$ para un flujo ON OFF de parámetros $\lambda = \mu = 0{,}1$ y $h = 1$. En general, la forma es la misma para diferentes juegos de parámetros, cambiando únicamente la escala de variación de s y x.

4.3. Ejemplo: el caso de dos enlaces

Como una primera aproximación a la resolución del problema de optimización derivado de la ecuación 3.22, estudiamos un caso simple, de una red con dos enlaces y tres LSP, como se muestra en la figura 4.2.

Vamos a centrar nuestro estudio en el nodo 2, estudiando como se comporta la probabilidad del pérdida en función de los flujos de entrada a la red. Por lo tanto una magnitud de interés será el el tráfico instantáneo a la entrada del nodo en cuestión, que vendrá dada por la función $g_2(x_1, x_2, x_3)$ definida en la ecuación 3.19. Para el caso en estudio el tráfico que arriba al buffer del segundo enlace, tendrá dos componentes, aquél proveniente del LSP 3 y por otro lado el tráfico a la salida del Nodo 1. Éste a su vez vendrá dado por la función $g_1(x1, x2)$. Por lo tanto, la expresión para el tráfico instantáneo que arriba al nodo 2 será

$$g(x_1, x_2, x_3) = x_3 + \frac{x_2\, C_1}{\text{máx}\,\{x_1 + x_2, C_1\}}$$

El principal problema a resolver para estudiar la performance de la red será resolver el problema de optimización de la ecuación 3.22. La función g_2 antes calculada representa las restricciones de dicho problema, donde la región factible es $g(x_1, x_2, x_3) \geq C_2$ y la misma se observa en la figura 4.3.

Para este ejemplo se utilizó C_1 al 80 % del tráfico máximo y C_2 al 60 %.

La función objetivo surge de la ecuación 3.21 y la misma se compone de la suma de las tres funciones de velocidad de cada uno de los flujos que atraviesan la red.

$$I(x_1, x_2, x_3) = I_1(x_1) + I_2(x_2) + I_3(x_3)$$

donde la función I_j es la función de velocidad del tráfico X_j. Para el caso de estudio, en que se supone que cada flujo esta compuesto por una superposición de fuentes markovianas ON OFF independientes, la función de velocidad es la del teorema de Cramér:

$$I_j(x) = \sup_s \{sx - \Lambda(s)\}$$

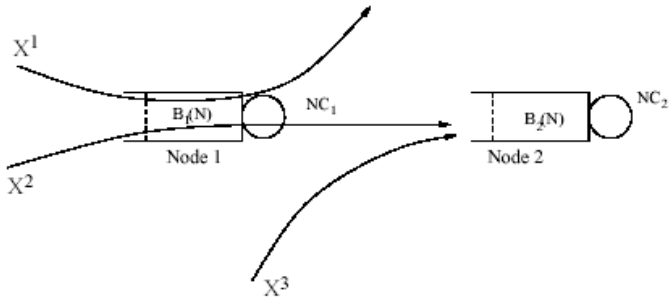

Figura 4.2: Red con dos enlaces

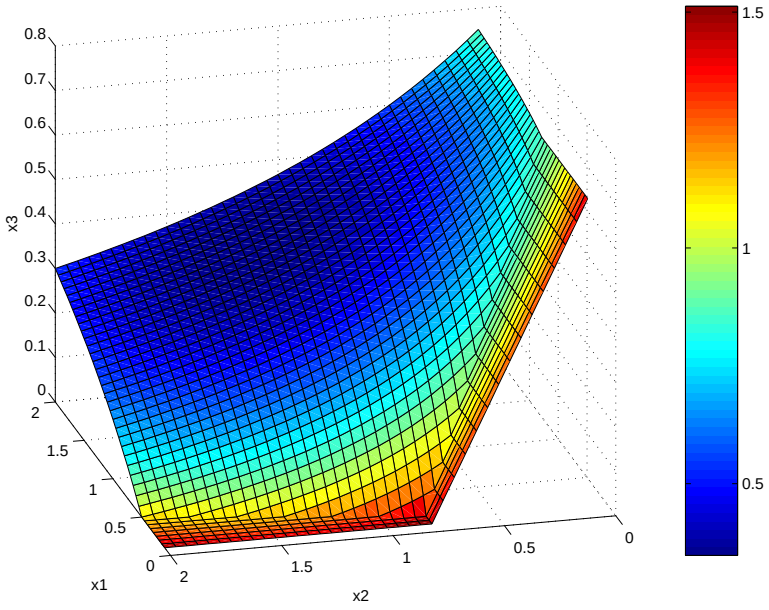

Figura 4.3: Frontera de la región factible para el problema de optimización y valores de la función objetivo

y es de la forma vista en la figura 4.1. Si no se conoce el modelo, la función $\Lambda(s)$ se puede estimar a partir de trazas por métodos que se presentarán en el capítulo 5.

La solución al problema, junto a una comparación con resultados de simulaciones, se presentará en el capítulo 11, luego de introducir las herramientas con las que se resolverá el mismo, lo que se realiza en los capítulos sucesivos.

Capítulo 5

Estimación de la función de velocidad

Uno de los principales problemas planteados para resolver la estimación de la probabilidad de overflow por el método discutido anteriormente es el de conocer la función de velocidad de cada flujo. Uno de los objetivos del proyecto es poder realizar las estimaciones a partir de observaciones del tráfico, y no de simulaciones. Esto vuelve necesario desarrollar estimadores de la función de velocidad de un proceso estacionario a partir de realizaciones del mismo. Como surge de la definición (ecuación 3.15), la función de velocidad está estrechamente ligada a la función ancho de banda efectivo, por lo que encontrar estimadores de éste permite llegar a los estimadores deseados para la función de velocidad.

El contenido de este capítulo está basado en gran parte en trabajos desarrollados por el Grupo ARTES (IIE-IMERL). En la primera sección se presenta el problema global de estimación definiendo qué magnitudes es necesario estimar. En las siguientes secciones se desarrollan los estimadores para dichas magnitudes y se analiza su comportamiento.

5.1. El problema de estimación

El problema que se ataca en este capítulo puede plantearse de la siguiente manera. Sea $\{X_t : t \geq 0\}$ un proceso estocástico estacionario, de tiempo discreto. Este proceso representa la tasa de arribo de trabajo desde una cierta fuente.

Para este proceso se definió en capítulos anteriores el ancho de banda efectivo:

$$\alpha(s,t) = \frac{1}{st} \log \mathbf{E}\left(e^{sX_t}\right)$$

y la función de velocidad:

$$I_t(x) = \sup_s \{sx - st\alpha(s,t)\}$$

En el análisis subsiguiente, tomaremos $t = 1$, debido a que ésta es la escala de tiempo relevante a los efectos del análisis mediante la asintótica de muchas fuentes y buffer pequeño. Esto además no es una pérdida de generalidad ya que los resultados pueden adaptarse a otras escalas.[1] Adoptaremos la notación:

$$M(s) = \mathbf{E}\left(e^{sX}\right) \tag{5.1}$$

[1] Esto es equivalente a un remuestreo adecuado del proceso de llegada de trabajo.

$$\Lambda(s) = \log \mathbf{E}\left(e^{sX}\right) \tag{5.2}$$

$$I(x) = \sup_s \{sx - \Lambda(s)\} \tag{5.3}$$

En estas ecuaciones, X representa una llegada típica de trabajo (por ejemplo X_0). El proceso es estacionario por lo que estas magnitudes son independientes de t.

El objetivo es conseguir un estimador de la función I (es decir, del valor $I(x)$ para cada x) a partir de una realización del proceso X_t, es decir, de una traza de tráfico.

Más específicamente, se dispone de muestras $\{x_t : 0 \leq t \leq n\}$ y se busca obtener, para cada x un estimador $I_n(x)$ que dependa de dichas muestras.

El principal problema en esta estimación es que la función I es una transformación compleja de la función $\Lambda(s)$, y no es trivial su estimación directamente a partir de los datos. Es por eso que se propone seguir un camino indirecto, en el cual se estime la función Λ como $\Lambda_n(s)$ y a esta función se le aplique la transformación de la ecuación 5.2, es decir:

$$I_n(x) = \sup_s \{sx - \Lambda_n(s)\} \tag{5.4}$$

A su vez, el valor de la tasa de pérdida de un cierto buffer ($\mathbf{I}$) que sale de resolver la ecuación 3.22 será estimado por un cierto valor $\mathbf{I}_n$ que surgirá de resolver la misma ecuación pero sustituyendo las funciones $I(x)$ por sus estimaciones, es decir:

$$\mathbf{I}_n = \text{ínf}\left\{\sum_{m=1}^{M} I_n^{(m)}(x_m) : g(x_1, \ldots, x_M) > C\right\} \tag{5.5}$$

donde $I_n^{(m)}$ es la estimación de I_m a partir de los datos, g la función de transferencia y C la capacidad del enlace en cuestión.

Esto presenta a su vez varios subproblemas que se discuten a lo largo de capítulo:

- Obtener un estimador de las funciones $M(s)$ y $\Lambda(s)$, es decir del ancho de banda efectivo, a partir de los datos. A su vez, determinar el orden del error en la estimación.
- Verificar en qué condiciones el estimador $I_n(x)$ de la ecuación 5.4 converge al verdadero valor $I(x)$ y el orden del error.
- Verificar en qué condiciones es posible estimar el valor de $\mathbf{I}$ resolviendo la ecuación 5.5.

A continuación se encaran uno a uno estos problemas.

5.2. Estimación del ancho de banda efectivo

La estimación del ancho de banda efectivo (o bien de $M(s)$ o $\Lambda(s)$) puede enfocarse desde dos puntos de vista distintos. En primer lugar, puede interesar buscar estimadores que puedan ser utilizados para diversos modelos de tráfico indistintamente, sin hacer hipótesis restrictivas sobre los mismos. Este enfoque se denomina *no paramétrico*. Su principal virtud es también su mayor defecto, ya que su generalidad hace que la convergencia sea más lenta, haciendo necesarias muestras mayores. El segundo enfoque se denomina *paramétrico* y consiste en admitir un modelo para el tráfico en estudio, y ajustar ciertos parámetros del modelo en función de las observaciones. Como en el estudio intervienen las hipótesis del modelo, los estimadores así construidos pierden generalidad pero ganan en precisión.

En el resto de la sección se analiza el problema de estimar el ancho de banda efectivo mediante estos dos enfoques. En particular, se analiza el modelo de Flujo Markoviano discutido en la sección 4.2, que es un modelo paramétrico y de extensa aplicabilidad en el modelado de las fuentes que se utilizan actualmente en la práctica.

5.2.1. Enfoque no paramétrico

El enfoque no paramétrico consiste en aplicar teoremas de validez general en la estimación de las magnitudes que intervienen. En nuestro caso, interesa estimar la magnitud $M(s) = \mathbf{E}\left(e^{sX}\right)$, que es la media de una variable aleatoria. Para ello se dispone de observaciones $\{x_t : 0 \leq t \leq n\}$ de un proceso estacionario. El estimador más simple que puede proponerse consiste simplemente en tomar:

$$M_n(s) = \frac{1}{n}\sum_{i=1}^{n} e^{sx_i} \tag{5.6}$$

Interesa estudiar si $M_n(s) \to M(s)$ para cada s cuando $n \to \infty$. Sin embargo, la convergencia de este estimador (un equivalente a la ley de los grandes números) no es inmediata en situaciones en las que hay dependencia entre las observaciones. Este es además el caso más común, ya que suponer que las cantidades de datos que arriban en intervalos consecutivos de tiempo son independientes no es una hipótesis razonable en la mayoría de las redes donde estos resultados son aplicables.

Sin embargo, existen generalizaciones de la ley de los grandes números a procesos *estacionarios* con hipótesis de *dependencia débil*. En particular, el estimador 5.6 es estudiado en detalle en [19], [6]. En [2] se prueba además que el error de estimación cometido por éste es asintóticamente gaussiano (teorema central del límite). Las hipótesis de dependencia débil son una generalización bastante amplia del concepto de independencia, por lo que la aplicabilidad de los resultados se vuelve mucho mayor.

A partir de este estimador, es directo obtener estimaciones de la función $\Lambda(s)$ y del ancho de banda efectivo. Simplemente se definen los estimadores:

$$\Lambda_n(s) = \log M_n(s) = \log\left(\frac{1}{n}\sum_{i=1}^{n} e^{sx_i}\right) \tag{5.7}$$

$$\alpha_n(s) = \frac{\Lambda_n(s)}{s} = \frac{1}{s}\log\left(\frac{1}{n}\sum_{i=1}^{n} e^{sx_i}\right) \tag{5.8}$$

Como las transformaciones aplicadas son continuas[2] la convergencia de M_n implica la de los otros estimadores.

Pasemos a estudiar el error cometido en la estimación en algunos casos de referencia. Tomaremos como medida del error al *error cuadrático medio*, definido para un estimador genérico T_n como:

$$ECM(T_n) = \mathbf{E}\left((T_n - T)^2\right)$$

Siendo T la magnitud a estimar. Más exactamente, utilizaremos la versión relativa o normalizada del mismo, para poder comparar entre distintos órdenes de magnitud:

$$ECMR(T_n) = \frac{\mathbf{E}\left((T_n - T)^2\right)}{T^2}$$

[2]En el caso de la ecuación 5.8 esto es cierto para $s \neq 0$ pero el caso $s = 0$ es trivial ya que coincide con la media del tráfico.

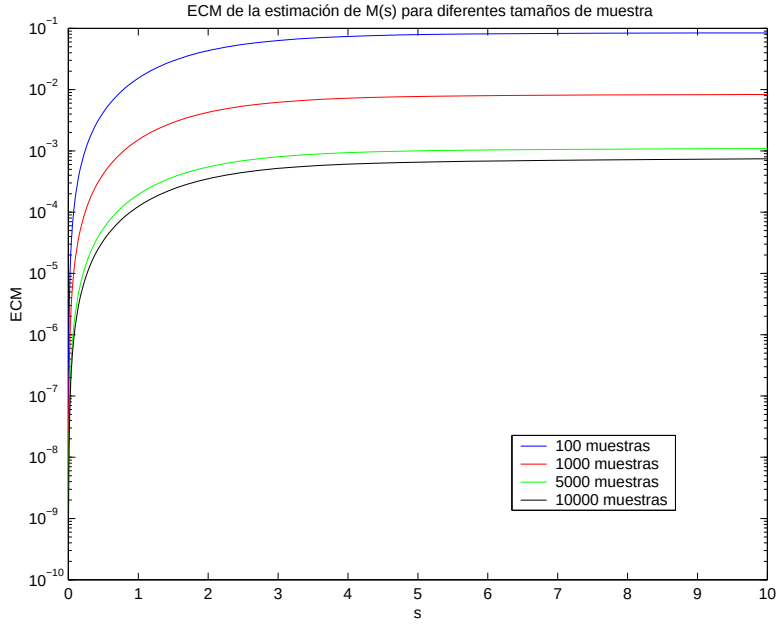

Figura 5.1: Error de estimación en $M(s)$

Para estimar el $ECMR$ se simularon realizaciones de un flujo Markoviano ON OFF de parámetros conocidos mediante la herramienta NS-2. Como se estudió en 4.2, para este modelo existe una fórmula explícita para M, Λ y α, lo que permite conocer el valor a estimar. Se realizaron muestras de distintos tamaños y se estimó el error medio cometido en cada valor de s promediando entre las trazas simuladas. Los resultados se grafican en las figuras 5.1, 5.2 y 5.3 para cada una de las magnitudes involucradas.

Analizando las figuras, se observa un comportamiento similar en todas las magnitudes. Es de notar que el $ECMR$ baja un orden de magnitud si la muestra aumenta en un orden, lo que coincide con la teoría para el caso de M_n donde puede probarse que el error cae aproximadamente como $1/n$. El mismo resultado se extiende a todos los estimadores.

Como conclusión, se observa que el error es relativamente uniforme en todos los valores de s, y que baja de 10^{-4} a partir de las 10000 muestras. Como el error medido es cuadrático, se deduce que habrá al menos 2 cifras de precisión a partir de esta cantidad de muestras, de lo que surge que es recomendable para este estimador trabajar con trazas a partir de este largo. Asimismo, se modificaron los parámetros del flujo markoviano obteniéndose en todos los comportamientos similares.

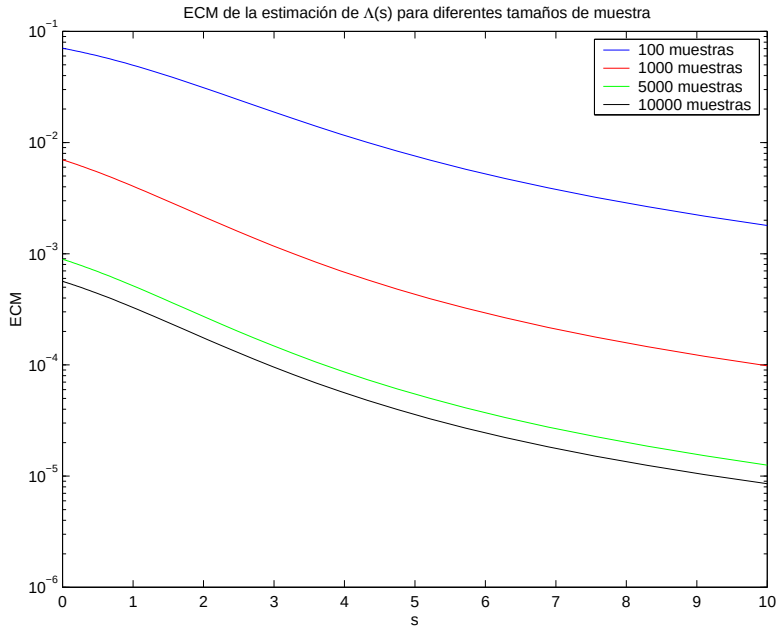

Figura 5.2: Error de estimación en $\Lambda(s)$

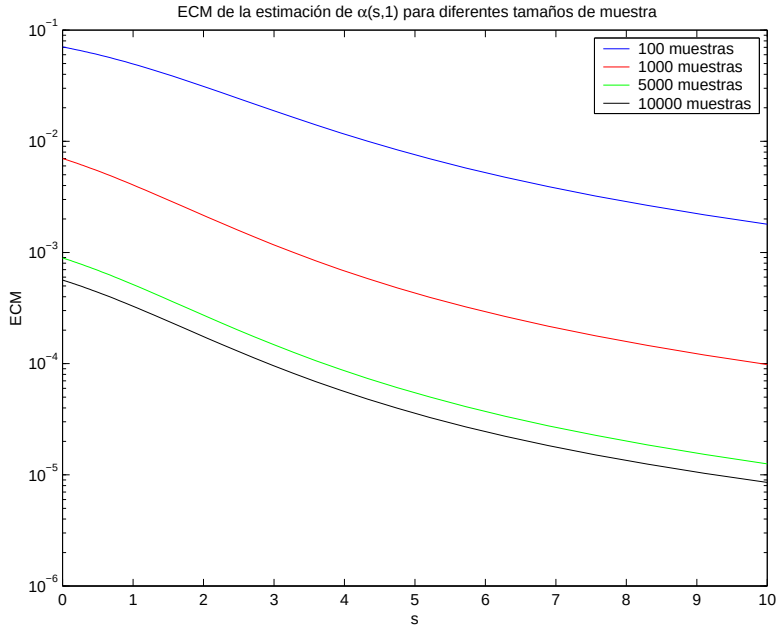

Figura 5.3: Error de estimación en $\alpha(s, 1)$

5.2.2. Enfoque paramétrico: el flujo markoviano

En el caso del flujo markoviano, es posible realizar un enfoque paramétrico de la estimación de las magnitudes de interés. Esto se debe a que existen fórmulas explícitas (ver sección 4.2)) que permiten calcular dichas magnitudes a partir de los parámetros de la cadena de markov que modula el proceso, que son la matriz Q y el vector π, y de los valores de las tasas de transmisión h_i en cada estado (matriz H), que supondremos conocidos. Este enfoque se presenta en [18] y se encuentra desarrollado en [3].

De lo visto en la sección 4.2 y utilizando la notación de este capítulo se tiene que:

$$M(s) = \pi \exp(Q + Hs)\mathbf{1} \tag{5.9}$$

El elemento clave es poder estimar las entradas de la matriz Q a partir de una muestra $\{x_t : 0 \leq t \leq n\}$ del proceso. Para ello, se utilizan los estimadores de Lebedev-Lukashuk:

$$q_{ij}^{(n)} = \frac{\text{Cant. transiciones } i \to j \text{ en la muestra}}{\text{Tiempo en el estado } i} \quad i \neq j \tag{5.10}$$

$$q_{ii}^{(n)} = -\sum_{j \neq i} q_{ij}^{(n)} \tag{5.11}$$

A partir de estos estimadores se construye un estimador Q_n de la matriz Q, que solo depende de las observaciones. A partir de la relación $\pi Q = 0$ se construye un estimador π_n de π resolviendo el sistema $\pi_n Q_n = 0$ con la Q_n estimada. Por último se construye:

$$M_n(s) = \pi_n \exp(Q_n + Hs)\mathbf{1} \tag{5.12}$$

que es sustituir en la ecuación 5.9 los valores estimados. Pueden estimarse también las otras magnitudes sustituyendo en las fórmulas que las definen:

$$\Lambda_n(s) = \log\left(\pi_n \exp(Q_n + Hs)\mathbf{1}\right) \tag{5.13}$$

$$\alpha_n(s) = \frac{1}{s}\log\left(\pi_n \exp(Q_n + Hs)\mathbf{1}\right) \tag{5.14}$$

Este procedimiento produce un estimador M_n que converge a M en todos los valores de s, y además el error cometido es asintóticamente gaussiano.

Una ventaja sumamente importante de este estimador es el hecho de que la información de la traza se resume completamente en las estimaciones de los parámetros. Éstas se realizan una única vez y luego se dispone de una fórmula de rápida evaluación para hallar $M_n(s)$. Esto lo diferencia del estimador no paramétrico 5.6 que debe realizar un promedio de todas las observaciones para cada valor de s, lo que lo vuelve mucho más lento de evaluar.

Resta estudiar el error cometido en la estimación. Para ello se retomó el ejemplo utilizado en la estimación no paramétrica y se calculó para este estimador el error cuadrático medio definido antes. Los resultados se presentan en la figura 5.4.

Se observa rápidamente que este estimador produce resultados más exactos. Se observa una precisión de dos cifras a partir de las 1000 muestras, lo que permite trabajar con trazas más pequeñas.

Todas las ventajas enumeradas justifican el utilizar este estimador cuando puede hacerse la hipótesis de que el tráfico es un flujo markoviano. Esto es tenido en cuenta más adelante en la implementación del software (sección 8.5).

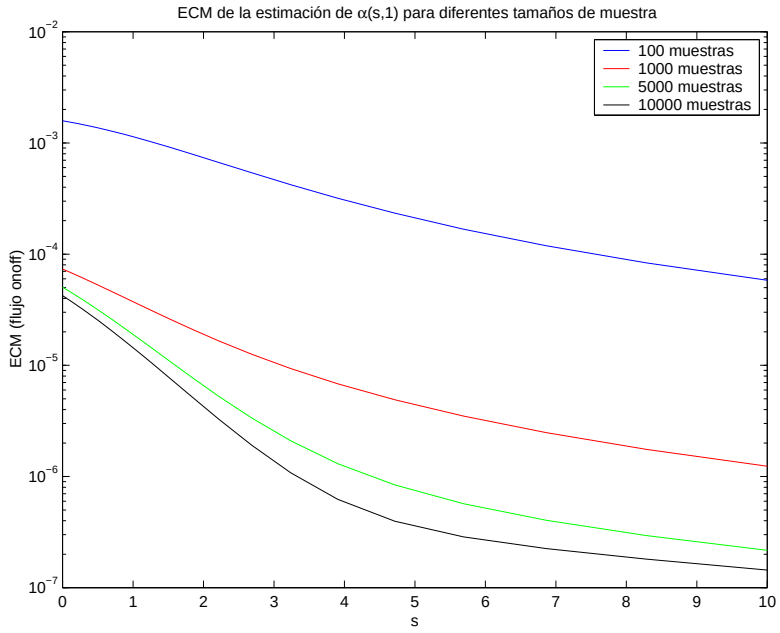

Figura 5.4: Error de estimación en $\alpha(s, 1)$ por la vía paramétrica

5.3. Estimación de la función de velocidad a partir del ancho de banda efectivo

El siguiente paso consiste en encontrar un estimador de la función de velocidad $I(x)$. Para ello, el camino a seguir ya fue planteado en la ecuación 5.4, y consiste en calcular el supremo de la definición de I sustituyendo $\Lambda(s)$ por una estimación.

El que este procedimiento provea un estimador válido no es trivial, y en [10] se estudia el mismo en detalle. Allí se demuestra que si $\Lambda_n(s) \to \Lambda(s)$ y $\Lambda'_n(s) \to \Lambda'(s)$ cuando $n \to \infty$, entonces $I_n(x) \to I(x)$. Esta hipótesis es verificada por lo estimadores de Λ estudiados en la sección anterior, por lo que esto no requiere de introducir hipótesis adicionales.

Además, en dicho trabajo se discute que la función $I_n(x)$ conserva varias de las propiedades de la función I real: también es no negativa, convexa y tiene un único mínimo en la media empírica de las observaciones, donde vale 0.

A los efectos prácticos es difícil hallar directamente el valor de $\sup_s \{sx - \Lambda_n(s)\}$ por lo que se buscó otro enfoque equivalente. Alcanza observar que si el valor s^* donde se alcanza el supremo es finito, y la función Λ es derivable, se verifica la ecuación:

$$x - \Lambda'(s) = 0$$

Si se dispone de una estimación de la derivada, Λ'_n, se divide el problema en dos pasos. Primero se estima s^* como:

$$s_n^* : x - \Lambda'_n(s_n^*) = 0 \tag{5.15}$$

y luego se estima $I(x)$ como:

$$I_n = s_n^* x - \Lambda(s_n^*) \tag{5.16}$$

Como estimación de la derivada hay varias alternativas. Si se dispone de una estimación de Λ puede estimarse Λ' mediante un cociente incremental. Este es el enfoque

adecuado para el caso del flujo markoviano, donde se dispone de una fórmula sencilla de cálculo por lo que las evaluaciones de Λ_n no son costosas. En el caso no paramétrico, se puede estimar la derivada observando que:

$$\Lambda'(s) = \left(\log(\mathbf{E}\left(e^{sX}\right))\right)' = \frac{\mathbf{E}\left(Xe^{sX}\right)}{\mathbf{E}\left(e^{sX}\right)}$$

por lo que puede estimarse Λ' por un cociente de promedios:

$$\Lambda'_n(s) = \frac{\sum_{i=1}^{n} x_i e^{sx_i}}{\sum_{i=1}^{n} e^{sx_i}} \tag{5.17}$$

Para la resolución de la ecuación 5.15 se utiliza el método de la secante ([7]) para la resolución de ecuaciones no lineales. Este método permite hallar rápidamente el valor buscado, si se parte de un punto inicial adecuado. Para el punto inicial, puede utilizarse el que surge de aproximar la función Λ verdadera (desconocida) por una primera aproximación a partir de la media y la varianza (ver [13]) dada por:

$$\begin{aligned} \Lambda(s) &= \mu s + \frac{1}{2}\sigma^2 s^2 \\ \Lambda'(s) &= \mu + \sigma^2 s \end{aligned}$$

lo que da un punto inicial $s_0 = (x - \mu)/\sigma^2$.

Como observación final, hay que resaltar que todas las hipótesis planteadas en esta sección no resultan restrictivas, ya que son válidas para todos aquellos casos en que la variable X esté acotada, lo cual es perfectamente acorde con el hecho de que X es la cantidad de trabajo que llega en un intervalo de tiempo finito.

Para evaluar la performance del estimador se continuó con el ejemplo de la sección anterior, donde se trabajó con un flujo markoviano, simulado en NS-2. En la figura 5.5 se presenta la estimación lograda para 50000 muestras utilizando el estimador no paramétrico del ancho de banda. A su vez, en la figura 5.6 se presenta la estimación para la misma realización del proceso utilizando el estimador paramétrico. Ambos muestran una excelente aproximación. En el caso utilizado, 50000 muestras corresponden a aproximadamente 4 minutos de tráfico.

Para observar la influencia del tamaño muestral se estudió la raíz cuadrada del error cuadrático medio para todo el recorrido de la función de velocidad. En este caso no se utilizó el error relativo debido a que la magnitud en estudio puede ser muy cercana a 0. Los resultados se observan en las figuras 5.7 y 5.8 para los estimadores no paramétricos y paramétricos respectivamente.

De dichas gráficas se concluye que ambos estimadores tienen aproximadamente la misma performance, cometiendo errores similares para los mismos tamaños muestrales. La gran ventaja del estimador paramétrico es que el tiempo de cálculo insumido es dos órdenes de magnitud menor que en el caso no paramétrico, debido a la rápida estimación del ancho de banda.

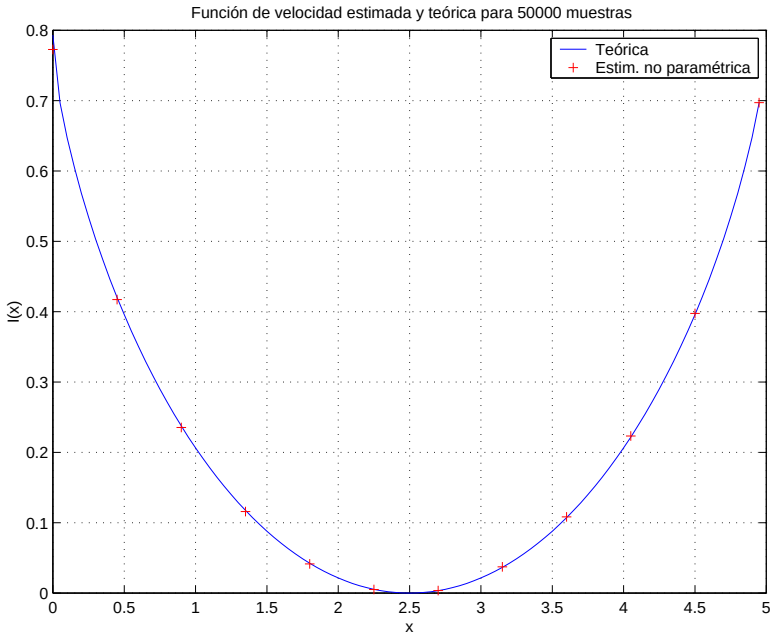

Figura 5.5: Error de estimación no paramétrica en la función $I(x)$

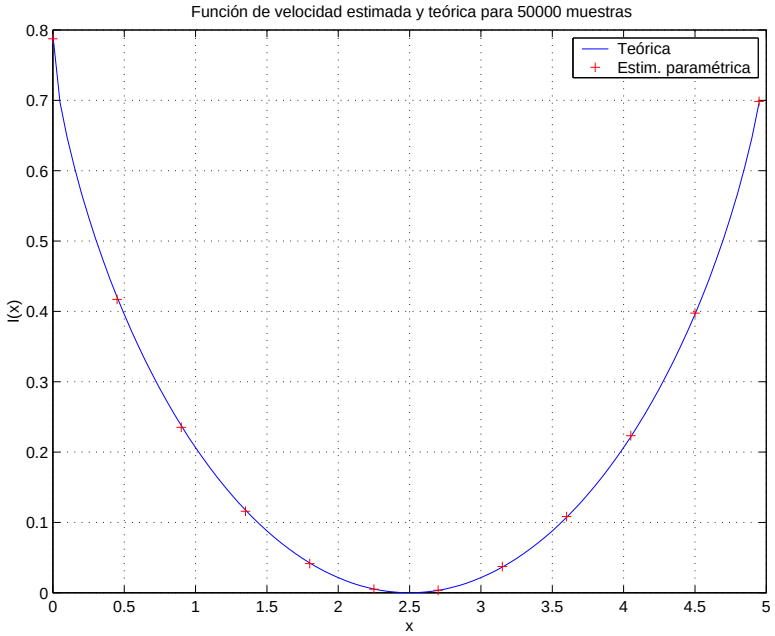

Figura 5.6: Error de estimación paramétrica en la función $I(x)$

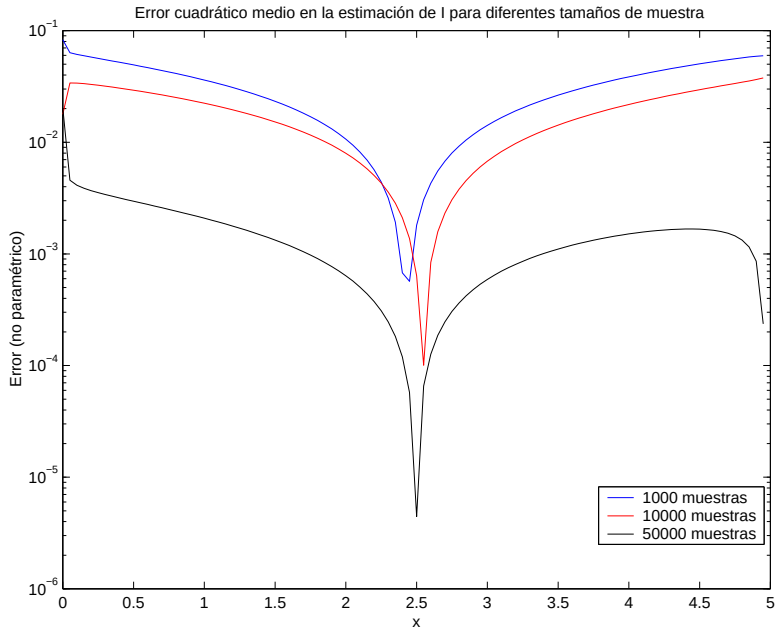

Figura 5.7: Error de estimación no paramétrica en la función $I(x)$

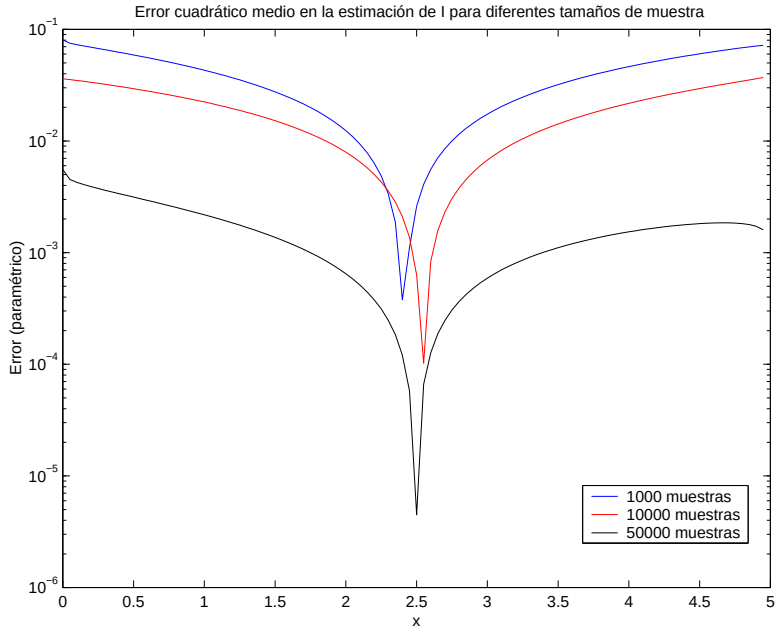

Figura 5.8: Error de estimación paramétrica en la función $I(x)$

5.4. Estimación de la tasa de pérdida

Disponiendo de un modo de estimar la función $I(x)$, dado por el estimador $I_n(x)$, se estima $\mathbf{I}$, la tasa de pérdida en un enlace a través de la ecuación 5.5. Este procedimiento es válido por un argumento similar al utilizado para demostrar que I_n estima I.

Como los estimadores construidos para la función de velocidad conservan las principales propiedades de ésta, debemos utilizar algún algoritmo de minimización apropiado para resolver el problema con restricciones que se presenta. En el capítulo siguiente se ataca este problema.

Los resultados de esta estimación son el objetivo central de este proyecto, por lo que posponemos la discusión de los resultados que se obtienen hasta las conclusiones del presente documento.

Capítulo 6

Problema de optimización

Como se explicó anteriormente, una de las etapas más importantes de la estimación de la probabilidad de pérdida de un determinado enlace, consiste en la resolución de un problema de optimización no lineal de varias variables, con restricciones no lineales. A su vez, tanto la función de las restricciones como la función objetivo son de difícil evaluación, y no se conoce una forma explícita para ellas. Lo anterior se debe a que ambas dependen de la topología de la red, por lo que serán diferentes en cada caso de estudio.

En el caso de la función objetivo, ésta surge a su vez de resolver un problema de optimización, el cual involucra la estadística del flujo en cada caso, lo que implica, como se dijo anteriormente, que no es posible obtener una expresión analítica de la misma, y además su evaluación puede resultar computacionalmente muy costosa.

Por otro lado, la función de las restricciones se calcula recursivamente sobre los LSP que son de importancia para el enlace a estudiar y depende del tráfico instantáneo de entrada a la red de cada uno de ellos. Cuando decimos que un LSP es de importancia para un determinado enlace, nos estamos refiriendo simplemente a que el tráfico que llega a dicho enlace se ve afectado (i.e. comparte algún buffer) de alguna manera por el tráfico del LSP. Lo anterior provoca también los dos efectos antes mencionados, es decir, que el cálculo puede ser costoso, y a su vez, va a ser diferente en cada caso de estudio.

Llamando f a la función objetivo y g a la función de las restricciones, podemos plantear el problema de optimización (P) en forma genérica, de la siguiente manera:

$$(P) \begin{cases} \text{mín } f(x) \\ \text{sujeto a } g(x) \leq 0 \end{cases} \tag{6.1}$$

donde $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}$ y $x \in \mathbb{R}^n$

En general, la dimensión del problema (n) va a depender de la cantidad de flujos que ingresan a la red en estudio (K), ya que la función f no es más que la suma las funciones de velocidad de cada uno éstos flujos. De todas maneras, en cada problema particular a resolver, n podrá ser diferente, siendo siempre menor o igual a K. Ésto es porque no todos los LSP son de importancia a la hora de resolver el problema para un enlace determinado.

En el caso que un LSP no tenga ninguna influencia sobre el tráfico que entra al enlace en cuestión, la función g no dependerá del mismo, así como tampoco dependerá el mínimo de la función objetivo. Lo anterior se debe a la forma que tiene la función f, que no es más que la suma de varias funciones escalares de variables independientes entre sí. De no ser por la existencia de las restricciones, para hallar el mínimo de toda la suma bastaría con hallar el mínimo de cada una de éstas funciones de una variable por separado. En general esto no es válido para nuestro estudio debido a la existencia

de las restricciones, pero en el caso en que un LSP no sea de relevancia para un enlace determinado la función de las restricciones será independiente del mismo, lo que nos permitirá minimizar su función de velocidad por separado y sin restricción alguna.

Todo lo anterior nos permite concluir que es innecesario resolver el problema completo de optimización en dimensión K, ya que todos aquellas coordenadas cuyo LSP correspondiente no sea de importancia, no afectarán el valor del mínimo. Por lo tanto, a los efectos de optimizar el cálculo, se optó por resolver un sub-problema del completo, conteniendo solo aquellas variables que sí sean de interés. Esto a su vez plantea un nuevo problema a resolver, que es determinar para cada enlace a estudiar aquellos LSP que son de importancia.

6.1. Técnicas de optimización

Es común en los problemas de minimización con *restricciones lineales*, que todos los puntos de la sucesión que converge al óptimo sean factibles, pero en el caso que las restricciones sean no lineales, no es para nada trivial (e incluso podría ser imposible) generar la secuencia de iterados que satisfagan exactamente todas las restricciones del problema. Por lo tanto, es prácticamente imprescindible que algunos de los puntos de la sucesión puedan ser no factibles, con la condición de que el óptimo si lo sea. Lo anterior presenta un compromiso, es decir, será necesario decidir si el punto x_{k+1} es "mejor"que el x_k en el caso que no sean factibles[1]. Para esto se define comúnmente una *función de mérito*, que en cierta forma balancea los dos objetivos (generalmente opuestos) de reducir la función objetivo y satisfacer las restricciones. Dicha función permitirá decidir cual de los dos puntos utilizar.

En base a lo que se presentó en la sección anterior, se optó por estudiar la posibilidad de resolver un problema sin restricciones para hallar el mínimo buscado, mediante alguna transformación adecuada del problema. Si llamamos x^* al mínimo de nuestro problema original (P) (ecuación 6.1), un enfoque común es el de construir una función cuyo mínimo *sin restricciones* sea o bien x^* o al menos esté relacionado con x^* de una manera conocida. En general, el problema original puede ser resuelto formulando una secuencia de subproblemas sin restricciones.

6.1.1. Métodos de penalidad

En nuestro caso particular, se utilizó lo que se conoce como "Métodos de penalidad", que en pocas palabras lo que hace es construir una función de mérito adecuada, la cual luego pasa a minimizar sin restricciones.

Como se decía, en general el mínimo de f sin restricciones se da en un punto no factible, o f podría no ser acotada inferiormente, por lo tanto las soluciones a cualquier secuencia de subproblemas sin restricciones va a converger a x^* solamente si dicha secuencia asegura la factibilidad en el límite. Una posibilidad para lo anterior, que fue la que se implementó, es la de agregar un término a la función objetivo que asigne una "penalidad"positiva que aumente ante las violaciones crecientes de las restricciones. En particular lo que se utilizó en nuestro caso fue de utilizar la siguiente *función de penalidad*

$$\phi_\rho(x) = f(x) + \rho \left(g_m(x)\right)^2 \tag{6.2}$$

[1]Observar que en el caso que ambos sean factibles, lo anterior es trivial, simplemente será necesario que $f\left(x_{k+1}\right) < f\left(x_k\right)$ para que x_{k+1} sea "mejor"que x_k

donde $g_m = \max\{g(x), 0\}$. Al término $\rho\left(g_m(x)\right)^2$ se le llama *término de penalidad*. Por lo tanto, en el caso en que se tenga un punto factible x_f, $\phi_\rho\left(x_f\right) = f(x_f)$ pero en el caso que se tenga un punto x_{nf} no factible, $\phi_\rho(x_{nf}) > f(x_{nf})$.

En las hipótesis en las que se trabaja en nuestro caso se puede probar sencillamente que, en caso que exista un mínimo sin restricciones para cada valor de ρ del problema de la ecuación 6.2 ($x^*(\rho)$), éste verifica:

$$\lim_{\rho\to\infty} x^*(\rho) = x^*$$

es decir, si se resuelve el problema sin restricciones sucesivamente, con valores de ρ crecientes, se obtiene un sucesión de puntos $x^*(\rho)$ que converge el mínimo buscado.

Lo anterior se puede interpretar de manera intuitiva de la siguiente manera, como ya se dijo, el valor de la función ϕ_ρ dentro de la región factible es igual al de la función f. En el caso en que el mínimo $x^*(\rho)$ se de afuera de dicha región, (i.e. $x^*(\rho)$ viole las restricciones), el valor de la función ϕ_ρ en dicho punto aumentará a medida que ρ crezca, hasta que éste deje de ser mínimo. Así ocurre en general con todos los puntos que no cumplen las restricciones, hasta que llega un momento en que el mínimo de ϕ_ρ se da un punto factible.

Con lo anterior se llegó a un problema de optimización sin restricciones, el cual deberá ser resuelto mediante algún método adecuado. El método utilizado se basa en el conocido *método de máxima pendiente*. En particular lo que se hará es implementar el método de máxima pendiente realizando búsquedas inexactas mediante la regla de Wolfe.

6.2. Método de máxima pendiente

El método de máxima pendiente esta dentro de la categoría de lo que se conoce como *métodos de descenso*, cuya principal característica es que el valor de la función objetivo $F(x)$ en el paso k es siempre mayor que el valor en el paso $k+1$ ($\forall k > 0$). De forma general, este tipo de métodos utilizan el siguiente algoritmo:

Algoritmo M (*Modelo de algoritmo para minimización sin restricciones en n dimensiones*) Sea x_k la estimación actual de x^*.

M1 [Test de convergencia] Si se satisfacen las condiciones de convergencia, el algoritmo termina con x_k como solución.

M2 [Cálculo de la dirección de búsqueda] Se halla un vector no nulo $p_k \in \mathbb{R}^n$, que será la dirección de búsqueda.

M3 [Cálculo del largo del paso] Se calcula un escalar positivo α_k (largo del paso) para el cual se cumple que $F\left(x_k + \alpha_k p_k\right) < F(x_k)$.

M4 [Actualizar la estimación del mínimo] Se establece $x_{k+1} \leftarrow x_k + \alpha_k p_k$, $k \leftarrow k+1$ y se vuelve al paso **M1**

Como notación se utilizará $F_k = F(x_k)$ y $g_k = g(x_k)$, donde $g(x)$ es el gradiente de la función objetivo en el punto x.

Vale la pena mencionar, que en el paso **M3** del algoritmo modelo, aparece la solución de un problema de una sola variable (hallar el escalar α_k). Para satisfacer la condición principal, es decir que $F_{k+1} < F_k\ \forall k > 0$, p_k y α_k deberán tener ciertas propiedades. A continuación se pasará a detallar lo hecho en cada uno se los pasos el algoritmo.

6.2.1. Convergencia del algoritmo

Se va a presentar de manera superficial las condiciones básicas que deben satisfacerse para la convergencia del algoritmo en cuestión. No es la intención de la presente sección hacer un análisis en profundidad de esto, en caso que se deseen más detalles, referirse a [17].

La condición de $F_{k+1} < F_k$ no es suficiente para garantizar la convergencia a un mínimo de F (e.g. considerar la función x^2 evaluada en la sucesión de puntos $(-1)^k \left(\frac{1}{2} + 2^{-k}\right)$).

Si se cumple dicha condición y la función F está efectivamente acotada, entonces la sucesión converge, pero lo importante es asegurarse que lo haga al punto x^*. Existen dos situaciones que deberán ser evitadas para asegurar la convergencia.

1. Sea cual sea la dirección de descenso p_k, la sucesión de largos de los pasos $\{\alpha_k\}$ podría ser monótonamente decreciente, ocurriendo que el algoritmo converja a algún punto sin estar cerca del mínimo buscado.

2. La sucesión de direcciones $\{p_k\}$ podría ser tal que, en una aproximación de primer orden F sea constante a lo largo de p_k. Esto se daría en el caso en que p_k se elija "casi" ortogonal a g_k.

Tanto las direcciones de descenso como los largos de los pasos deberán elegirse de manera de evitar las dos situaciones anteriores. A continuación veremos en cada caso la solución propuesta, así como la justificación de dicha elección.

6.2.2. Cálculo de la dirección de descenso

Una de las direcciones de descenso más utilizadas, principalmente por su simplicidad de cálculo así como su efectividad y que justamente le da el nombre al método, es la de máximo descenso, esto es $p_k = -g_k$. En cada paso se usa como dirección de búsqueda la opuesta al gradiente de la función objetivo en dicho punto, que es la dirección de máximo crecimiento de F.

Haciendo una buena elección del largo del paso de búsqueda, ésta elección de la dirección de descenso asegura la convergencia, ya que trivialmente evita la ortogonalidad de g_k y p_k.

6.2.3. Cálculo del largo del paso

La primera observación que se debe hacer al momento de elegir el método para el cálculo del largo de la búsqueda, es que una vez fijada la dirección de descenso, se tiene un problema de una sola dimensión. En general, lo que se busca es hallar un valor de α_k que minimize la función escalar $h(\alpha_k)$:

$$h(\alpha_k) = F\left(x_k + \alpha_k p_k\right)$$

En definitiva lo que estaría faltando al algoritmo de minimización en n dimensiones es resolver un problema de dimensión 1. Lo anterior es una gran ventaja, pero el problema de minimizar la función $h(\alpha_k)$ no necesariamente es sencillo.

En particular, en nuestro caso, que no se conoce una expresión analítica para la función F, no será directo el cálculo del α_k, sino que sería necesario implementar algún algoritmo iterativo. Si bien en un principio puede parecer óptimo hallar el mínimo de la función h, ésto podría complicar de gran manera el algoritmo, sin obtener mejoras

en los resultados finales. Es decir, no sería necesario hallar el mínimo exacto de dicha función en cada iteración del algoritmo, sino que sería suficiente elegir un valor de α_k que logre un menor valor de la función objetivo en el nuevo punto, sin que éste valor sea el mínimo. En definitiva, lo que se decidió hacer fue implementar un algoritmo de los que se conocen como de *búsqueda lineal inexacta*, que verifica lo anterior.

La búsquedas lineales inexactas no afectan la eficacia del algoritmo iterativo, ya que la precisión la va aportar el algoritmo **M** en sí, no cada una de las búsquedas lineales. Incluso se puede ver sencillamente que estos algoritmos inexactos tienen mejor efectividad que aquellos de búsqueda exacta, no solamente por el hecho de que no invierten tantos cálculos en la búsqueda del mínimo, sino por problemas inherentes a los algoritmos exactos en sí.

6.2.4. Búsqueda lineal inexacta

La idea detrás de todos los métodos de búsqueda lineal inexacta es la de conseguir mediante pocas evaluaciones de la función un punto "suficientemente bueno". De alguna manera se deberá establecer un criterio que nos permita decidir cuando un punto es bueno o no. En general, en todas las búsquedas lineales se plantean un test para cada valor de α_k analizado, y construyen α_{k+1} según el resultado. En pocas palabras dicho test consistirá en decidir para un α_k dado si éste es adecuado (condición a), en cuyo caso terminaría la búsqueda, o es demasiado grande (condición b) o es demasiado chico (condición c). Como condición única, las tres zonas, a,b y c deben constituir una partición de $\mathbb{R}$, es decir cada valor de α_k deberá estar clasificado en alguna de las tres condiciones anteriores.

Los algoritmos de búsqueda lineal inexacta utilizan dos variables auxiliares t_g y t_d, las cuales permiten definir el rango de valores entre los cuales se buscará el nuevo valor α_k. De forma genérica, éstos algoritmos tienen la siguiente estructura:

P0. $\begin{cases} \alpha_k \text{ inicial} \\ t_g = 0, t_d = \infty \text{ (o } t_d \text{ indefinido)} \end{cases}$

P1. Test: $\begin{cases} \text{Condición a: FIN} \\ \text{Condición b: } t_d \leftarrow \alpha_k \\ \text{Condición c: } t_g \leftarrow \alpha_k \end{cases}$

P2.
- Si todavía no esta definido t_d o $t_d = \infty$: **Extrapolación** Se elige $\alpha_k > t_g$ (Por ejemplo $\alpha_k = 5t_g$)
- Si t_d sí esta definido: **Interpolación** Se elige $\alpha_k \in [t_g, t_d]$, $\alpha_k = \frac{t_g + t_d}{2}$

Una de las cosas a definir en cada comienzo del algoritmo, es el valor de α_k inicial. Por lo general, y en particular en nuestro caso se utilizó como α_k el valor de α_k hallado en la iteración global anterior (del algoritmo **M**).

Como muestra el esquema general del algoritmo, en el paso 2 (P2) se pueden tener dos situaciones diferentes, las cuales son llamadas extrapolación e interpolación. La primera de éstas se da solamente al comienzo del algoritmo, en el caso en que t_d no está aún definido (o $t_d = \infty$). Se hace crecer el α_k hasta el punto que el test del paso 1 da como resultado b, o sea, ya se llegó a un valor de α_k que supera la zona de valores adecuados. La interpolación es lo que ocurre con más frecuencia, que es el caso en que se va reduciendo el intervalo $[t_g, t_d]$ con el objetivo de llegar a la zona adecuada. Existen variaciones para la elección del nuevo valor de α_k en las interpolaciones, que

utilizan algún tipo de información adicional de la función $h(\alpha_k)$ para hacer una mejor elección del mismo. Por ejemplo se pueden citar métodos que mejoran la eficiencia de las interpolaciones aproximando la función por polinomios, y luego hallando el mínimo de éstos. En nuestro caso llegamos a la conclusión que implementar alguno de estos tipos de interpolación no mejoraba significativamente la eficiencia del algoritmo, por lo que no se llevó a cabo. Se agregaba complejidad al método cuyo punto problemático, computacionalmente hablando no radicaba en esa parte.

Por último, hace falta decidir qué vamos a considerar como un valor de α_k "adecuado". Para esto lo que se utilizará es la *Regla de Wolfe*, cuyas tres zonas a, b y c se definen a continuación:

Condición a. $h(\alpha_k) < h(0) + m\,\alpha_k h'(0), \quad h'(t) \geq m'\,h'(0)$

Condición b. $h(\alpha_k) > h(0) + m\,\alpha_k\,h'(0)$

Condición c. $h(\alpha_k) \leq h(0) + m\,\alpha_k\,h'(0), \quad h'(t) < m'h'(0)$

donde $0 < m < m' < 1$.

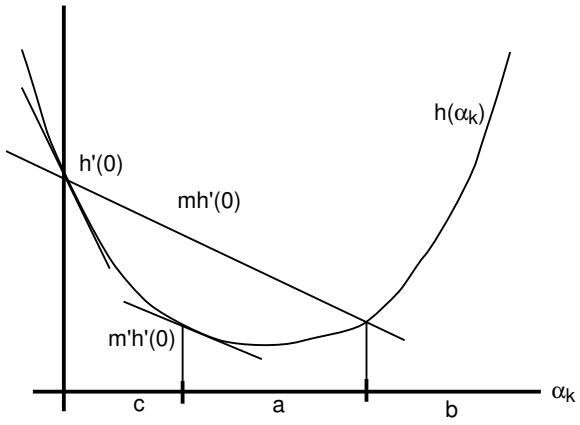

Figura 6.1: Zonas definidas por la Regla de Wolfe

En la figura 6.1 se pueden observar claramente las tres zonas definidas por la regla de Wolfe. En pocas palabras, esta regla decide que el valor de α_k es adecuado, si cumple dos cosas. Primero, que la pendiente este más cercana a cero de lo que estaba en el punto x_k ($\alpha_k = 0$), según un factor (m') a definir por el usuario, en nuestro caso se uso $m' = 0{,}5$. Segundo, debe verificar que el valor de la función sea menor que un cierto valor prefijado, que viene dado por otro parámetro m, el cual al elegirse menor que m' asegura que el extremo derecho de la zona de búsqueda sea mayor que el punto donde se da el mínimo. En nuestro caso se utilizó $m = 0{,}1$.

6.3. Algoritmo de minimización

En conclusión, para la resolución del problema de minimización de la sección 3.4 se implementó un algoritmo de acuerdo a lo antes presentado, es decir, se transformó el problema con restricciones en un problema sin restricciones mediante un método de penalidad. Luego para le resolución de éste último se llevo a cabo el método de máxima pendiente, para el cual se utilizaron búsquedas lineales inexactas según la regla de Wolfe para hallar los largos de los pasos de búsqueda.

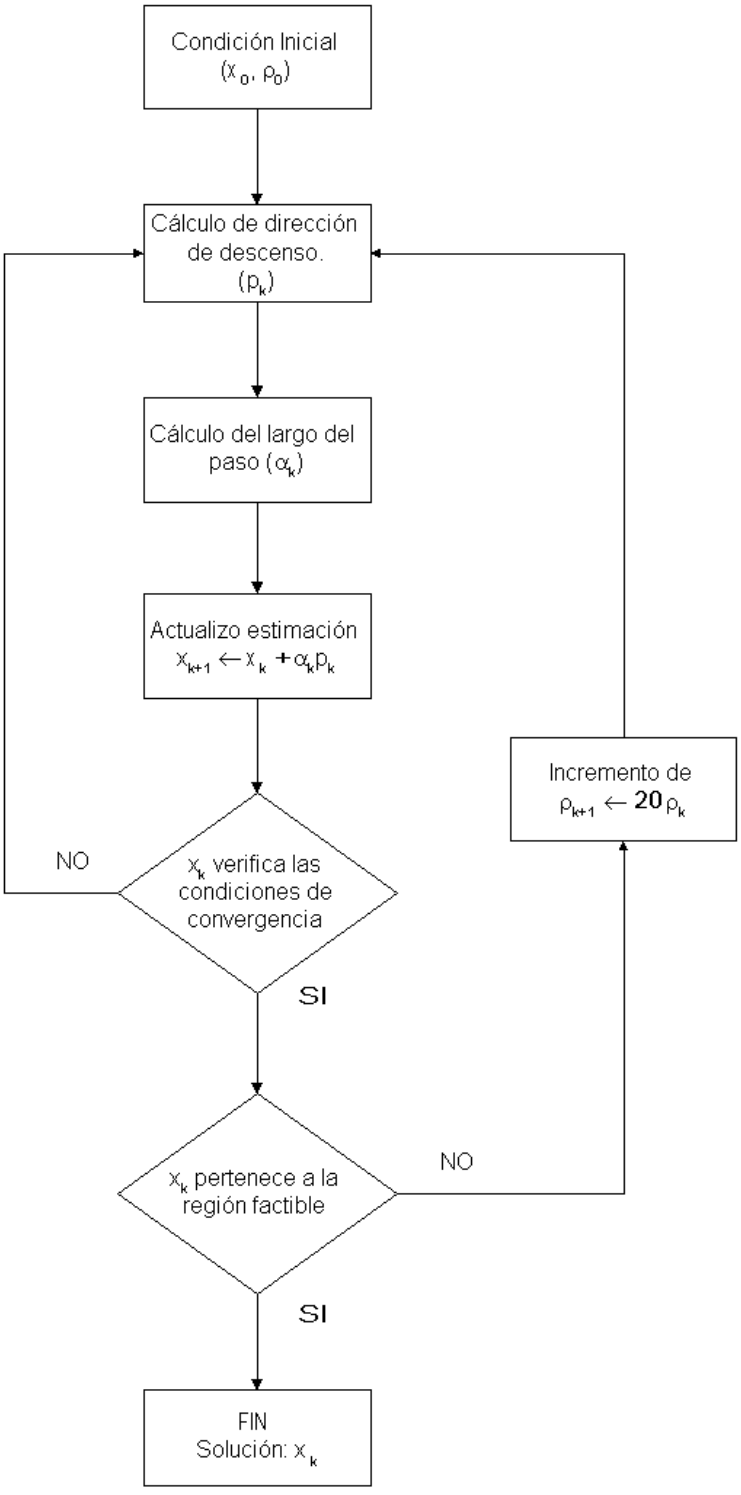

Figura 6.2: Diagrama de Flujo del algoritmo iterativo

En la figura 6.2 se puede apreciar un esquema del algoritmo iterativo implementado. En cada iteración del algoritmo se resuelve el problema auxiliar P_ρ

$$(P_\rho) \begin{cases} \text{mín}\, \phi_\rho = f(x) + \rho\,(g_m(x))^2 \\ x \in \mathbb{R}^n \end{cases}$$

para un valor determinado de ρ. Como se explicó antes, el mínimo del problema P_ρ es siempre menor o igual al mínimo del problema original y la igualdad se da en el caso que el mismo sea un punto factible, es decir en el caso que la penalidad sea nula. Por lo tanto, luego de cada resolución del problema auxiliar se evalúa si el mínimo $x(\rho)$ es un punto factible. En el caso que lo sea concluye toda la iteración y se tiene el óptimo, siendo $x^* = x(\rho)$. En caso contrario, se aumenta el valor de ρ (en particular en nuestro caso se hace $\rho_{k+1} = 20\rho_k$) y se vuelve a resolver el problema P_ρ. La consecuencia de hacer crecer el valor de ρ es obtener un mínimo del problema P_ρ cada vez mayor y por lo tanto cada vez más proximo al mínimo del problema original. Como se explicó anteriormente, esto se llevó a cabo hasta que el mínimo $x(\rho)$ fuera un punto factible, es decir, que sea un mínimo del problema original P.

En la sección 8.7 se discuten los detalles prácticos de la implementación del anterior algoritmo.

Parte III

Arquitectura de Software

Capítulo 7

Introducción a la herramienta de software

Uno de los objetivos con los que se comenzó este proyecto, era el de desarrollar una herramienta de software, que permitiera realizar estimaciones de performance sobre una determinada red, en base a la teoría anteriormente desarrollada. Para esto, dentro del amplio espectro de lenguajes de programación existentes, se optó por utilizar Java, ya que todos los integrantes de este proyecto están familiarizados con el mismo.

A su vez, Java presenta una serie de ventajas que justifican su elección, entre las cuales se pueden mencionar algunas como su gran versatilidad, debido a que permite programar orientado a objetos y el hecho de ser un lenguaje mundialmente difundido lo que permite un fácil acceso a documentación, bibliografía y librerías existentes.

A la hora de encarar el desarrollo de esta herramienta, nos planteamos una serie de condiciones que la misma debía intentar cumplir. Debido a la extrema complejidad de la teoría en que se basa el proyecto, es deseable que la herramienta permita que un usuario no familiarizado con dicha teoría pueda obtener resultados que le sean de utilidad. Para esto el funcionamiento debe ser transparente al usuario, quien solo debe tener conocimientos básicos de su red y los flujos que por la misma circulan.

Otra condición no imprescindible pero sin duda deseable es que la herramienta provea resultados confiables con tiempos de cálculo mínimos, con el objetivo de llegar a implementar técnicas de TE en linea. Para esto, si bien es necesaria la realización de desarrollos posteriores al presente proyecto, estos carecerían de sentido si nuestra herramienta no cumple con el anterior objetivo. En este mismo sentido, es condición necesaria que las bibliotecas finales sean fácilmente reutilizables constituyendo así la primera parte del proyecto más general antes mencionado.

Teniendo en cuenta los anteriores conceptos se basó nuestra arquitectura en la división en packages independientes entre sí, orientados a agrupar funcionalidades y conceptos, permitiendo así reemplazar partes de la misma de acuerdo con las necesidades futuras. Es así que se diseño un package destinado a la representación de la red, llamado *Topología*, el cual interactua con otro llamado *InterfazSistema*, que es el encargado de controlar el flujo de ejecución del programa. Estos se ven complementados por el package *Excepciones* que sirve de contingencia ante las posibles anomalías propias de la interacción con agentes externos. Finalmente se implementó un package *InterfazUsuario*, que provee una interfaz gráfica mediante la cual el usuario puede interactuar amigablemente con el programa.

En la figura 7.1 se muestra como se pretende que sea la forma de trabajo de la herramienta, indicando los principales mensajes que se intercambian entre las distintas entidades.

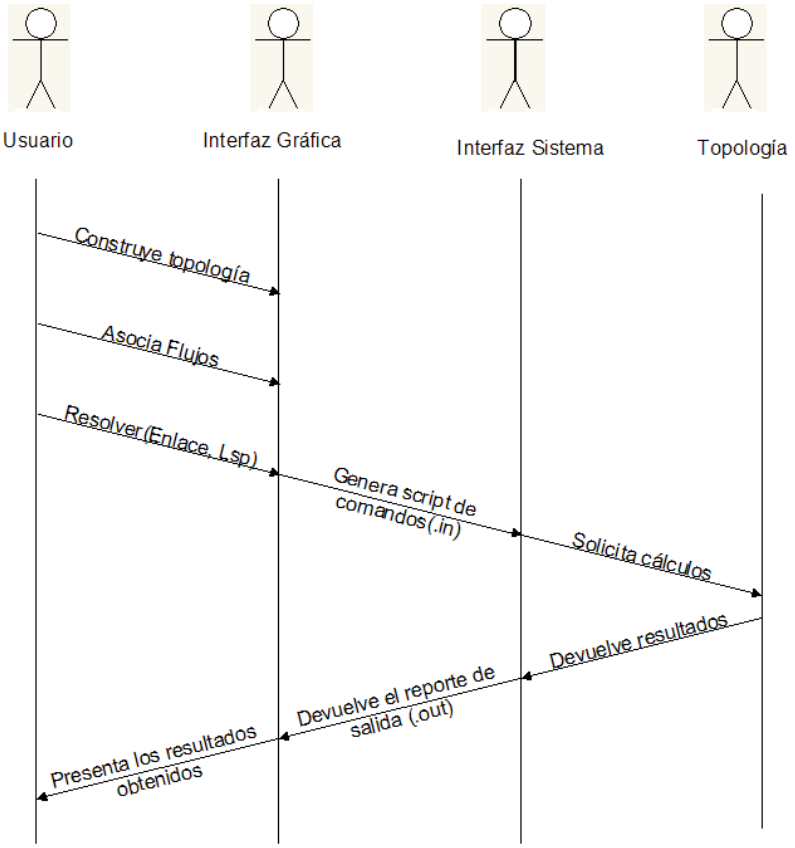

Figura 7.1: Caso de uso típico

En la figura anterior se aprecia el rol que juega cada uno de los packages en el cálculo de estimaciones de performance. De esta manera, el usuario comienza la ejecución del programa construyendo una topología de red y solicitando cálculos mediante el el package *InterfazGrafica*. Con esta información, este último genera un script de comandos, utilizando un lenguaje entendible por el package *IntefazSistema* quien solicita al package *Topología* la realización de los cálculos pertinentes. Una vez completados los cálculos se recorre un proceso inverso, donde *InterfazSistema* levanta los resultados obtenidos, generando un reporte de los mismos. Por último, la interfaz gráfica se encarga de presentar los resultados obtenidos al usuario. Todo lo anterior cuenta con el soporte del package *Excepciones* quien provee de un marco de control sobre todos las posibles ejecuciones que puedan conducir a situaciones no previstas.

Para llevar a cabo los procesos anteriores, cada uno de los packages cuenta con una serie de clases, de acuerdo a lo que se muestra en la figura 7.2

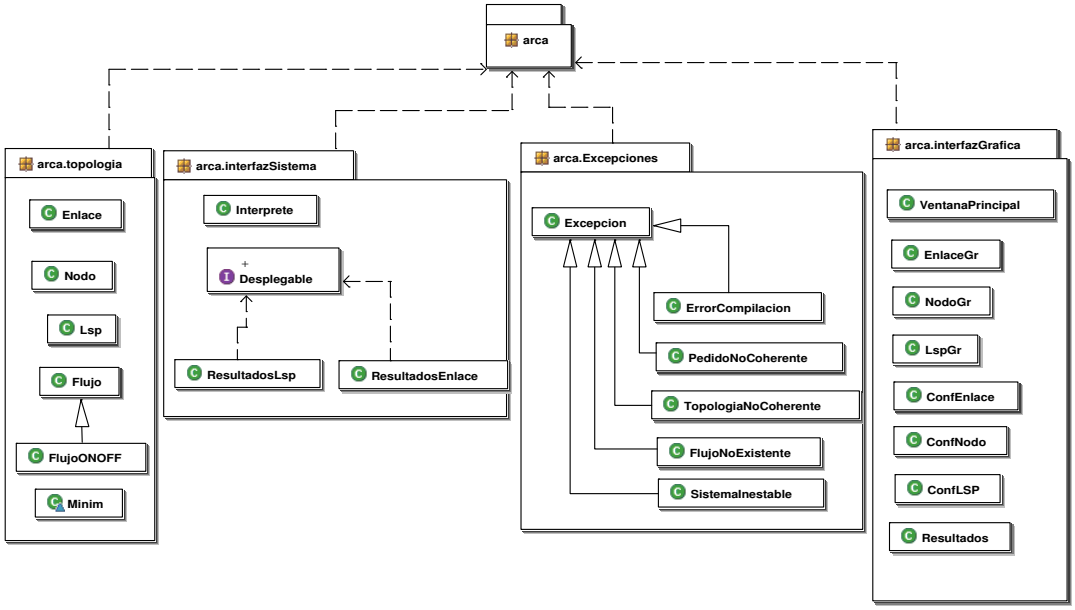

Figura 7.2: Diagrama de packages de la herramienta de software

En los siguientes capítulos, se procederá a describir en detalle la composición de cada package y la forma en que interactuan sus correspondientes clases.

Capítulo 8

Representación de la red

Una parte neurálgica del presente proyecto consistió en abstraer la arquitectura de la red en estudio y representarla de una manera fidedigna mediante un conjunto de clases de un lenguaje orientado a objetos como es JAVA.

En el proceso creativo de dicha arquitectura, se consideraron una serie de factores que influyeron de distinta manera en el resultado final.

Uno de los principales objetivos buscados, era poder representar de la forma más general posible, cualquier tipo de red, en particular, aquellas sobre las cuales se pueden definir caminos virtuales, dada su versatilidad para la aplicación de técnicas de TE. Debido a que nuestro proyecto toma como referencia las redes de caminos virtuales, en particular MPLS, el diseño busca contemplar todas las funcionalidades que estas redes implementan.

Por otra parte, el presente proyecto, no pretende abarcar todas las aristas posibles del problema de estimar la performance de una red, restando aún, encarar diversos aspectos de la misma. Esto implica la posibilidad continuar con nuestro estudio en una etapa posterior. Lo anterior conduce a que nuestra arquitectura debe contar con la imprescindible condición de la reutilizabilidad, que permita una fácil extensión de la arquitectura implementada.

La correcta división del proyecto en paquetes (packages), apunta en la misma dirección, ya que si otro grupo de desarrollo decide profundizar en otras áreas específicas, puede reutilizar únicamente los paquetes de interés.

Basados en los lineamientos generales trazados anteriormente, buscamos identificar dentro de la red, aquellos componentes que por sus características físicas o funcionales representen un concepto dentro de la misma, para asociarles una clase que describa sus propiedades e implemente sus funcionalidades. Es así que mediante la interacción de las mencionadas clases se llegó a arquitectura de red que se describirá en breve y que proporciona el soporte físico a los procesos que allí se desarrollan.

En la sección 8.1 se describe y fundamenta la forma en la que se organizó la información considerada para la representación de la red. A continuación en las subsiguientes secciones se describe más en detalle cada una de las clases que componen el package que da nombre al capítulo.

8.1. El package Topología

Luego de manejar varias opciones, llegamos a la conclusión que el modelo de red que mejor servía a nuestros intereses contenía tres niveles de abstracción:

- Un nivel que describa la topología de la red. Para ello se implementaron dos clases, *Nodo* y *Enlace*.
- Un nivel donde se establecen caminos sobre dicha topología, representado por la clase *Lsp*.
- Un nivel que describa las propiedades del tráfico que circula por la red, lo cual se hace en la clase *Flujo*, de la cual heredarán otras clases, que describirán diferentes modelos de tráfico.

Todas estas clases conforman un mismo package, que da nombre a esta sección, al cual se agrega la clase *Minim* con el objetivo de agrupar herramientas matemáticas, que realizan cálculos sobre la topología. El diagrama UML de la figura 8.1 permite una mejor visualización del relacionamiento de las clases que componen este package.

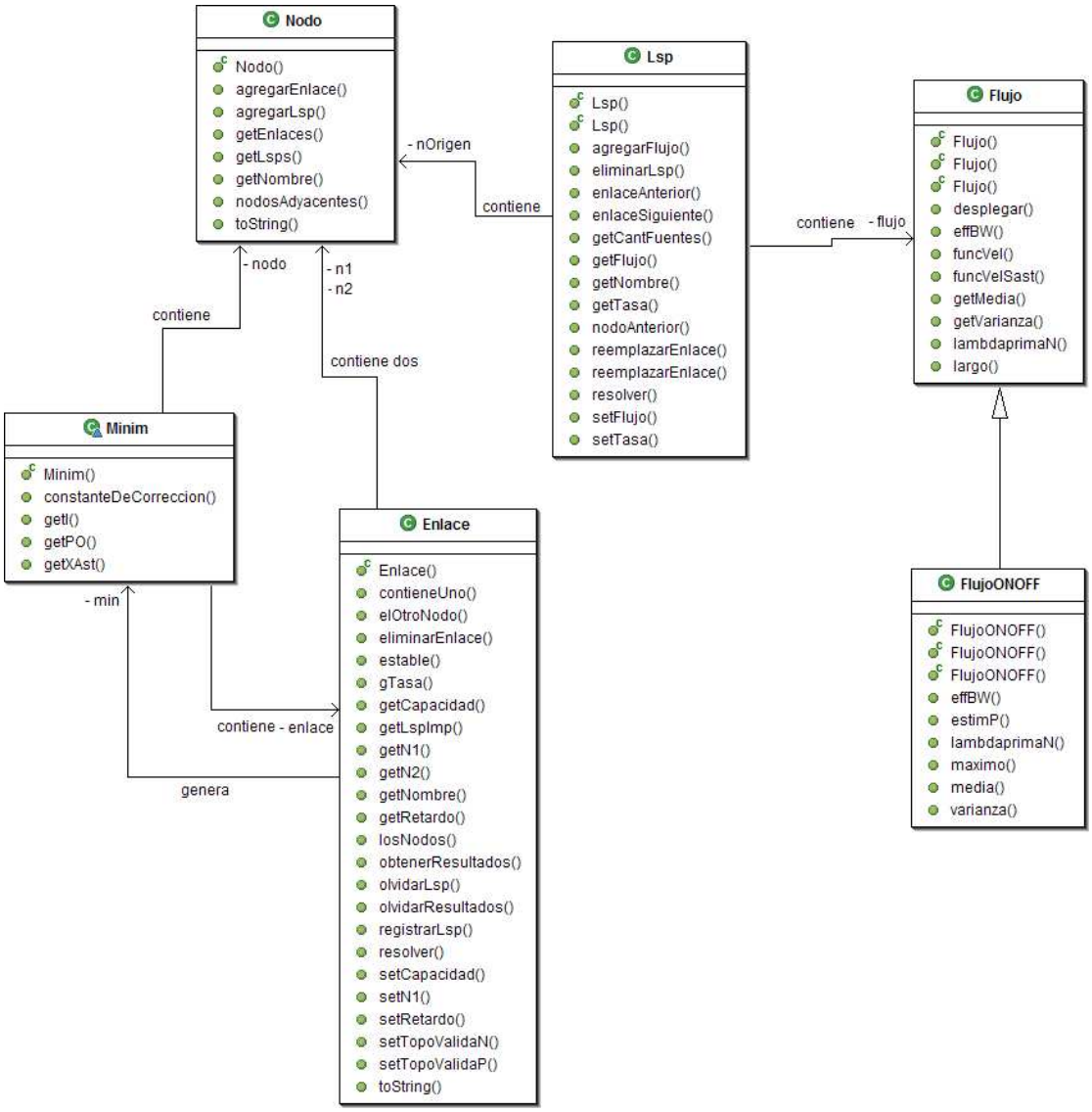

Figura 8.1: Diagrama UML del package Topologia

A continuación se describe más en detalle las clases anteriormente mencionadas.

8.2. La clase Nodo

El *Nodo*, representa dentro de nuestra arquitectura de red un elemento de conexión, en el convergen o se originan un número no predeterminado de enlaces, los cuales luego formaran los caminos que atravesarán la misma. En este sentido los nodos son fundamentales para establecer el sentido en que se recorren estos caminos.

Por otra parte los nodos permiten realizar de forma relativamente sencilla un chequeo lógico de la continuidad de los caminos establecidos, permitiendo una temprana detección de posibles errores en la diagramación de la red a estudiar. Esto es posible ya que cada instancia de Nodo lleva un registro de los enlaces y lsp que pasan por el. Como se deduce de lo anterior, el *Nodo*, tiene una función auxiliar en lo que refiere a la transmisión de datos dentro de la red.

8.2.1. Campos principales

Nombre Debido a la naturaleza topológica de los nodos, se hace imperativa contar con una manera de referenciarlos. Todas aquellas clases que representan elementos físicos de la red deberán crearse a partir de los nodos. Por lo tanto este campo identifica de manera única cada uno de los nodos. Por un tema de robustez, a este campo se le dio alcance privado.

Enlaces Como se dijo previamente, la clase Nodo representa el elemento de interconexión entre los enlaces, por lo que será útil que cada nodo tenga un registro de aquellos que se originan o llegan a él. Para ellos se guardan referencias a las instancias de los objetos Enlace en un contenedor de visibilidad privada, el cual se maneja con un método adecuado.

Lsps Al igual que para el caso de los enlaces, los nodos registran cada uno de los LSPs que por ellos pasan.

8.2.2. Constructor

La clase *Nodo* cuenta con un único constructor, el cual recibe como parámetro el nombre a asignarle a la instancia creada. En la creación de la instancia se inicializan los contenedores antes mencionados

8.2.3. Métodos de acceso y definición de parámetros

Para cada uno de los parámetros de la clase *Nodo* existen métodos correspondientes que permiten asignar y recuperar los valores de éstos (`get` y `set`).

8.2.4. Métodos que agregan funcionalidad a la clase

agregarEnlace

Este método permite registrar el enlace recibido como parámetro en el contenedor "enlaces"descrito anteriormente.

Lsp

Este método permite registrar el LSP recibido como parámetro en el contenedor "lsps"descrito anteriormente.

nodosAdyacentes

Permite recuperar los nodos comunicados con la instancia en la cual se invoca. Puede ser útil para reconstruir localmente la topología de la red.

8.3. La clase Enlace

El *Enlace* es el elemento fundamental de la red en lo que a transmisión de datos se refiere. Este concentra la mayoría de los parámetros relativos a esta tarea como ser la Capacidad de procesamiento de datos, el buffer, etc.

Cada instancia de *Enlace* representa una conexión bidireccional entre dos instancias de *Nodo* que recibe en su construcción. Estas se almacenan como parámetros ordenados de manera de determinar el sentido en que se utiliza dicho *Enlace*. Es así que se definió que un *Enlace* se recorre en sentido "positivo" si dentro de un camino, el primer *Nodo* involucrado es aquel que se recibió en primer lugar en la construcción y el sentido será "negativo" en caso contrario. Esta definiciones son arbitrarias, y sirven a los efectos de distinguir y separar los caminos que pasan por un determinado *Enlace* en uno y otro sentido. Esto último es necesario a la hora de determinar qué caminos comparten recursos en un mismo *Enlace* y por ende son relevantes para calcular la probabilidad de overflow.

La clase *Enlace* cuenta con métodos que permiten determinar parámetros relevantes para el cálculo de la probabilidad de overflow, los cuales se describen en las siguientes subsecciones.

8.3.1. Campos principales

Nombre Debido a las mismas razones que se detallaron en la sección 8.2.1 es necesario referenciar a los enlaces, por lo que este campo es su principal identificador.

Nodos (n1,n2) Cada instancia de la clase *Enlace* contiene referencias a los dos nodos que el mismo conecta, las cuales se almacenan en este campo.

capacidad Cada instancia de la clase *Enlace* debe tener conocimiento de su capacidad de transmisión, la que se almacena en este campo. Un factor a tener en cuenta a la hora de definirla, son las unidades, que deben estar de acuerdo con las unidades utilizadas en el flujo. Para un caso de uso práctico, referirse al capítulo 9.

retardo En este campo se almacena el retardo de transmisión de la instancia de enlace en cuestión. Si bien en esta versión de la herramienta de software este campo permanece inutilizado, se consideró que dada importancia de este parámetro en los enlaces reales era de conveniencia incluirlo de modo de facilitar la reutilización de la clase.

resP, resN Luego de realizados los cálculos solicitados por el usuario, cada instancia de la clase *Enlace* crea dos instancias de la clase *ResultadosEnlace* para almacenar en éstas los resultados obtenidos. Cada una de las instancias de *ResultadosEnlace* almacena los resultados correspondientes a cada sentido en los que circula tráfico por el enlace.

lspP, lspN Cada uno de los enlaces definidos en la topología de red tiene un registro de los LSP que circulan por él. Éstos se separan según el sentido en que circulan por el enlace y se almacenan ordenados en el contenedor correspondiente.

lspImpP, lspImpN Estos contenedores almacenan la lista de caminos virtuales que afectan al tráfico para cada sentido del enlace. Estos LSP "importantes"son aquellos que recorren el enlace y aquellos que comparten recursos con el tráfico que arriba al enlace en algún buffer anterior en la red.

topoValidaP, topoValidaN Estas variables booleanas determinan si los resultados almacenados en los campos anteriormente descritos son compatibles con la topología actual de la red. Esto es de particular utilidad para evitar cálculos innecesarios sobre redes previamente resueltas, ya que de ser solicitado un cierto cálculo, este solo se realizará, si dicho resultado no se ha calculado aun o si existe pero esta variable es falsa, ya que indica que la topología de la red sufrió algún cambio, lo que podría afectar el resultado anterior.

8.3.2. Constructor

La clase *Enlace* cuenta con único constructor el cual recibe como parámetros, el nombre que se le quiere asignar a la instancia a ser creada, los nodos que unirá y la capacidad de la misma.

8.3.3. Métodos de acceso y definición de parámetros

Para algunos de los parámetros de la clase *Enlace* existen métodos correspondientes que permiten asignar y recuperar los valores de éstos (`get` y `set`). En algunos casos particulares, como ser el nombre de la instancia, no se cuenta con un método para modificar el mismo de forma posterior a la creación de la instancia, ya que no se consideró como una buena política de programación.

8.3.4. Métodos relacionados con la función de transferencia

encLSP

Es de vital importancia el conocer aquellos flujos que incidan sobre el tráfico que arriba a un determinado enlace compartiendo recursos en otros tramos de la red. Si bien esto no es estrictamente necesario, y se podría trabajar con todos los LSP definidos en la red, por razones de eficiencia se optó por considerar solo aquellos que influyen en la probabilidad de overflow de manera de reducir las dimensiones del problema de optimización. El número de LSP de importancia para un enlace determina la dimensión del mismo.

Para la determinación de la dirección en la que se quiere hallar los LSP importantes, el método recibe como parámetro de entrada un nodo. Como primera medida *encLsp* determina aquellos LSP que pasan directamente por el enlace en cuestión, por razones obvias éstos son de importancia y los almacena en un contenedor apropiado. A continuación para cada uno de estos LSP encuentra el enlace previo en el sentido en que se están recorriendo y sobre cada uno de estos enlaces realiza el mismo procedimiento que para el enlace original. Este procedimiento se repite hasta encontrar el primer enlace de cada LSP lo que corta la recursión ya que no hay enlaces anteriores.

En la recursión anteriormente descrita, una vez que se determina que un LSP es de importancia, se verifica que el mismo no haya sido encontrado anteriormente para no

considerar repetidas veces un mismo camino. Por último este método almacena la lista hallada en el campo **lspImpP** o **lspImpN** según el sentido.

getLspImp

Este método constituye la interfaz mediante la cual se puede acceder a los LSP importantes hallados por el **encLsp** ya que este método es de alcance privado.

El cálculo se realiza solamente cuando los parámetros **lspImpP** o **lspImpN** aún no han sido inicializados o en el caso que estén desactualizados de acuerdo con la variable **topoValida** permitiendo así realizar cálculos innecesarios.

Se realiza un chequeo de coherencia en el cual se verifica que el nodo recibido como parámetro sea uno de los del enlace en cuestión, produciendo una excepción del tipo *PedidoNoCoherente*.

fMazumdar

Este método implementa la función definida en la ecuación 3.18, que permite calcular la tasa instantánea de salida de tráfico a partir del tráfico de entrada al enlace, como se discutió en el capítulo 4. Éste recibe como parámetros un arreglo con los valores de las tasas de ingreso de cada uno de los LSP que recorren un determinado enlace, así como la capacidad del mismo y un índice que indica del cual de todos los tráficos se desea hallar la tasa de salida.

gMazumdar

Debido a la gran relevancia de la función de transferencia para el cálculo de la probabilidad de overflow, ésta se implementó utilizando dos métodos, el que da nombre a esta sección y **gTasa**. Dicha función de transferencia (g) se presentó en la sección 3.4 y representa la restricción del problema de minimización.

Como ya se adelantó, el cálculo de la misma se realiza recursivamente y éste utiliza solamente aquellos LSP que son de relevancia para el enlace en cuestión. Para ello se utiliza el método **encLsp** que se describió anteriormente.

Como se mencionó anteriormente uno de los objetivos era el de poder realizar estimaciones de performance sobre cualquier tipo de red, por lo que surge la necesidad de implementar una función g robusta, e independiente de la topología de red en estudio. Específicamente, el método descrito en esta sección calcula la tasa de tráfico que arriba a un enlace dado (aquel que la invoca) por un LSP dado como parámetro a la entrada. Debido a lo anterior surge la necesidad de la implementación recursiva de este método, es decir, el hecho que calcula la velocidad del tráfico que arriba a un enlace en cualquier parte de la red, a partir de los tráficos a la entrada.

El método simplemente recorre el LSP que le fue pasado como parámetro hacia el origen del mismo, calculando en cada uno de los nodos que éste recorre, el aporte de tráfico de cada uno los LSP que a dicho nodo arriban. Para determinar el aporte de cada uno de estos flujos, se llama a la función **fMazumdar** en cada uno de los nodos antes mencionados. Cada uno de los LSP que este método utiliza para el cálculo conoce el valor de la tasa a la entrada por medio del método **gTasa**.

gTasa

Esta función constituye el complemento a la función **gMazumdar**, resultando de la acción conjunta de éstas la correcta evaluación de la función de transferencia. De manera

similar a lo que ocurre para la función **getLspImp**, **gTasa** es el medio de acceso a la función de transferencia, siendo **gMazumdar** de visibilidad privada.

La misma recibe como parámetro a la entrada un nodo, que permite determinar el sentido del enlace, y un arreglo de valores, en los que se desea calcular la función de transferencia. Para evaluar la función en el arreglo $(x_1, \ldots, x_n)$[1] **gTasa** realiza el siguiente procedimiento

- Recorre el contenedor **lspImp**, estableciendo en el i-ésimo LSP importante el valor correspondiente (x_i), de la tasa de entrada de trabajo a la red.
- En el enlace en que se solicitó el cálculo se llama a la función **gMazumdar** para cada uno de los LSP de éste.
- Se realiza sobre todos los LSP que arriban al enlace en estudio la suma del tráfico que aporta cada uno de éstos y este es el valor que retorna la función.

Al igual que en las anteriores funciones se realiza un chequeo de coherencia en el cual se verifica que el nodo recibido como parámetro sea uno de los del enlace en cuestión, produciendo una excepción del tipo *PedidoNoCoherente*.

8.3.5. Métodos relacionados con la topología

En esta sección se describen aquellos métodos implementados en la clase *Enlace* que son únicamente destinadas a obtener resultados diversos sobre la topología de la red, es decir los nodos y los enlaces de la misma.

estable Como es sabido, una de las condiciones que deben cumplir todos los enlaces de la red, previo a el comienzo de los cálculos. Por esta razón, en la clase *Enlace* se implementó un método que verifica la estabilidad del una instancia determinada, chequeando que la capacidad del enlace sea suficiente por lo menos para cursar las medias de los flujos que por él circulan.

elOtroNodo En varias oportunidades durante la ejecución del programa, es necesario conocer, a partir de un nodo que pertenece al enlace, el otro de éstos. Esto ocurre por ejemplo cuando se quieren recorrer todos los enlaces de un LSP, con el propósito de realizar algún cálculo, o simplemente revisar la continuidad de éste. Este método toma como parámetro de entrada un nodo, que debe pertenecer al enlace, y en caso que no sea así, se genera una excepción del tipo *PedidoNoCoherente*.

contieneUno Por razones similares a las descritas en el método **elOtroNodo** se implementó un método que verifica la pertenencia de un nodo al enlace en cuestión. Devuelve una variable lógica que toma el valor verdadero en caso que el nodo que es pasado como entrada, coincida con algunos de los del enlace.

registrarLsp y olvidarLsp Esta pareja de métodos controla las variables **lspP** y **lspN** permitiendo agregar un LSP a la lista correspondiente en el enlace. A su vez, como los nodos también llevan un registro de los LSP que por ellos pasan, el método **registrarLsp** también agrega al LSP a los dos nodos que conforman el enlace. La función que permite deshacer los cambios en los registros anteriores es **olvidarLsp**.

[1] El orden en que se pasan los valores en el arreglo debe coincidir con el orden de los LSP importantes.

eliminarEnlace En los casos en que se desee realizar cambios a la topología que incluyan la eliminación de enlaces, se tiene este método que lo realiza de manera adecuada, borrando todas las referencias a la instancia a eliminar en los correspondientes entidades (LSPs y nodos).

8.3.6. Métodos relacionados con el cálculo de la probabilidad de overflow

El funcionamiento general de la herramienta de software consiste en solicitar cálculos de probabilidad de overflow sobre una topología de red previamente definida. Para esto, el usuario invoca comandos que directamente piden resolver uno varios enlaces, utilizando el método **resolver** de la clase *Enlace*.

La razón por la que se manejaron de esta manera las solicitudes de resolución es porque se buscó implementar una arquitectura distribuida, donde cada instancia de *Enlace* sea capaz de resolverse y almacenar los resultados. De esta manera se evitó desarrollar una clase con alcance global que llevara a cabo todas las instrucciones sobre los enlaces y almacenara los resultados, lo que generaría un intercambio significativo e innecesario de parámetros y una gran sobrecarga sobre dicha clase.

Todos los resultados que surgen de los cálculos de la probabilidad de overflow, se almacenan un objeto de la clase *ResultadosEnlace*, de manera que cada enlace tenga accesible en todo momento los resultados ya calculados. A continuación se describe el método que lleva a cabo esta tarea, así como los métodos que permiten acceder a los resultados.

resolver

Esta sección pretende describir todos los procesos que se desencadenan una vez que el usuario solicita la resolución de un determinado enlace. El método **resolver** toma como parámetro de entrada una instancia de la clase *Nodo*, que le permita definir cual de sus buffers debe resolver. Como en los casos anteriores el presente método realiza una verificación de la existencia del nodo que recibe al enlace. En la siguiente figura se presenta el diagrama de flujo que describe la secuencia de dichos procesos.

Como se puede apreciar en la figura 8.2, la primera tarea que realiza el método **resolver** es determinar los LSP que afectan al buffer que se desea resolver mediante el método **encLsp** descrito anteriormente. En aquel caso en que no exista ningún LSP importante para el buffer en cuestión, se termina el proceso con una observación indicando que el enlace no cursa tráfico en esa dirección. Como es esperable, en este caso la probabilidad de overflow será cero.

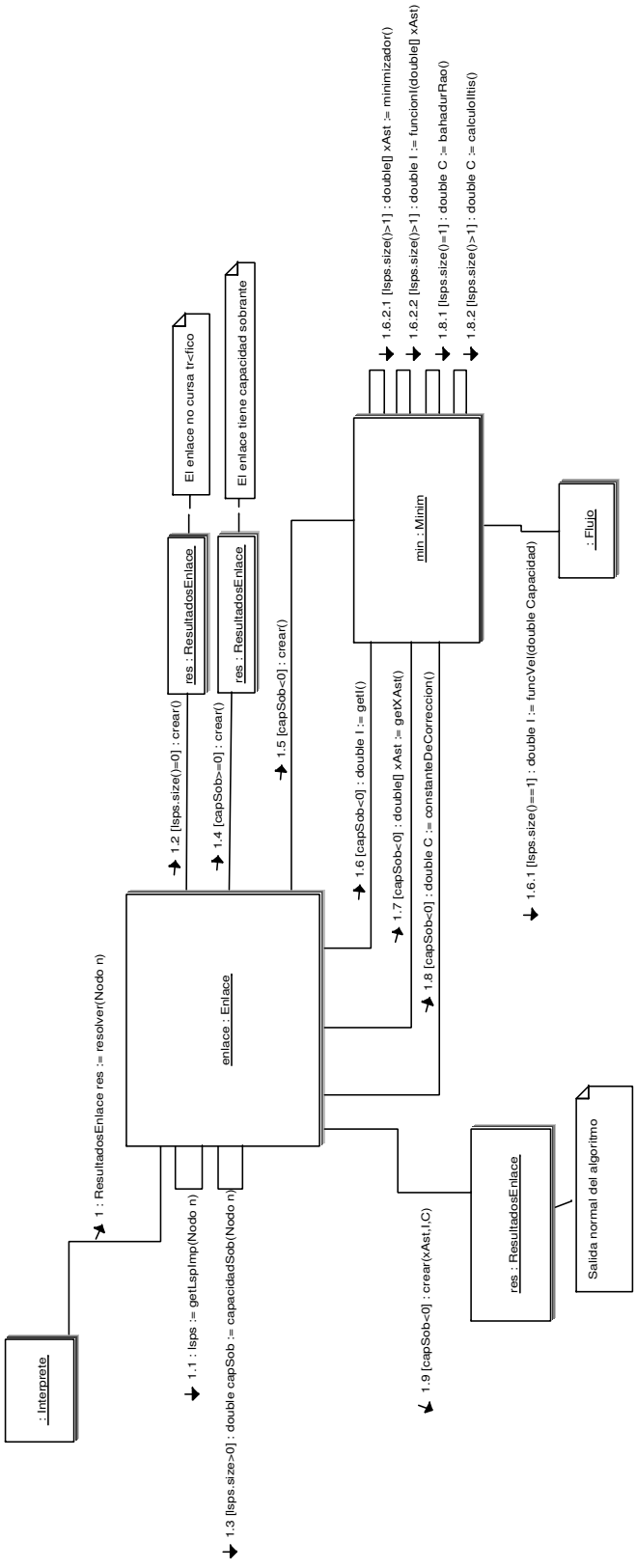

Figura 8.2: Esquema general de resolución

En caso de que existan LSPs importantes, debe verificarse si el enlace está sobredimensionado. Esto es equivalente a calcular la velocidad máxima de arribo de trabajo permitida por los enlaces que le anteceden. Si ésta es menor que su propia capacidad, entonces el enlace no perderá trabajo, y por ende está sobredimensionado. En este caso la resolución del problema es trivial ya que tampoco hay pérdidas ($I = \infty$) por lo que se finaliza la resolución con un mensaje notificando la capacidad sobrante en el enlace. Para calcular la tasa máxima posible de arribo se utiliza la función **gTasa**. La máxima tasa de arribo que puede recibir el enlace se obtiene al forzar al máximo las tasas de llegada en los LSP que pasan por el enlace, y forzar a 0 los LSP que no pasan por el enlace en cuestión, ya que éstos restarían capacidad en los nodos previos, afectando a los que sí pasan.

El siguiente paso diferencia según la cantidad de LSPs importantes en el enlace en estudio. Si hay un único LSP importante, el problema de minimización es trivial, consiste en tomar $I_m(C)$ donde m es el flujo asociado a ese LSP y C la capacidad del enlace. En este caso, la dimensión del problema es 1 por lo que el coeficiente de corrección se calcula utilizando el teorema de Bahadur-Rao y se devuelven los resultados.

En caso de que la dimensión del problema sea mayor que uno, es decir, hay más de un LSP importante, se crea una instancia de la clase Minim que se encarga de resolver el problema de optimización asociado al cálculo de la tasa de pérdida. Con el objetivo de no realizar gastos innecesarios de memoria y no generar redundancia indeseada, no se guarda referencia a dicha instancia de la clase *Minim*. A continuación, se calcula el coeficiente de corrección utilizando el Teorema de Iltis y se devuelven los resultados.

En los dos casos anteriores el enlace que se esta resolviendo utiliza el método **getI** de la clase *Minim*, que se encarga de realizar los cálculos correspondientes en cada caso. Por último el método **resolver** recupera de la instancia de *Minim* los valores de la probabilidad de overflow, la tasa de pérdida (I), el punto en que se alcanza el mínimo ($xAst$) y el valor del coeficiente de corrección de Iltis.

En todos los casos anteriores el enlace crea un objeto de la clase *ResolverEnlace* donde almacena todos los parámetros antes descritos que obtiene de la instancia de *Minim*, junto a las observaciones que según el caso surgen.

obtenerResultados

Este método permite recuperar de una instancia de enlace los resultados obtenidos de las resoluciones y en caso que el enlace no haya sido resuelto aún, se llama el método **resolver**.

8.4. 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.

La clase *Lsp* es la primera que no representa ningún componente físico de la red y se encuentra en el segundo nivel de abstracción que se mencionó anteriormente. Cada *Lsp* consta de una concatenación ordenada de enlaces, a la cual se le agrega el *Nodo* desde el cual se origina el *Lsp*, con el fin de definir el sentido en el que se recorre el mismo. Estos parámetros son constitutivos del *Lsp*, por lo cual son imprescindibles para la construcción de las instancias de esta clase. Por otra parte, cada *Lsp* determina el camino que seguirá un determinado flujo de tráfico dentro de nuestra red. Es por esto que cada *Lsp* tendrá asociado una instancia de la clase *Flujo*, que caracterizará los requerimientos del tráfico cursado a través del *Lsp* en cuestión.

8.4.1. Campos principales

enlaces Como ya se mencionó los LSP están constituidos por una serie de enlaces ordenados que son almacenados en este campo, en un arreglo de instancias de *Enlace*.

nombre Al igual que el resto de los elementos de red antes descritos, los LSP tienen un campo en que se almacena un identificador de los mismos.

nodos De la misma manera que se almacenan todos los enlaces que conforman el LSP, se hace lo propio para los nodos, utilizando un contenedor adecuado para esta tarea. La principal función de dicho campo, aparte de proveer una descripción del LSP es la de proveer redundancia para poder realizar chequeos de continuidad sobre el camino.

nOrigen Como ya se adelantó, los LSP representan los caminos por donde circulará el tráfico en la red, y todos ellos tienen definido un sentido de circulación. Para este propósito, cada instancia de la clase tiene registro del primer nodo del camino, permitiendo identificar de manera sencilla e inequívoca la dirección del flujo de datos sobre el LSP.[2]

flujo La principal función del LSP es la de proveer el camino por donde circula el tráfico en la red, por lo que cada uno tiene asociada una instancia de la clase *Flujo*, almacenada en este campo.

tasa En la sección 8.3 se presentaron las funciones **gTasa** y **gMazumdar**. Este campo es utilizado durante la evaluación de dichas funciones para definir, en los extremos de los LSP, el valor de la tasa de tráfico.

cantFuentes Cada uno de los LSP de la red, conformará el agregado de número N de fuentes de trabajo, que comparten alguna característica. Este campo almacena dicho valor, que luego será utilizado por otras clases para la estimación de la probabilidad de overflow.

8.4.2. Constructores

En la clase se proveen dos constructores, cuya única diferencia radica en el hecho que se puede crear un LSP sin definir a la vez la instancia de la clase *Flujo* asociada, lo que se debe realiza posteriormente mediante el método **agregarFlujo**.

De esta manera, cada instancia de esta clase se debe crear pasando como parámetros la lista de enlaces que la conforman, en nodo origen y el nombre que se le desea asignar. Aparte de los anteriores el segundo constructor toma como parámetros adicionales los referentes al tráfico, es decir, la cantidad de fuentes y una instancia de *Flujo* que representa el tipo de tráfico que circula por el LSP.

Las funciones que realizan cada uno de los constructores son de inicialización, de registro y de verificación. Entre las primeras, se encuentra toda la inicialización de los campos antes descritos a partir de los datos a la entrada. A su vez, cada LSP registra su instancia en los enlaces por los que pasa. Por último se realiza una verificación de la continuidad del mismo, mediante el método **coherencia**.

[2]Observar que esto es imprescindible sobre todo en el caso de utilizar LSP con un único enlace.

8.4.3. Métodos de acceso y definición de parámetros

Para todas las variables de cada LSP se proveen métodos que permiten recuperar su valor desde cualquier otro objeto de la arquitectura. A su vez, para el caso de la variable **tasa** existe un método que permite definir su valor, lo que resulta imprescindible por la función que ésta cumple.

8.4.4. Métodos relacionados con la topología

Se implementaron varios métodos que permiten obtener información constitutiva del LSP, así como realizar modificaciones sobre el mismo, teniendo en cuenta las repercusiones que éstas puedan tener sobre las demás entidades del programa.

enlaceAnterior y enlaceSiguiente Estos dos métodos reciben como parámetro de entrada uno de los enlaces del LSP y devuelven el enlace anterior y el siguiente respectivamente. En ambos casos se realiza la comprobación que el enlace pertenezca al LSP y en caso que no lo haga genera una excepción del tipo *PedidoNoCoherente*.

nodoAnterior De manera similar a lo que ocurre con los enlaces, a partir de un nodo pasado como entrada, este método devuelve en nodo anterior en el LSP, realizando las verificaciones correspondientes.

reemplazarEnlace Para aquellos casos que se desee realizar cambios en la estructura del LSP, se implementó un método que realiza los cambios correspondientes en la lista de enlaces, tomando como parámetros de entrada el enlace a ser reemplazado y el que tomará su lugar. A su vez, este método modifica todos los registros existentes en los enlaces afectados por el LSP para que estén de acuerdo con la nueva estructura del mismo, utilizando los métodos **registrarLsp** y **olvidarLsp**

8.4.5. Métodos relacionados con el cálculo de la probabilidad de overflow

En la implementación de la herramienta, existe la posibilidad que un usuario solicite la resolución de un LSP completo, indicando así que se desea conocer la probabilidad de overflow. En este caso la clase cuenta con el método **resolverLSP** que al ser invocada devuelve un objeto de la clase *ResultadosLsp* conteniendo lo que se obtiene de la minimización de todos los enlaces del LSP.

El método *resolverLsp* realiza los pedidos de resolución a cada uno de los enlaces del LSP, almacenando los resultados de cada uno en un arreglo, para luego crear el objeto de la clase *ResultadosLsp* con dicho arreglo. La clase no cuenta con una variable donde se almacena localmente la referencia a dicho objeto de resultados, por un tema de coherencia. Debido a que las resoluciones de enlaces se realizan independiente de las de los LSP y los enlaces pueden sufrir variaciones en cualquiera de sus parámetros, puede ocurrir que una instancia de LSP contenga un objeto con sus resultados no actualizados. Podrían haber ocurrido cambios en alguno de los enlaces, provocando nuevos resultados, sin que el LSP tenga conocimiento de éstos. Por lo tanto se optó por que el LSP recupere los resultados de cada enlace cada vez que el usuario desee conocer los resultados de éste. Lo anterior no provoca cálculos innecesarios, ya que como se mencionó en la sección 8.3, cada enlace realizará las estimaciones de la probabilidad de overflow solamente en el caso que no las tenga almacenadas aún.

8.4.6. Métodos auxiliares

agregarFlujo Como ya se mencionó, es posible que el objeto flujo sea asociado al LSP posteriormente a la creación de éste, lo que se realiza mediante este método.

coherencia Para evitar que surgan problemas durante la resolución de un LSP, causando gran cantidad de cálculos innecesarios, en el constructor del LSP se realiza el chequeo de continuidad antes mencionado. Esto es posible invocando a este método, que devuelve una variable lógica cuyo valor indica la coherencia del LSP.

8.5. La clase Flujo

Esta clase describe las propiedades y los requerimientos de los distintos tipos de tráfico que ingresan a la red en estudio. Almacena las características estadísticas de una fuente representativa del agregado que circula por un camino de la red.

El objetivo de la misma es proveer un marco de trabajo definiendo las funcionalidades principales que la clase debe tener, para luego construir otras clases que hereden a partir de ésta. Fue especialmente diseñada para uniformizar la interfaz que se presenta al resto de las clases implementadas. Se pretende que otros modelos específicos de tráfico se implementen como clases que heredan de *Flujo*, reescribiendo aquellos métodos para los cuales existan formas más específicas de cálculo, que tengan en cuenta las hipótesis del modelo particular. Como consecuencia de esto, esta clase debe realizar una implementación genérica de dichas funciones, que sean aplicables a los tipos de tráfico con hipótesis generales. Las instancias de esta clase representan entonces tipos de tráfico para los cuales el usuario desconoce el modelo que los rige.

La principal forma en que pueden construirse instancias de la clase *Flujo* (o sus herederas) es a partir de trazas de tráfico. Es por ello que la clase provee métodos para leer datos desde archivos, almacenarlos y luego realizar cálculos estadísticos a partir de los mismos. En particular, a los efectos de la herramienta implementada, se incluyeron métodos para algunos cálculos básicos, así como aquellos relacionados con la teoría de grandes desviaciones.

A continuación se describen más en detalle cada uno de los componentes.

8.5.1. Campos principales

Todos los campos aquí descritos tienen visibilidad de alcance **protected** para facilitar la implementación de clases que extiendan a *Flujo*.

traza Consiste en un arreglo con muestras de la cantidad de trabajo que ingresa a la red para una realización particular del proceso de llegada. Las muestras se asumen realizadas a intervalos regulares, y se considera que dicha tasa de muestreo es la misma para todas las instancias con que se trabaja.

mu Campo donde se almacena la tasa media de arribo de trabajo del tráfico representado.

var Campo donde se almacena la varianza de la tasa de arribo del tráfico representado.

maximo Campo donde se almacena la tasa máxima de arribo de trabajo del tráfico representado.

coef Consiste en un arreglo que almacena los coeficientes del polinomio utilizados en el método **funcVelAprox** para aproximar la función de velocidad del tráfico. Se almacenan como una variable global de la instancia para evitar su recálculo lo que sería completamente ineficiente.

8.5.2. Constructores

Se proveen diferentes constructores para esta clase. En primer lugar se define un constructor por defecto para permitir la herencia. Luego se proveen constructores para crear instancias a partir de trazas de tráfico, que es la única forma general de recabar información estadística sobre un tráfico para el cual no se dispone de un modelo. Se implementó un primer constructor que recibe como parámetro el arreglo de valores que constituye la traza. Si no se dispone de dicho arreglo, se implementó un segundo constructor que recibe como parámetro la ubicación de un archivo desde donde se leerán los datos, y utiliza el método **leerTraza** que se describe más adelante. En el caso de que la ubicación no pueda ser leída, se produce una excepción del tipo *ErrorCompilacion*.

Todos los constructores (salvo el por defecto, que no realiza acción alguna) establecen los valores de los campos **mu**, **var** y **maximo** utilizando los respectivos métodos. Se pretende que aquellas clases que heredan de *Flujo* mantengan este comportamiento en los constructores.

8.5.3. Métodos de intercambio con el sistema

En la implementación actual se incorpora un único método de intercambio con el sistema, el método **leerTraza**, que recibe una ubicación (relativa o absoluta) de un archivo de texto, devolviendo un arreglo de valores que representa un vector con las muestras de llegada de trabajo.

El archivo de texto debe contener cada muestra de la traza en un renglón, formando una única columna y solo puede contener números que representan el trabajo total arribado en el intervalo de muestreo.

Se implementó como un método estático ya que no depende de ninguna instancia de la clase, solo provee funcionalidad auxiliar.

8.5.4. Métodos que realizan cálculos estadísticos sobre el tráfico

Aquí se agrupan todos aquellos métodos que realizan algún tipo de cálculo estadístico sobre la traza de tráfico. Estos son utilizados para realizar luego estimaciones sobre la performance de la red al cargarla con fuentes de trabajo que respondan a este tipo de tráfico.

largo

Devuelve el tamaño de la traza (cantidad de unidades de tiempo que abarca la misma).

media

Devuelve una estimación de la tasa media de llegada de trabajo, que consiste en el promedio de los valores muestreados en la traza. Asume que el proceso de llegada de trabajo verifica ciertas hipótesis mínimas de ergodicidad para garantizar la convergencia de la media muestral al valor medio.

varianza

Devuelve una estimación de la varianza de la tasa de arribos (o potencia del proceso) utilizando un estimador estándar basado en el promedio de las desviaciones cuadráticas respecto a la media estimada. Al igual que **media**, requiere hipótesis mínimas de ergodicidad para garantizar la convergencia.

maximo

Devuelve el máximo valor observado en la traza. Constituye una estimación de la tasa pico de arribo de trabajo.

effBW

Este método devuelve una estimación el ancho de banda efectivo (effective bandwidth) definido en la ecuación 3.11. Recibe como parámetros el valor del parámetro espacial (s) y temporal (t) donde se desea calcular el mismo.

En la clase *Flujo* debe proveerse una estimación no paramétrica (ver capítulo 5) de dicha magnitud, ya que no se dispone de un modelo de tráfico. Esta función podrá ser redefinida por clases que hereden de *Flujo* que dispongan de estimaciones paramétricas del ancho de banda efectivo.

La estimación no paramétrica elegida es la ya discutida en el capítulo 5 y resumida en la ecuación 5.8. Para estimar la esperanza presente en 3.11 se implementa un promedio sobre los valores de la traza, por lo que requiere hipótesis de ergodicidad, discutidas en el capítulo 5.

La escala de tiempo t pasada como parámetro se utiliza para acumular el trabajo muestreado en intervalos de largo t, que es la variable que interviene en el promedio.

Esta estimación es relativamente lenta ya que implica realizar una operación complicada sobre la traza completa, para cada valor de s y t que se desea.

lambdaprimaN

Este método implementa la estimación de la ecuación 5.17 para estimar la derivada de la función logarítmica generatriz de momentos del tráfico de arribo, que fue definida en 5.2. Esta función es útil para utilizar el método visto en la sección 5.3 para poder estimar la función de velocidad de forma no paramétrica. Se basa en un cociente de promedios sobre los valores de la traza, por lo que requiere hipótesis similares de ergodicidad que el resto de las estimaciones.

8.5.5. Métodos que implementan la estimación de la Función de Velocidad

La estimación de la función de velocidad del proceso de arribo de trabajo representado por la instancia de flujo es una de las partes centrales de la implementación de la herramienta de software. La estimación precisa y eficiente de dicha función a partir de trazas de tráfico es absolutamente necesaria para que las estimaciones de los parámetros de calidad de servicio mediante técnicas de grandes desvíos resulte precisa.

Las técnicas estadísticas disponibles para la estimación de la función de velocidad se discutieron en la sección 5.3. Dichas técnicas se basan principalmente en estimaciones derivadas del ancho de banda efectivo. Sin embargo, éstas requieren resolver una ecuación no lineal, por lo que debe estimarse el ancho de banda efectivo en para diferentes valores de s. A su vez, siendo la función de velocidad de cada flujo un componente fundamental

de la función objetivo del problema de minimización que se debe resolver, surge la necesidad de su repetida evaluación.

Como se comentó anteriormente, las estimaciones requeridas son relativamente lentas debido a que se requiere realizar operaciones sobre la traza, lo que insume mucho tiempo. Para reducir los tiempos de cálculo podrían utilizarse trazas más cortas del tráfico a la entrada de la red, pero esto provocaría serios problemas en cuanto a la calidad de los resultados obtenidos, ya que los mismos son fuertemente dependientes de la estadística de dicho tráfico y la utilización de trazas pequeñas traería aparejado malas estimaciones. Nos vemos entonces enfrentados a un compromiso entre el tiempo de cálculo y la precisión de la estimación.

Para resolver este problema, se consideró la posibilidad de evaluar la función de velocidad de dos maneras distintas, la manera más precisa pero a su vez mas costosa en tiempos de cálculo y una forma alternativa de menor precisión pero de evaluación más rápida. Se describen a continuación los métodos que controlan y efectúan este cálculo.

funcVel

Este método es el que presenta la interfaz externa al cálculo de la función de velocidad $I(x)$ de la instancia de flujo. Recibe como parámetros el valor de x donde se desea evaluar la función y un booleano que actúa como bandera indicando si se desea un resultado aproximado.

El método se encarga de derivar el control de la ejecución a los métodos **funcVelEx** para el caso en que se desee un resultado exacto o a **funcVelAprox** para el caso en que se desee una aproximación.

funcVelEx

Este método recibe como parámetro el valor x donde se desea calcular la función de velocidad $I(x)$ del tráfico, y devuelve una estimación de su valor por el método descrito en 5.3.

Para ello, primero utiliza el método **funcVelSAst** para resolver la ecuación 5.15 y hallar el punto s^* a sustituir en la definición del ancho de banda efectivo para calcular la función de velocidad. Esta es la parte lenta de la ejecución.

El resultado se utiliza para calcular el estimador de $I(x)$ de la ecuación 5.16, lo que requiere una llamada adicional al método **effBW**. Este método se encarga además de devolver $+\infty$ en aquellos casos en que x está fuera del rango de variación de la tasa de arribo, que es el valor que debe tomar la función de velocidad en este caso.

funcVelSAst

Este método resuelve la ecuación 5.15 para hallar el valor s^* que relaciona el ancho de banda efectivo con la función de velocidad. Como la ecuación que aparece es no lineal, y no conocemos las derivadas de la función que interviene (o es costosa su evaluación) se implementa el método de la secante (ver [7]) para resolver la ecuación. Esto requiere de sucesivos llamados al método **lambdaprimaN** lo que enlentece su ejecución.

funcVelAprox

Este método recibe como parámetro el valor x donde se desea calcular la función de velocidad $I(x)$ del tráfico, y devuelve una aproximación polinómica del valor del mismo.

Esta versión alternativa de cálculo de la función de velocidad consiste en aproximar la misma mediante una función polinomial, la cuál resulta de muy fácil evaluación. Luego de un estudio de la función a aproximar, se llegó a la conclusión que es suficiente utilizar un polinomio de grado 4, ya que se obtiene un excelente balance entre complejidad de cálculo y exactitud de la aproximación.

En pocas palabras el método utilizado consiste en tomar 5 puntos centrados alrededor de la media del tráfico, e imponer a un polinomio genérico a pasar por dichos puntos. Sea $p(x) = ax^4 + bx^3 + cx^2 + dx + e$ el polinomio interpolante, e $I(x)$ la función a interpolar, se planteo el siguiente sistema:

$$I(x_i) = p(x_i) = ax_i^4 + bx_i^3 + cx_i^2 + dx_i + e \quad \forall\, i = 1, \ldots, 5 \tag{8.1}$$

donde los valores de x_i se eligieron de manera que dos de ellos estén por debajo de la media del tráfico, dos por encima y uno exactamente en la media. Con esta elección de puntos nos aseguramos de tener información proveniente de todas las diferente zonas de la función. Recordar que, como se mencionó en 5, la función I es convexa y tiene un mínimo justamente en la media del tráfico, donde vale 0.

Para determinar los coeficientes del polinomio interpolante se debió resolver un sistema lineal (conocido como Matriz de Vandermonde), el cual se plantea fácilmente a partir de la ecuación 8.1.

$$\underbrace{\begin{pmatrix} x_1^4 & x_1^3 & x_1^2 & x_1 & 1 \\ x_2^4 & x_2^3 & x_2^2 & x_2 & 1 \\ x_3^4 & x_3^3 & x_3^2 & x_3 & 1 \\ x_4^4 & x_4^3 & x_4^2 & x_4 & 1 \\ x_5^4 & x_5^3 & x_5^2 & x_5 & 1 \end{pmatrix}}_{Matriz\ de\ Vandermonde} \begin{pmatrix} a \\ b \\ c \\ d \\ e \end{pmatrix} = \begin{pmatrix} I(x_1) \\ I(x_2) \\ I(x_3) \\ I(x_4) \\ I(x_5) \end{pmatrix}$$

El anterior sistema de ecuaciones se debe resolver mediante un método específico para sistemas lineales. En particular, debido a que la matriz de Vandermonde tiene un número de condición alto, lo que introduce errores en la solución, se decidió transformar el problema a través de la descomposición QR de la misma, pasando a resolver un sistema equivalente, pero con un mejor número de condición. Para implementar estos pasos se utilizaron funciones de la biblioteca de matrices *Jama*, desarrollada por el NIST[3].

La primera vez que se llama a este método, se calculan los coeficientes correspondientes a esta instancia, lo cual requiere 5 evaluaciones de funcVelEx. Sin embargo, ésto se realiza en una única oportunidad, almacenando los coeficientes hallados en el campo **coef**. Las subsiguientes evaluaciones de este método se vuelven entonces muy eficientes en tiempo, ya que la evaluación se simplifica de manera muy significativa, no siendo necesario recorrer la traza en cada evaluación.

Por supuesto que este método nos plantea el problema de la precisión de nuestros resultados, ya que se utiliza una función aproximada. Como primer dato a tener en cuenta, mencionamos que el simple hecho de utilizar una realización específica del tráfico proveniente de una fuente determinada introduce de por sí un error, producto de la aleatoriedad intrínseca del proceso, el cuál no podemos evitar de ninguna manera. Por lo tanto, en general es innecesario evaluar de manera costosa cada vez las funciones si éstas mismas ya introducían un error cuyo orden es generalmente mayor que el que aparece al utilizar el polinomio interpolante.

De todos modos, se optó por contrastar los resultados de la interpolación mediante las funciones aproximadas contra las funciones reales. En la figura 8.3 se pueden aprecias

[3]NIST: National Institute of Standards and Technology

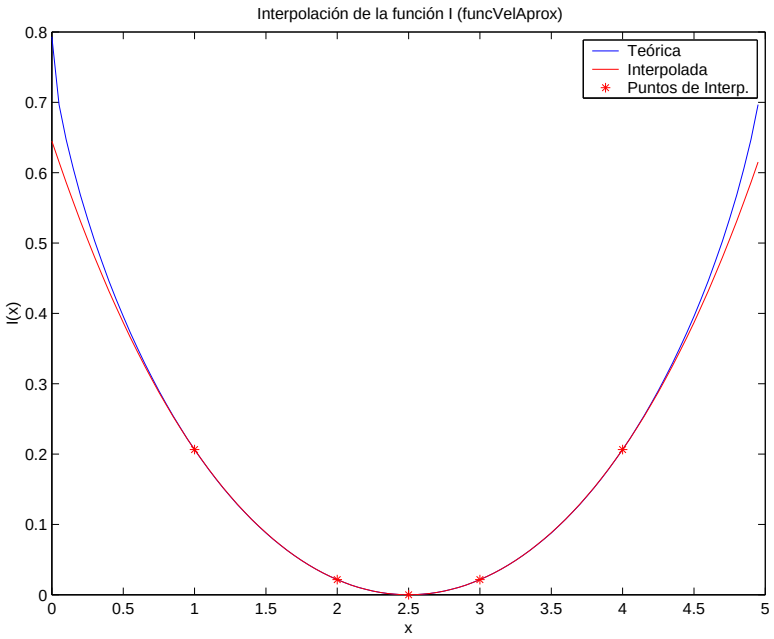

Figura 8.3: Interpolación de la función de velocidad

los resultados del contraste para un caso representativo. Allí se muestra la función de velocidad teórica y el resultado de la interpolación. Si se trabaja con trazas, no se tendrá la función de velocidad teórica sino una estimación de la misma, como se vio en 5.3. Sin embargo, las estimaciones conservan las propiedades de la función de velocidad real, por lo que el comportamiento es similar. Además, el error cometido en la interpolación es siempre mucho menor que el error de estimación, visto en 5.3.

Como se observa en dicha figura, para la zona del entorno de la media la interpolación brinda buenas aproximaciones, perdiendo precisión lejos de esta lo cual no es un problema, ya que por lo general la zona de trabajo es la primera.

8.5.6. Métodos de acceso a los parámetros

Se proveen los métodos de visibilidad pública **getMedia**, **getVarianza**, y **getMaximo** para recuperar los correspondientes parámetros estadísticos calculados y almacenados en los campos de alcance protegido.

A su vez, se dispone de un método que permite desplegar a consola la traza de tráfico. Esto de gran utilidad en los momentos de testeo, desarrollo y debugging de la biblioteca.

8.6. La Clase FlujoONOFF

Un tipo de tráfico es de particular importancia en las redes modernas es el de modelo de tráfico Exponencial ON-OFF, también llamado Flujo Markoviano ON OFF. El mismo ya fue presentado en la sección 4.2.

Para representar este tipo de tráfico en la estructura del programa, se extendió la clase Flujo, redefiniendo aquellos métodos que pueden calcularse de forma más eficiente o más exacta utilizando la hipótesis de que el tráfico consiste en un Flujo Markoviano ON-OFF

En particular, debido a que existe un modelo paramétrico para este tipo de tráfico, se agregan algunos campos y se redefinen los métodos que realizan estimaciones estadísti-

cas. A su vez, aparecen nuevos constructores. Pasamos a detallar ahora únicamente aquellos campos y métodos que presentan diferencia con la superclase *Flujo*.

8.6.1. Campos principales

Además de los campos presentes en *Flujo*, se agregan aquellos campos que contienen los parámetros del modelo. Como se presentó en la sección 4.2, dichos parámetros se resumen en 3 matrices, *pi* (vector de probabilidades en régimen de cada estado), Q (matriz de intensidades de transición entre estados) y H (matriz diagonal con la tasa de arribo de trabajo en cada estado). Para almacenarlas, se optó por utilizar la librería de matrices *Jampack* disponible en Internet, realizada por la empresa MathWorks Inc. La razón de la utilización de esta librería es que provee métodos sencillos para el cálculo de exponenciales de matrices, que es necesario para utilizar la fórmula del ancho de banda efectivo (ec. 4.2).

Es así que se agregan los campos **pi**, **Q** y **H**, implementados como matrices de la clase *ZMat* de *Jampack*.

8.6.2. Constructores

En esta clase, aparece un nuevo constructor que recibe tres arreglos bidimensionales conteniendo los valores de los parámetros *pi*, Q y H, a los efectos de construir una instancia de *FlujoONOFF* cuyos parámetros se conozcan exactamente. Esto permite utilizar la instancia para realizar estimaciones teóricas de las probabilidades de overflow, por ejemplo.

A su vez, se redefinen los constructores de la clase *Flujo* a partir de trazas para utilizar las nuevas funciones de estimación de parámetros (media, varianza, máximo), que proveen estimaciones que utilizan el modelo.

8.6.3. Métodos que realizan cálculos estadísticos sobre el tráfico

La utilización del modelo de flujo markoviano permite redefinir los métodos que realizan cálculos estadísticos en la clase *Flujo* por versiones más eficientes. La base teórica de los métodos que se describen a continuación se discutió en la parte paramétrica de la sección 5.2.

estimP

Éste método es el que sirve de base al resto de las estimaciones. Utiliza el procedimiento desarrollado en [3] y resumido en la sección 5.2 para estimar la matriz Q del flujo markoviano a partir de los saltos de una traza de tráfico. Implementa el estimador de la ecuación 5.10, y luego a partir de la matriz Q obtenida se estima el vector *pi* resolviendo la ecuación de balance.

Para estimar los valores de H se utiliza el hecho de que se trata de un FlujoONOFF por lo que las tasas de arribo de trabajo serán 0 en el estado OFF, mientras que la tasa pico se estima como el máximo observado en la traza.

Por último, los valores estimados se almacenan en los campos descritos anteriormente.

media

Devuelve la media (estimada o teórica) de la tasa de arribo de un flujo ON OFF calculada como $\pi_{ON} h_{ON}$ siendo π_{ON} la probabilidad en régimen del estado ON y h_{ON}

la tasa pico de transmisión.

varianza

Devuelve la varianza (estimada o teórica) de la tasa de arribo de un flujo ON OFF calculada como $\pi_{ON}(h_{ON})^2 - \mu^2$ siendo π_{ON} la probabilidad en régimen del estado ON, h_{ON} la tasa pico de transmisión y μ la media del tráfico.

maximo

Devuelve la tasa pico de transmisión, h_{ON}.

effBW

Una de las principales ventajas de el modelo de flujo markoviano es que posee una fórmula explícita para el ancho de banda efectivo. A su vez, como esta fórmula depende de los parámetros del modelo, es fácil construir un estimador paramétrico de esta magnitud. Este estimador, presentado en la ecuación 5.14, permite redefinir el método effBW de *Flujo* para que utilice esta fórmula, permitiendo una estimación rápida y eficiente del ancho de banda efectivo, ya que la estimación de parámetros se realiza en una única oportunidad.

lambdaprimaN

Como se dispone de un estimador rápido del ancho de banda efectivo, para estimar la derivada de la función logarítmica generatriz de momentos se optó por utilizar un cociente incremental basado en la estimación del ancho de banda efectivo, ya que la expresión paramétrica de esta función resulta en extremo complicada, involucrando una serie. Por lo tanto, se aplica el cociente incremental de la ecuación 8.2 lo que requiere dos evaluaciones de effBW, pero estas son rápidas debido a que se trata de un flujo de este tipo.

El disponer de los métodos **effBW**, **lambdaprimaN** permite acelerar la estimación de la función de velocidad. No se requiere redefinir los métodos que implementan dicho cálculo pues, como se discutió en la sección 5.3, el método de estimación de la función de velocidad no varía.

8.6.4. Métodos auxiliares

Se incorporó un método estático auxiliar, **expm**, que provee funcionalidad a la clase para calcular fácilmente la exponencial de matrices.

Recibe como parámetro una matriz A y devuelve $\exp(A)$. Se utilizó la librería *Jampack* para encotnrar un cambio de coordenadas que permitiera llevar la matriz a su forma diagonal, para luego calcular la exponencial de ésta, que es sencilla, y luego deshacer el cambio para obtener la exponencial deseada.

8.7. La clase Minim

Esta clase implementa el algoritmo de minimización descrito en el capítulo 6, así como herramientas complementarias para los cálculos de performance. Se decidió crear una clase específicamente con este cometido, basándonos en los patrones UML, más precisamente, en el patrón *experto*. La idea consiste en que dicho algoritmo esté a cargo

de una clase que conceptualmente este por encima de la red y tenga acceso a toda la información necesaria.

Con esta clase se buscó independizar los algoritmos matemáticos de las clases que implementan la red. Cuando se desee determinar la tasa de pérdida en algún *Enlace* de la red en estudio, se construye una instancia de la clase *Minim* asociada a dicho enlace y a uno de los nodos de éste (para indicar el sentido), y la misma se encarga de resolver el problema de optimización adecuado, para luego almacenar los resultados obtenidos.

Este diseño es una solución de compromiso entre la separación del código en clases independientes, y reducir la transferencia de parámetros a los métodos intervinientes. Se consideraron algunas alternativas, pero se optó por privilegiar la separación del código, lo que permitiría desarrollar más fácilmente otras posibles implementaciones de la clase *Minim*. Como efecto agregado, en este diseño no es necesario almacenar una instancia de la clase *Minim* para cada problema a resolver, sino que se crea a medida para cada problema de interés, y luego sólo se almacenan los resultados relevantes liberándose memoria. Ésto redunda en una mejora de la performance.

8.7.1. Campos principales

Enlace Este campo guarda una referencia al enlace del cual se pretende obtener la probabilidad de overflow.

lsps Este contenedor agrupa aquellos LSP de importancia para el enlace que se quiere resolver

Nodo Nodo que permite discernir el sentido del tráfico para el cual se estimará la probabilidad de overflow.

cantTotal Este parámetro indica la cantidad de fuentes generadoras de tráfico que inciden sobre el enlace en cuestión.

coefMezcla Este conjunto de coeficientes ordenados en relación a los lsp importantes, permite al algoritmo de minimización, determinar las proporciones con las cuales participan cada uno de los tráficos que afectan al enlace.

resAprox Esta variable booleana permite controlar que función de velocidad se debe utilizar para la realización de los cálculos. En todos los casos la misma se inicializa en true de modo de comenzar la inicialización utilizando la función de velocidad aproximada mediante un parámetro, la cual se describe en la sección 8.5.5. El control de esta variable, está a cargo del algoritmo de minimización, como se verá en el método minimizador.

rho Esta variable representa la constante que se utiliza para definir la función de penalidad utilizada para resolver el problema de optimización, la cual se presentó en la ecuación 6.2.

h Este parámetro define el paso óptimo para el cálculo de los cocientes incrementales utilizados a lo largo del algoritmo, para aproximar la derivadas que se hacen necesarias.

xAst Este arreglo de reales, contiene el punto de $\mathbb{R}^K$ donde se alcanza el mínimo del problema de optimización

I Este campo guarda el valor de la función de velocidad en el mínimo, en el cual se basa el cálculo de la probabilidad de overflow del enlace analizado.

C En este campo se almacena la constante de corrección asociada al problema de minimización, la cual dependerá de la dimensión del mismo, como ya se explicó en secciones anteriores.

8.7.2. Constructor

Esta clase cuenta con un único constructor, el cual recibe como parámetros el enlace del cual se quiere estimar su probabilidad de overflow y un nodo de este para indicar el sentido de interés para el cálculo.

En el proceso de construcción, se determinan los LSP del enlace recibido como parámetro, modificando el parámetro correspondiente, con lo que luego se calculan la cantidad total de fuentes y los coeficientes de la mezcla de los tráficos.

8.7.3. Métodos de acceso y definición de parámetros

Para acceder de manera sencilla a algunos de los parámetros de esta clase se implementaron los métodos `get` y `set` correspondientes. Vale la pena aclarar que con el fin de minimizar, dentro de lo posible, la interfaz de esta clase solo se implementaron los métodos imprescindibles, existiendo parámetros que no cuentan con este tipo de métodos.

En algunos casos particulares, como ser el nombre de la instancia, no se cuenta con un método para modificar el mismo de forma posterior a la creación de la instancia, ya que no se consideró como una buena política de programación.

8.7.4. Métodos que implementan el algoritmo de minimización

A la hora de abordar la implementación práctica del algoritmo de minimización, se adoptó la estrategia de dividir el problema en dos partes con el objetivo de simplificar tanto su resolución como su posterior comprensión por parte de un potencial usuario. Por una parte se creó un método que básicamente controla el ciclo del algoritmo y otro que realiza la búsqueda lineal inexacta mediante la Regla de Wolfe explicada con anterioridad. Lo anterior tiene como consecuencia directa la fácil reutilización del código, ya que en caso que se desee modificar el algoritmo de búsqueda lineal inexacta, solo bastaría con reemplazar el método correspondiente.

minimizador

Este es el método que controla el ciclo utilizado para la resolución del problema de optimización. Este comienza por determinar las medias de los flujos que atraviesan el enlace con el objetivo de proporcionar al algoritmo un punto de partida factible. Esto se hace multiplicando dichas medias por un factor aleatorio cercano a 1, lo que genera el primer arreglo de valores en donde evaluar la función objetivo y desde donde realizar la primera búsqueda lineal. Es a partir de aquí que se comienza a transitar el camino ya descrito en la presentación teórica del algoritmo, sobre la cual se profundizará a continuación en la implementación de la misma.

Una vez iniciado el algoritmo seguirá iterando según lo establecido en el diagrama de flujo de la figura 6.2 mientras no cumpla con los requerimientos allí descritos. Desde el punto de vista de la programación, la continuidad del ciclo está supeditada a las condiciones de parada, estas son aquellas condiciones que de satisfacerse por el iterando actual (x_k), éste será óptimo. La primera condición, es la de la factibilidad de x_k, condición totalmente necesaria para que x_k sea óptimo del problema P. Las otras dos

condiciones que se utilizaron para verificar la optimalidad son en los valores de los $\{x_k\}$, es decir, la sucesión de iterados, y los valores de $\{\phi_\rho(x_k)\}$, es decir la sucesión de valores funcionales.

En particular, para los valores de cada estimación del x^* se utilizó la siguiente condición:

$$\frac{\|x_{k+1} - x_k\|}{\|x_{k+1}\|} < \beta_1$$

donde $\beta_1 \in \mathbb{R}$ es una tolerancia fijada de acuerdo a la forma de las funciones en cuestión, que en el programa está representada por la variable local **tol1**. En pocas palabras, la anterior condición establecía que en caso que dos valores sucesivos de estimados estuvieran suficientemente próximos, se consideraba que al algoritmo esta cerca de su punto límite. Con el objetivo de tener una mayor certeza a cerca de la convergencia del algoritmo, la anterior como las subsiguientes tolerancias se tomaron relativas al valor del último iterado.

Para los valores de la función objetivo, la condición que se utilizó es muy similar:

$$\left|\frac{\phi_\rho\left(x_{k+1}\right) - \phi_\rho\left(x_k\right)}{\phi_\rho\left(x_{k+1}\right)}\right| < \beta_2$$

donde $\beta_2 \in \mathbb{R}$ es la tolerancia correspondiente a esta segunda condición, la cual se representa por la variable local **tol2**. En este caso lo que se buscaba era muy similar a la condición de los x_k, pero es de gran importancia que ambas condiciones coexistan, ya que podría ocurrir que si bien dos valores sucesivos de x están muy próximos, los valores funcionales de ϕ_ρ en estos puntos todavía sean muy diferentes. Esto significa que si bien el algoritmo esta avanzando en pasos muy pequeños, lo hace significativamente en cada uno de ellos, en cuando a mejoras en los valores de la ϕ_ρ se refiere.

De lo anterior se desprende la manera en que se llevó a cabo la implementación de dichas condiciones de parada. Para que el algoritmo termine se deben cumplir a la vez las tres condiciones, no alcanza con que se cumpla alguna(s) de ellas por separado.

Como ya se adelantó en la descripción de parámetros, este método es el encargado de controlar que función de velocidad se utiliza en cada caso. En la sección 8.5.5 se presentó una interpolación que permitía aproximar la función de velocidad (I). Esta tenía como objetivo reducir el tiempo de procesamiento en cada evaluación de la misma. Utilizando la anterior aproximación, se llevó a cabo una alternativa más eficiente para la implementación del algoritmo de minimización. La idea general de la misma consiste en comenzar la iteración utilizando la función aproximada (**funcVelAprox**), y de esta forma aproximarse rápidamente al mínimo buscado. Una vez cerca de este se pasa a utilizar la versión más precisa (**funcVelEx**). El control de cual de las versiones se selecciona se realiza mediante mediante el booleano **resAprox**, el cual si es true, indica que debe usarse la aproximación polinómica, en caso contrario se deberá utilizar la versión exacta.

Al inicio de la iteración **resAprox** siempre es true, una vez que se llega al óptimo del problema mediante las aproximaciones polinómicas, se verifica que cumpliera las condiciones de convergencia mediante las funciones reales. En los casos en que esto no sucede, se comienza a iterar nuevamente a partir de dicho punto, pero ésta vez utilizando las funciones reales, para esto se cambia el valor de **resAprox** a false. Lo que se observó, fue que en los pocos casos en que el óptimo aproximado[4] no lo era también de manera exacta, solamente eran necesarias unas pocas iteraciones para encontrar el óptimo exacto.

[4]Llamemos óptimo aproximado a aquél obtenido mediante las funciones aproximadas

Todo lo anterior nos permitió implementar un algoritmo que de manera rápida encuentra el óptimo o al menos un punto muy cercano él. A pesar de utilizar funciones aproximadas, podemos asegurar que el óptimo obtenido es tan preciso como el que se obtendría en caso de iterar en todo momento con las funciones exactas, ya que siempre se verifican las condiciones de parada utilizando estas últimas. Además, el algoritmo posee la ventaja adicional de que el tiempo de ejecución se redujo sensiblemente (alrededor de 40 veces en algunos casos).

busqueda

Este método implementa la búsqueda lineal inexacta mediante la Regla de Wolfe descrita en la sección 6.2.4. Para esto el método recibe como parámetros el iterando actual x_k y la dirección de descenso dada por el opuesto del gradiente, devolviendo el desplazamiento a realizar sobre esta dirección para reducir el valor de la función objetivo.

Como primera medida, se calcula la pendiente en el punto de partida con el objetivo de calcular las dos pendientes auxiliares que determinarán la región de aceptación. Luego la extrapolación inicia la búsqueda, la cual continua hasta encontrar un candidato que cumpla las condiciones antes descritas.

Con el objetivo de evitar que la búsqueda se prolongue excesivamente en pos de conseguir una precisión innecesaria y para que el algoritmo general converja aun en condiciones netamente desfavorables, se implementó una mejora a la regla de Wolfe, que consistió en agregar lo que comúnmente se conoce como una condición se seguridad a dicha búsqueda. Más específicamente, en caso que la función $h(\alpha_k)$ sea una función convexa[5], el mismo no tiene problemas, pero en el caso que la función sea muy irregular, puede ocurrir que la búsqueda nunca termine. Por lo tanto la condición que se agregó para el fin de la búsqueda fue que el tamaño del intervalo $[t_g, t_d]$ sea mayor a una cierta tolerancia prefijada. Esto nos permite terminar la búsqueda en el caso que nuestro valor de α_k no estuviera dentro de la zona a, pero sin embargo los valores de t_g y t_d fueran tan próximos que esto nunca ocurriera.

Vale la pena mencionar que en el caso la búsqueda terminara debido a esta nueva condición, el nuevo punto $x_k + \alpha_k p_k$ sería un mejor punto que el x_k ya que la dirección p_k es de descenso siempre. Lo que ocurre en estos casos es que el punto que se obtiene no esta dentro de la zona que se considera aceptable al principio de la búsqueda, pero en ningún caso es peor que el valor que se tenía antes, por lo que la convergencia global no se ve mayormente afectada.

8.7.5. Métodos relacionados con el algoritmo de minimización

La clase *Minim* también implementa las funciones que serán utilizadas por el algoritmo, como la función objetivo (aquella que se quiere minimizar) y la función que impone las restricciones.

funciónI

Este método recibe como parámetro un arreglo de reales ordenados, en cuyas componentes se quiere evaluar la función de velocidad de cada uno de los Lsp importantes que pasan por el enlace a resolver.

[5]Recordar que la función I a minimizar es convexa, pero el efecto de las restricciones puede causar que h no lo sea

Una vez evaluadas las funciones de velocidad en los puntos correspondientes, este método devuelve la suma de dichos valores ponderados por los coeficientes de mezcla, lo que constituye la función objetivo del problema de minimización.

Vale la pena aquí dedicar unas palabras a los coeficientes de mezcla. Estos coeficientes que suman 1, representan como ya se adelantó la proporción en la que influyen cada uno de los flujos involucrados. Esto permite en una posterior etapa calcular la probabilidad de overflow en un enlace en función de la cantidad total de fuentes que intervienen.

geM

Este método establece la restricción para el problema de minimización que permite generar la función de penalidad. Para esto utiliza la función **gTasa** de la clase *Enlace* evaluándola en un arreglo de reales que recibe como parámetro y compara el resultado obtenido contra la capacidad del enlace en cuestión retornando

$$\text{máx}\{C - g(x_1 \dots x_k), 0\}$$

con lo que si la función g es mayor que la capacidad del enlace no habrá penalidad y la misma será proporcional a la diferencia en caso contrario.

Como comentario final sobre este método, es necesario destacar que se implementó de forma que el parámetro lo pueda recibir tanto como un arreglo de reales o como un objeto matriz mediante la utilización de la librería Jampack.

apenal

Este método implementa la función de penalidad que realmente se utiliza como función objetivo, para transformar un problema con restricciones en un problema que no las tiene. Para esto recibe un vector[6] sobre el cual evaluar ambas funciones anteriores para implementar la función de penalidad descrita en 6.2.

8.7.6. Métodos que calculan las constantes de corrección

El método **calculoIltis** calcula, en base al desarrollo teórico de la sección 3.2, la constante de corrección para la probabilidad de overflow que allí se presenta. El método **bahadurRao** hace lo propio para problemas de dimensión uno. Ambos métodos reciben el vector donde se produjo el mínimo de la suma de las funciones de velocidad, donde deberán evaluar las funciones y derivadas correspondientes para realizar los cálculos correspondientes.

Por otra parte el método **constanteDeCorreccion** recibiendo el vector donde se produce el mínimo y el valor de la I determina a que método invocará en función de las dimensiones del problema y en caso de que el valor de la función I sea infinito por alguna razón, simplemente devolverá 0 ya que la corrección no tiene sentido.

8.7.7. Métodos que estiman las derivadas

En sucesivas oportunidades, se hace necesaria la estimación de las derivadas de las funciones involucradas en el problema. Como se puede apreciar existen métodos especializados para realizar las derivadas de distinto orden de las funciones que lo requieren, como ser **gradApenal**, **gradI**, **gradG**, **hessI** y **hessG**. En general estos métodos reciben como parámetro un vector donde evaluar la función, ya sea en forma de arreglo

[6] Al igual que en el método **geM**, este parámetro se puede pasar como un arreglo o como un objeto matriz de Jampack

o como matriz, y devuelven el valor de la derivada solicitada. Como se puede apreciar, al trabajar en problemas de dimensión mayor a 1, el cálculo de una derivada se traduce en cálculos de gradientes y hessianos. Todos estos métodos se llevaron a cabo mediante la implementación de la conocida aproximación de los cocientes incrementales, que se describe a continuación.

Consideremos una función escalar $f : \mathbb{R} \to \mathbb{R}$. La aproximación de la derivada primera de f en un punto x por cocientes incrementales es:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} \tag{8.2}$$

siendo h un parámetro que se estima para cada función.

Para la derivada segunda la aproximación se muestra a continuación:

$$f''(x) \approx \frac{f(x+h) + f(x-h) - 2f(x)}{h^2} \tag{8.3}$$

Si extendemos el concepto anterior a funciones de varias variables, la idea es la misma y consiste en aplicarle los desplazamientos de la ecuación 8.2 a la variable respecto de la cual se desea derivar, manteniendo fijas el resto de las variables. Para el caso de una función de dos variables $g : \mathbb{R}^2 \to \mathbb{R}$, la derivada con respecto a la primer variable en el punto (x, y) queda:

$$g_x(x, y) \approx \frac{g(x+h, y) - g(x-h, y)}{2h} \tag{8.4}$$

La derivada segunda con respecto a la misma variable en el mismo punto queda:

$$g_{xx}(x, y) \approx \frac{g(x+h, y) + g(x-h, y) - 2g(x, y)}{h^2} \tag{8.5}$$

Finalmente solo resta estimar las derivadas cruzadas de dicha función en el mismo punto, lo cual se muestra a continuación, la que resulta imprescindible para el cálculo de los hessianos de las funciones **G** e **I**:

$$g_{xx}(x, y) \approx \frac{g(x+h, y+h) + g(x-h, y-h) - g(x-h, y+h) - g(x+h, y-h)}{4h^2} \tag{8.6}$$

Es recurrente en todos los casos anteriores la presencia de la variable auxiliar h, la cual determina el desplazamiento respecto del punto sobre el cual se desea calcular la derivada, que nos permite realmente calcular el cociente incremental. La determinación de este parámetro no es trivial ya que una mala elección del mismo puede llegar a provocar una mala estimación de la derivada, lo que implicaría tener una pobre noción del comportamiento de la función en el entorno del punto elegido. Para todos los casos anteriores se realizó un estudio del valor óptimo del parámetro h, que minimiza el error de la aproximación, dependiendo de la función a derivar.

8.7.8. Métodos auxiliares

Con el objetivo de simplificar el código y bajo la consigna "divide y vencerás"se implementaron algunos métodos que permiten realizar operaciones con arreglos de números (vectores) como ser sumar un arreglo más una constante escalar por otro (**xMastd**), calcular la norma euclideana de un vector (**norma**) o de la diferencia de dos de éstos (**normaDif**).

Capítulo 9

Interfaz de usuario

Para diagramar una determinada red y para que sobre ella las bibliotecas de software especializadas puedan realizar los cálculos solicitados, es necesaria la generación de un archivo de texto con un formato específico.

El package *InterefazSistema* cuenta con los métodos necesarios para reconocer y ejecutar los comandos incluidos en el antedicho texto, así como también para generar el archivo que contiene el reporte de los resultados obtenidos.

Con el único objetivo de simplificar la tarea del usuario, se desarrolló una interfaz de usuario, que básicamente permite de un modo gráfico, generar la topología deseada, configurar las propiedades de los componentes de la misma, definir caminos virtuales, seleccionar los cálculos a realizar y obtener el reporte de resultados. Esta interfaz, a partir del archivo gráfico, genera el archivo de texto correspondiente para que el package *InterefazSistema* lo interprete y luego despliega en pantalla el archivo de reportes obtenido

En este capítulo se describen los componentes y las funcionalidades del package *InterefazSistema*. En particular, en la sección 9.1 se describe la clase que se encarga de interpretar los comandos del usuario. En la sección 9.2 se describe el formato que debe seguir el archivo de texto que define la red, y el formato que tiene el archivo de salida. Para finalizar en la sección 9.3 se describen los componentes de software que conforman la Interfaz de Usuario y se da una breve guía de como utilizarla.

Interfaz del Sistema

El package que da nombre a esta sección, se compone de cuatro clases, cuya interacción permite interpretar el archivo de comandos. En la figura 9.1 se puede apreciar el diagrama UML de la interacción de las clases que conforman el mencionado package.

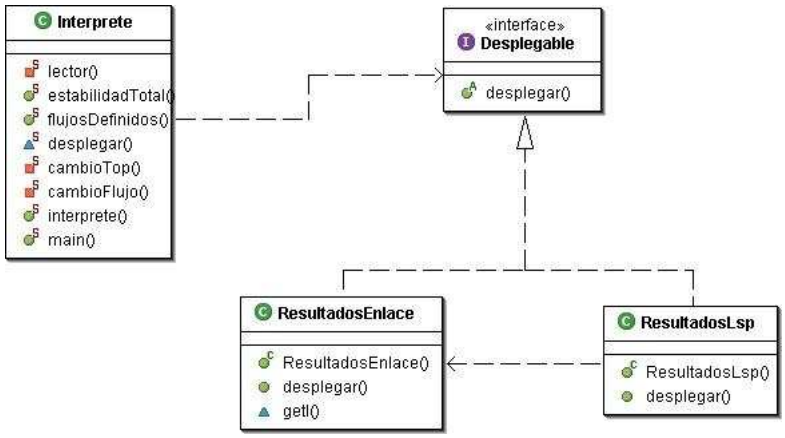

Figura 9.1: Diagrama UML del package InterfazSistema

De ahora en más, se pasará a describir de forma separada la clase principal del package y dentro de ésta se hará hincapié en sus principales métodos, y cómo colaboran entre sí. El resto de las clases solo se utiliza para almacenar resultados y su funcionamiento puede comprenderse a partir del Javadoc.

9.1. La clase Intérprete

El objetivo de esta clase es proveer métodos que permitan al usuario construir una cierta topología de red y realizar los cálculos que desee sobre la misma. Para ello, implementa métodos estáticos que permiten recibir comandos predefinidos desde un archivo de texto, y actuar en función de ellos .

9.1.1. Campos

nodos, enlaces, lsps, flujos Estos contenedores permiten almacenar las instancias y referencias a las mismas, de las clases de igual nombre a medida que se van creando a partir del archivo de comando. Esto permite ir construyendo la red según lo estipulado y acceder a dichas instancias cuando sea necesario.

resultadosArray Este contenedor almacena de forma ordenada los objetos **resultado** que se generan a partir de cada estimación, para su posterior inclusión en el reporte de salida.

redEstable Esta variable booleana es utilizada para determinar si existen en la red, enlaces que no cumplen con las condiciones de estabilidad requeridas. Por defecto siempre se inicializa en true y será modificada por el método que chequea la estabilidad de la red.

hayFlujos Esta variable booleana que se inicializa en falso, indica si el usuario por error definió algún LSP al cual no le asoció ningún flujo. Esto será verificado por el método correspondiente.

in Esta variable de tipo *BufferedReader* es la que nos permite en JAVA leer líneas desde un archivo de texto, y guardar esto como un string, sobre el cual trabajaremos.

out Esta variable de tipo *PrintWriter* es la opción elegida para imprimir en el reporte de salida los resultados obtenidos.

El método lector

Este método que se encarga de leer el archivo de comandos, analizar cada uno de ellos e invocar a los métodos adecuados. Para esto nos basamos en el uso de expresiones regulares ("regular expressions") que proporcionan las librerías de JAVA, que permiten detectar aquellas líneas que comienzan con alguna palabra predeterminada y guardar el resto de las palabras que le siguen. De esta forma se puede reconocer los comandos de creación de los componentes de la red y crear las instancias correspondientes de los objetos de la red, del package *Topología* almacenando referencias a dichas instancias, para poder acceder a ellas posteriormente. En este proceso se setean las propiedades de las instancias en los valores específicados por el usuario, o en caso contrario, deja los valores por defecto correspondientes.

Una vez finalizada la diagramación de la red se procede a reconocer los comandos que apuntan a realizar cálculos, lo que desencadena una serie de procedimientos previos, que tienen como objetivo asegurar la correcta ejecución del programa.

Como primera medida se verifica la correcta definición de la red, contemplando todas las posibles fallas estipuladas en la sección 10.2. Una vez superada esta etapa se procede a comprobar la estabilidad de la red definida, teniendo como referencia, las situaciones descritas en la sección 10.3. Recién cuando la etapa de validación de la red, fue satisfactoriamente culminada, se procede a realizar los cálculos solicitados por el usuario los cuales se almacenan en el contenedor antes descrito.

9.1.2. Métodos que verifican la correcta definición de la red

estabilidadTotal

Este método una vez definida la red recorre todos los enlaces existentes en la misma y verifica que estos cumplan con las condiciones de estabilidad requerida para el correcto funcionamiento del programa. En caso de encontrar algún enlace que no verifique dichas condiciones, modifica el valor de la variable redEstable a falso lo que genera una excepción de tipo *SistemaInestable*.

flujosDefinidos

Este método se encarga de verificar que todos los LSP definidos tengan asociado una instancia de la clase *Flujo*. Para ello recorre el contenedor **lsps** y en caso de encontrar algún LSP sin flujo asociado generará una excepción del tipo *flujoNoExistente*.

9.1.3. Método de acceso al programa

Interprete es el método encargado de controlar el flujo de ejecución del programa y a partir del cual el usuario accede a ejecutar su archivo de comandos.

Éste método recibe como único parámetro la ubicación del archivo de comandos y se encarga del control de la ejecución del programa. Como primera acción le indica al método **lector** la ubicación del archivo de comandos desde donde se cargará la red. Durante la ejecución de **lector** este método provee de contención para las posibles excepciones que puedan surgir y una vez concluidos los cálculos, se encarga de generar el archivo de reportes en función de los resultados obtenidos. Para el almacenamiento de dichos resultados, se desarrollaron las clases *ResultadosEnlace* y *ResultadosLsp* las cuales implementan la interfaz Desplegable. Estas tienen como objetivo la fácil recuperación de los resultados y su posterior despliegue en el archivo de salida.

9.1.4. Métodos destinados a detectar cambios en la topología de la red

cambioTop

Este método recorre el contenedor de los enlaces y modifica todos aquellos parámetros de cada instancia que indican que los resultados almacenados son actuales. Con esto se logra que de haber cambios en la topología, los resultados almacenados sean descartados, y se calculen nuevamente.

cambioFlujo

Este método al igual que el anterior desecha los resultados existentes en caso de que existan cambios en los flujos que atraviesan la red.

olvidarTopología

Este método como su nombre bien lo indica, permite borrar de los contenedores todos los componentes de una determinada topología. Esto impide que el usuario sobrescriba dos redes distintas con los obvios resultados que esto traería aparejado.

9.2. Formato de los archivos

Como ya se mencionó, las bibliotecas básicas del programa, reciben comandos via texto y los resultados obtenidos son almacenados en archivos del mismo tipo. A continuación, se describirá en detalle el formato que debe seguir el archivo de comandos.

9.2.1. Formato del archivo de entrada

La primera parte del archivo estará dedicada a la construcción de la red. Para ello es necesario ir definiendo uno por uno los elementos que la componen.

Para crear un nodo, solo es necesario indicarlo, iniciando la sentencia con la palabra `nodo` y luego asignarle un nombre el cual debe ir a continuación, separado por un espacio. Por ejemplo a continuación se crean tres nodos, de forma de comenzar la construcción de una red de topología sencilla.

```
nodo primerNodo
nodo segundoNodo
nodo tercerNodo
```

Para poder definir correctamente un enlace se utiliza la palabra clave `enlace` y es imprescindible la previa definición de los nodos que serán conectados por este, como se explicó anteriormente. La sentencia de creación de un enlace es un poco más compleja ya que además de asignarle un nombre, es necesario definir los nodos que serán origen y fin del mismo (el nodo ingresado en primer lugar definirá el sentido positivo), así como la capacidad del mismo. Es importante notar que la capacidad del enlace debe estar expresada en concordancia con los parámetros de los flujos intervinientes. En particular, debe estar en la misma unidad que las trazas muestreadas que se utilicen para construir instancias de *Flujo*.

A modo de demostración y continuando con el ejemplo anterior se definirán dos enlaces, el primero conectará los Nodos primerNodo y segundoNodo, definidos anteriormente con una capacidad de 8 paquetes por unidad de tiempo. El segundo *Enlace* unirá los Nodos segundoNodo y tercerNodo con una capacidad de 6 paquetes por unidad de tiempo.

```
enlace enlUno primerNodo segundoNodo 8
enlace enlDos segundoNodo tercerNodo 6
```

Para definir caminos sobre la red, se utiliza la palabra clave `lsp`. Se debe indicar como parámetros el nombre del LSP, en nuestro caso `lsp1`, luego el nodo donde se origina el mismo (`primerNodo`), entre corchetes y separados por un espacio, la lista ordenada de nombres de aquellos enlaces por los que pasa el LSP. Por último, la cantidad de fuentes generadoras de tráfico que conforman el agregado que circula por el LSP.

A modo de ejemplo construiremos tres LSP utilizando los dos Enlaces definidos previamente.

```
lsp lsp1 primerNodo [enlUno enlDos] 50
lsp lsp2 primerNodo [enlUno] 50
lsp lsp3 segundoNodo [enlDos] 50
```

Finalmente solo resta asociar un tráfico a cada LSP. El cómo hacerlo dependerá del tipo de Flujo considerado y de la forma en que se construirá el mismo.

En el caso en que el flujo sea de tipo general y se construya a partir de una traza de tráfico, la sentencia de definición será como la que sigue

```
flujo traza1.tr lsp1
```

donde `traza1.tr` es el archivo donde se guarda la traza de tráfico extraída de las mediciones al ingreso a la red y `lsp1` es el LSP al cual se asociará el flujo que se está creando en ésta sentencia. Vale la pena destacar que para el correcto funcionamiento del programa, el archivo que contiene la traza de tráfico, se debe encontrar en el mismo directorio que el script de comandos que define la red.

Si el Flujo a asociar a lsp1, fuera de tipo Exponencial ON-OFF, pero de igual manera se creara a partir de una traza de tráfico, la sentencia solo cambiaría la palabra de comienzo de la manera que sigue:

```
flujoONOFF traza2.tr lsp2
```

Como ya se adelantó, existe también la posibilidad de crear un flujo Exponencial ON-OFF a partir de parámetros conocidos. Para ello solo es necesario sustituir la ubicación de la traza por los parámetros en el orden que se describe con el siguiente ejemplo.

```
flujoONOFF 0 5 0.1 0.1 lsp3
```

Siendo los parámetros las tasas de llegada de tráfico en los estados OFF y ON respectivamente, los inversos de los tiempos medios de permanencia en cada estado, y por último el nombre del lsp.

Mediante la secuencia de comandos hasta ahora presentada a modo de ejemplo, se construyó la red de la figura 9.2, sobre la cual se continuará trabajando en el resto de la descripción de las funcionalidades del programa.

Una vez definida completamente la red, se puede pasar a la próxima etapa de estimar la performance de la misma. Para esto, se permite estimar la probabilidad de pérdida en cualquier enlace de la red mediante una sentencia que comience con la palabra `resolver` y a continuación indique el nombre del enlace, seguido del nodo del enlace que interesa analizar. Por ejemplo, para estimar la probabilidad de overflow del enlace `enlDos`, en el nodo `segundoNodo`.

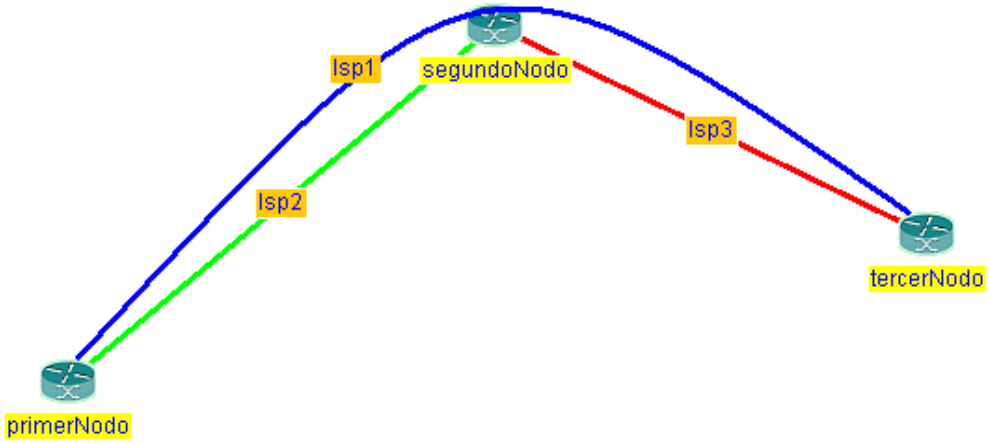

Figura 9.2: Topología del ejemplo construido.

```
resolver Enlace enlDos segundoNodo
```

También el programa permite estimar la probabilidad de pérdida en todos los enlaces de un LSP, mediante una sentencia similar a la anterior, que comienza con `resolver` y a lo que le sigue el nombre del LSP deseado.

```
resolver Lsp lsp1
```

9.2.2. Formato del archivo de salida

Se pasará a describir ahora la forma en que se presentan los resultados obtenidos y cómo el usuario podrá acceder a ellos.

El archivo de texto en el cual se almacenarán los resultados tendrá el mismo nombre que el script de entrada, con la única diferencia que la extensión de este archivo será .out, para identificarlo claramente como un archivo de salida. En él se pueden encontrar tanto los resultados obtenidos, si la resolución resultó satisfactoria, así como los errores en el caso que por algún motivo la simulación no se pudo llevar a cabo.

En el caso que la estimación se haya completado satisfactoriamente el reporte tendrá un formato como el que se presenta a continuación para la resolución del enlace `enlDos` para el nodo `segundoNodo` de la red construida a modo de ejemplo anteriormente.

```
%%%%%%%%%%    RESULTADOS    %%%%%%%%%%

Archivo de entrada: tresnodos.in
Archivo de salida: tresnodos.out

......  1. Comando: resolver Enlace enlDos segundoNodo  ......

    Enlace: enlDos Nodo: segundoNodo
        I = 0.01518686053602425
        xAst = 2.986537291636384  2.4963953899120255  3.0134635359669355
```

```
    Coeficiente de correccion = 2.2888145883886346
    Probabilidad de Overflow = 0.01915267874480316
```

Como primera información, se presenta el nombre del archivo que generó la estimación y el nombre del archivo de salida, esta información aparece independientemente después de los cálculos requeridos, ya sean resoluciones de Enlaces, LSPs o ambos. A continuación se encuentra con un índice y entre puntos suspensivos, el comando del script del cual se detallan, inmediatamente más abajo, los resultados obtenidos, entre los que se encuentran: el valor mínimo de la función de velocidad `I` (tasa de pérdida), el punto donde se alcanza dicho mínimo (`xAst`), el coeficiente de corrección (`Coeficiente de correccion`) y la probabilidad de desborde para la cantidad de fuentes especificada (`Probabilidad de overflow`).

Si lo que se resuelve es un LSP, el archivo de salida presentará un formato de estructura similar a la anterior y la cual se puede apreciar en el siguiente ejemplo.

```
......  2. Comando: resolver Lsp lsp1  ......

    Enlace: enlUno Nodo: primerNodo
        I = 0.21203033361554996
        xAst = 3.9955026103342575   4.004497718608113
        Coeficiente de correccion = 0.6876213408935257
        Probabilidad de Overflow = 4.2558759273907684E-11

    Enlace: enlDos Nodo: segundoNodo
        I = 0.01518686053602425
        xAst = 2.986537291636384  2.4963953899120255  3.0134635359669355
        Coeficiente de correccion = 2.2888145883886346
        Probabilidad de Overflow = 0.01915267874480316

    I del lsp lsp1: 0.01518686053602425
```

Nuevamente lo primero en aparecer entre puntos suspensivos es el comando que generó la serie de resultados que se detallan más abajo. Luego aparecerán de forma ordenada los resultados de todos los enlaces que conforman el LSP. Para una fácil comprensión de los resultados, se indica a qué enlace pertenece cada grupo de resultados obtenidos. Como item final del reporte de los resultados obtenidos para el LSP, se identificará la Tasa de Pérdida Equivalente del LSP, que es la correspondiente al enlace más comprometido.

Si por alguna razón el procesamiento no se pudo completar, el archivo de salida presentará un mensaje de error, conteniendo información relevante sobre los motivos que llevaron a la interrupción del procesamiento. Estas anomalías, son controladas mediante excepciones que evitan la progresión innecesaria de los cálculos y que aparte guardan información sobre el problema detectado, que posteriormente se utilizará para clasificar el mismo y ayudar al usuario en su resolución.

9.3. Interfaz gráfica de usuario

Como ya se explicó en la introducción del capítulo, la interfaz gráfica, no pretende más que simplificar, en la medida de lo posible, el trabajo del usuario a la hora de

configurar la red y acceder a los resultados.

Con esta interfaz, se pretende evitar que el usuario se vea en la tediosa necesidad de escribir el archivo de comandos cada vez que necesita realizar cálculos sobre una red.

En la sección 9.3.1 de este capítulo se presenta el package *InterfazGráfica* que agrupa aquellas clases encargadas de implementar la interfaz gráfica. Finalmente en la sección 9.3.2 se brinda una introducción al uso de dicha interfaz, reproduciendo el ejemplo construido en la sección 9.2.

9.3.1. El package InterfazGráfica

Con este package, se cierra todo todo lo relativo a la herramienta de software desarrollada para este proyecto. Este permite reproducir la estructura de la red, mediante entidades gráficas, para luego volcar esa información a la elaboración automática del archivo de texto correspondiente a la red gráfica. A continuación se presenta el diagrama de clases UML

Las clases *NodoGr*, *EnlaceGr* y *LspGr* son las antes mencionadas entidades gráficas, (nodo gráfico, enlace gráfico y lsp gráfico) que emulan a sus pares del package *Topología*. Dichas entidades tienen la capacidad de almacenar los mismos parámetros que las clases a las que representan, pero sin tener la funcionalidad de las últimas, ya que que no disponen de sus métodos pues no se utilizarán más que como contenedores de información, para la realización del archivo de texto.

Estas entidades son las que realmente crea el usuario mediante la interfaz gráfica. Estas son administradas por la clase *VentanaPrincipal*, quien se encarga de su construcción, almacenamiento, modificación y eventualmente su borrado, hasta que la red (gráfica) queda como definitiva[1] cuando el usuario solicita algún cálculo. En este momento la clase *VentanaPrincipal*, invoca al método **traductor** quien se encarga de generar el archivo de texto que represente la red gráfica creada por el usuario y que luego interpretará las clases del package *interfazSistema*. Luego de procesado el archivo de texto, el programa despliega el informe con los resultados obtenidos, con lo que concluye el ciclo de interacción entre software y usuario. Para visualizar este ciclo se presenta a continuación en la figura 9.4 el diagrama de colaboración UML del sistema completo.

[1]Se habla de definitiva ya que el usuario no podrá efectuar más cambios, hasta que el programa termine su procesamiento

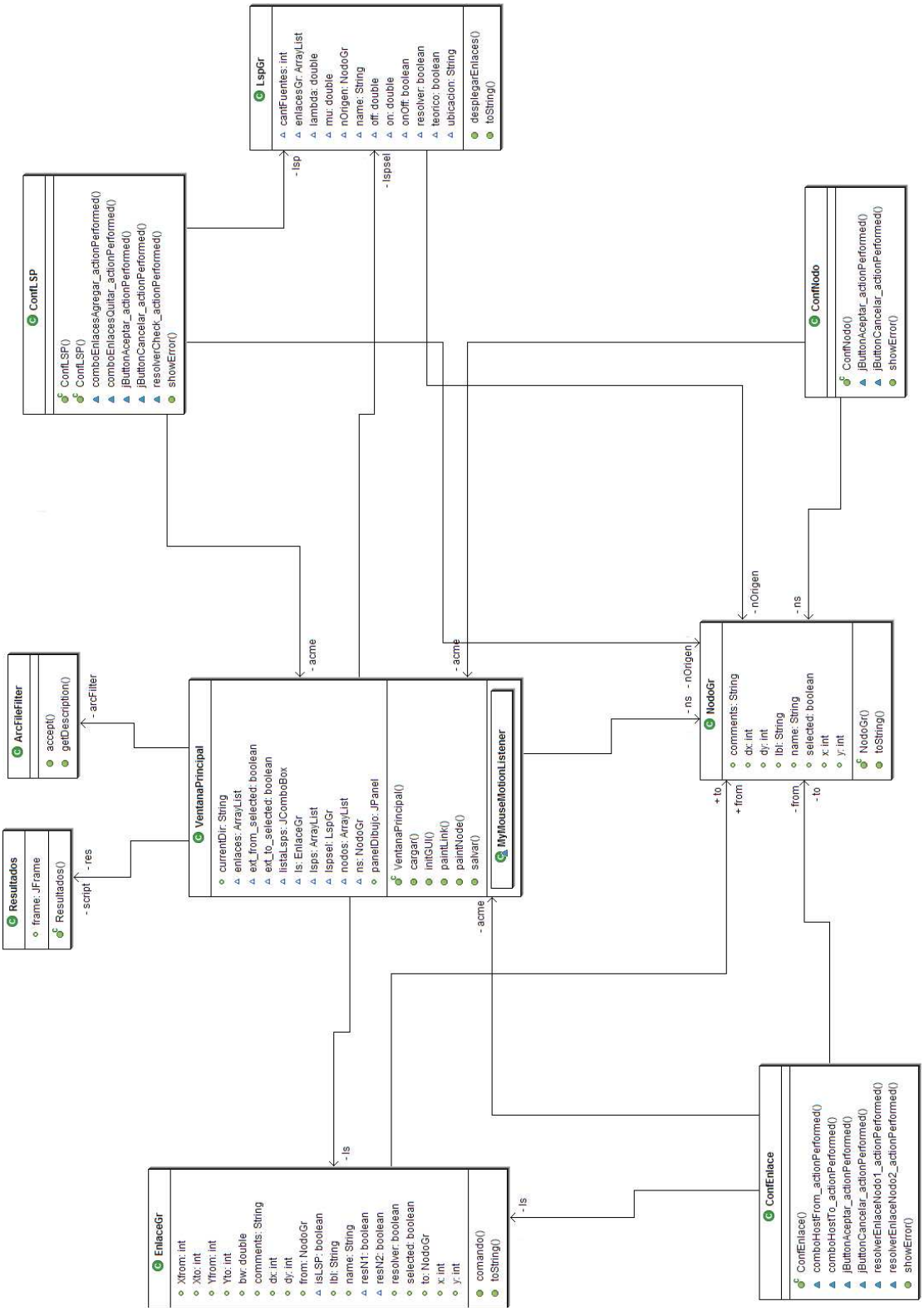

Figura 9.3: Diagrama de clases UML de la interfaz gráfica

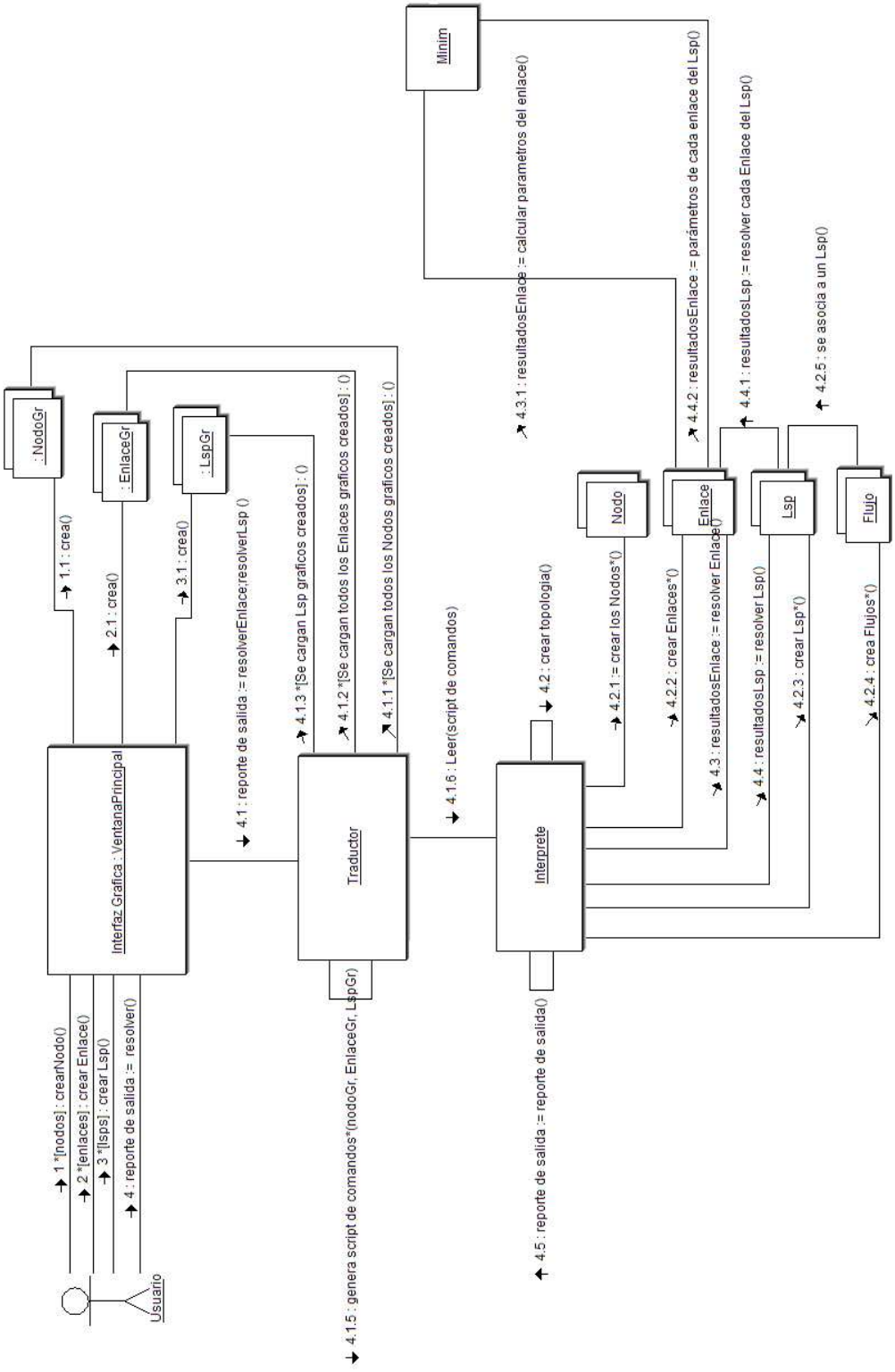

Figura 9.4: Diagrama de colaboración UML del sistema completo

9.3.2. Modo de uso de la interfaz gráfica

A continuación se reconstruirá la topología de tres nodos ya vista en la sección 9.2, con el fin de presentar la interfaz gráfica y también de demostrar la simplicidad de su

uso.

Como en el archivo de comandos, el primer paso es generar los nodos. Para ello, solo es necesario hacer click en el botón *Crear Nodo* del panel *Topología*. En nuestro caso creamos tres nodos los cuales llamamos de igual manera que el ejemplo antes visto. El resultado se aprecia en la figura 9.5.

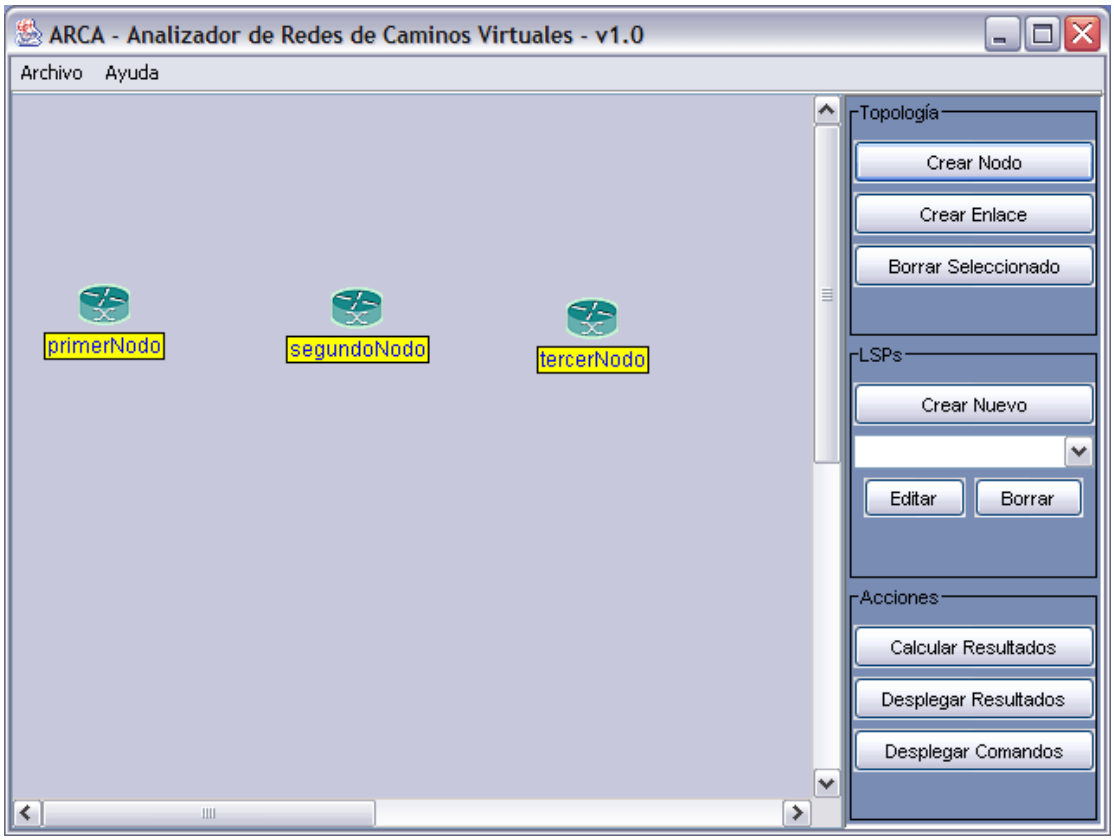

Figura 9.5: Construcción de los tres nodos

Con cada click, el programa crea por defecto una instancia de *NodoGr* con un nombre predefinido. En caso de que el usuario desee cambiar dicho nombre, se debe hacer doble click sobre el nodo en cuestión y la siguiente ventana aparecerá.

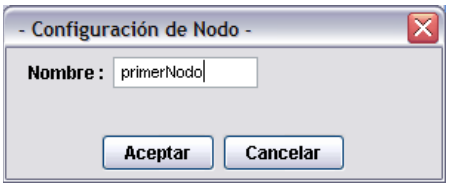

Figura 9.6: Ventana de configuración de un Nodo

Una vez generados los nodos se procede a crear los enlaces, que en nuestro caso serán dos. Para esto se debe hacer click en el botón *Crear Enlace* del panel *Topología*. Esto genera instancias de la clase *EnlaceGr* las cuales tienen por defecto un nombre prefijado, su capacidad en 0 y no tienen nodos asociados. Para asociarle un nodo a un nuevo enlace, solo se requiere arrastrar con el mouse el extremo deseado del mismo hasta el nodo en cuestión. Si creamos dos enlaces y los asociamos a los nodos existentes, se genera la topología deseada en la figura 9.7.

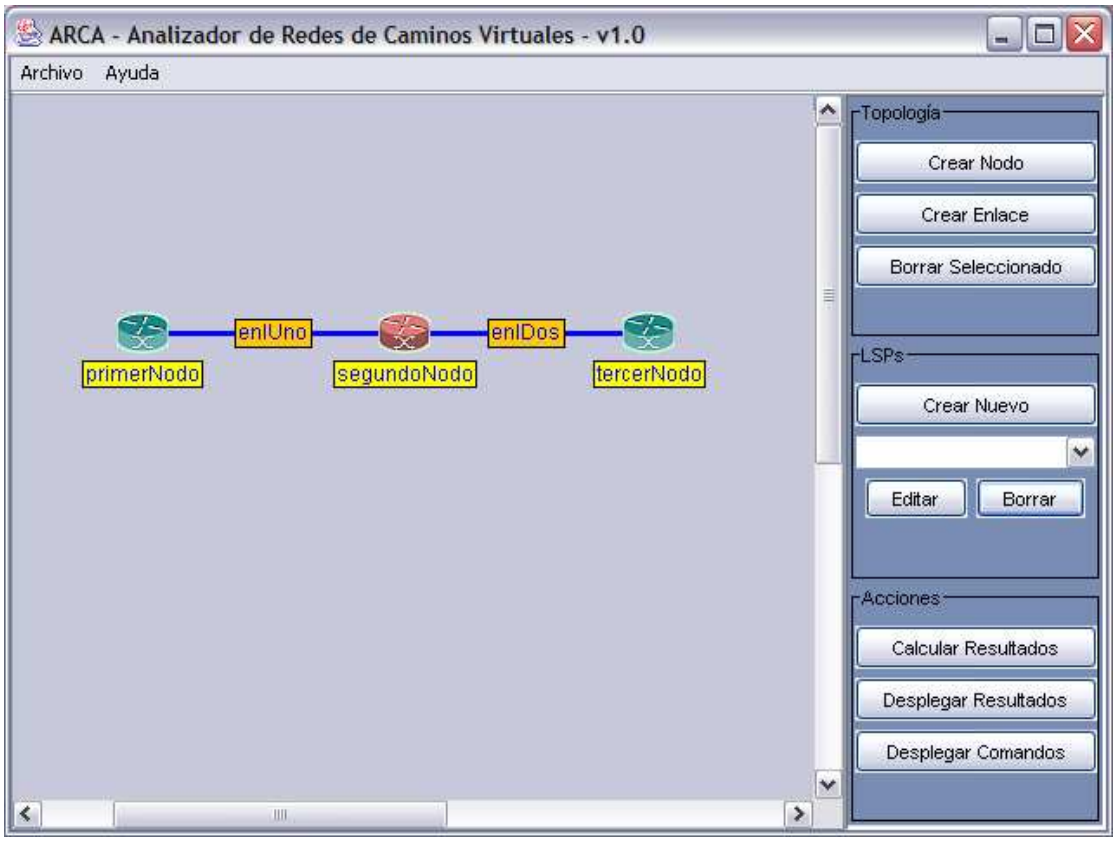

Figura 9.7: Topología completa

Como la capacidad por defecto de los enlaces es cero, si intentáramos realizar algún cálculo, el programa retornaría error por la segura presencia de enlaces inestables. Para modificar este y otros parámetros de un enlace se debe hacer doble click sobre el enlace en cuestión, a lo que el sistema responde mostrando la siguiente pantalla.

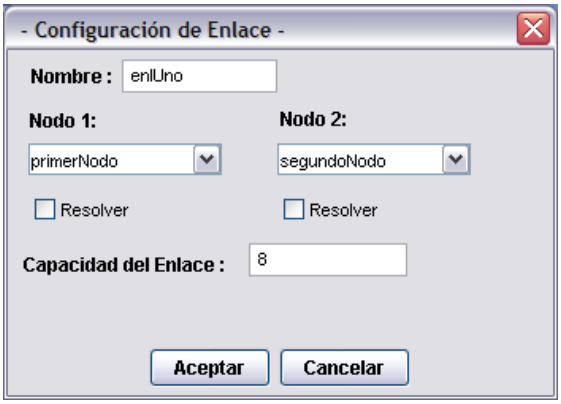

Figura 9.8: Pantalla de configuración de enlaces

La anterior pantalla permite no solo modificar el nombre y la capacidad del enlace, sino que también se identifican los nombres de los nodos asociados, si los hay, y se deja la opción de resolver el enlace para cualquiera de los dos nodos.

Luego de configurada la topología de la red, se deben definir los Lsp. Para esto se debe hacer click en el botón *Crear Lsp* del panel *Lsp* lo que activa la siguiente pantalla.

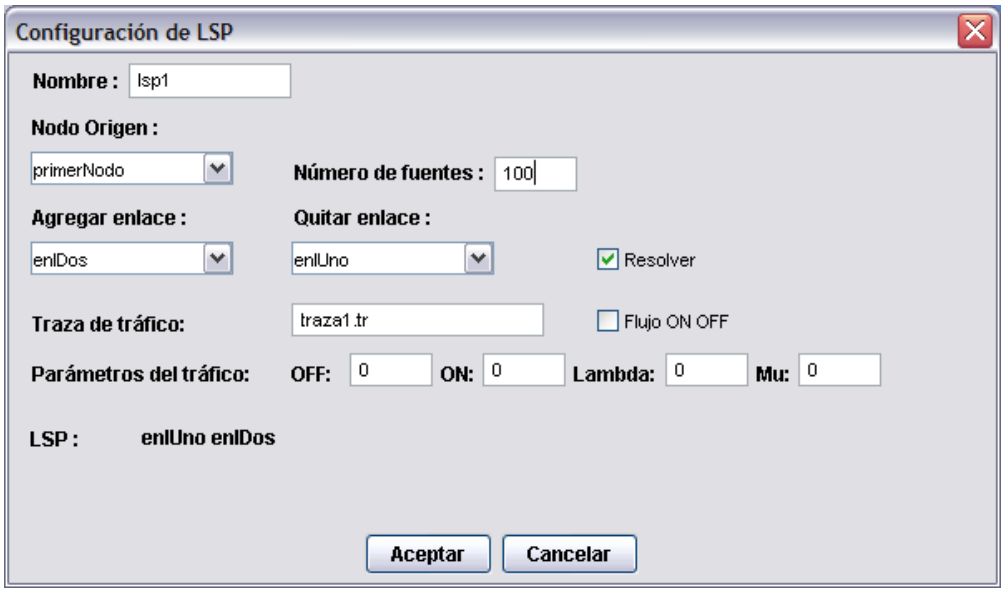

Figura 9.9: Pantalla de configuración de los Lsp

Aquí se puede cambiar el nombre por defecto del Lsp, se elige el de entre todos los nodos creados, aquel que se considerará como el de origen para el Lsp. Luego de entre los enlaces existentes se seleccionan aquellos enlaces que conforman el Lsp, los cuales van apareciendo en el campo *LSP*. La pantalla de configuración de los Lsp también cuenta un campo que permite introducir el nombre del archivo de la traza a partir de la cual se estimarán los parámetros del Flujo que se asociará automáticamente al Lsp. Se puede elegir también si este flujo debe ser tratado como exponencial ON-OFF o no. Para el caso del Flujo ON-OFF también se brinda la posibilidad de definirlo indicando sus parámetros.

Finalmente si se quiere resolver este lsp, entendiéndose por esto, resolver todos sus enlaces esto se debe indicar marcando la correspondiente casilla de verificación (check-box).

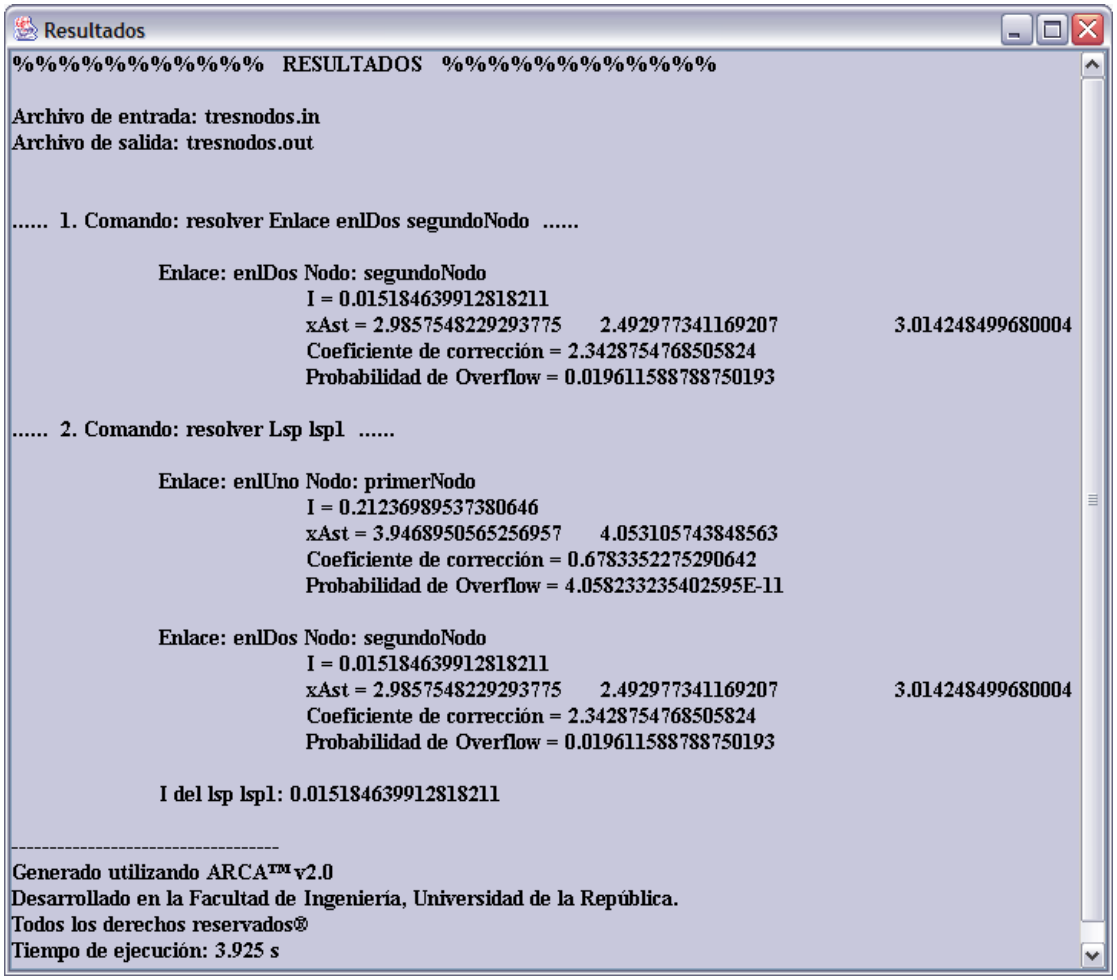

Figura 9.10: Reporte de resultados

Luego de definir los tres Lsp como en el ejemplo de la red antes construida, solo resta pedir que se realicen los cálculos. Para esto se debe hacer click en el *Calcular Resultados* del panel *Acciones*. Si la configuración no fue guardada[2] el programa solicitará hacerlo. Si la configuración ya estaba guardada, el programa realizará los cálculos y mostrará el reporte de los resultados. Para el caso de la red configurada, el reporte se puede apreciar en la figura 9.10.

[2]Como un archivo de extensión .arc

Capítulo 10

Manejo de errores

Toda aplicación que se relaciona con agentes externos, es propensa a que de dicha interacción se generen errores de distinta índole. Nuestro programa no es la excepción, ya que de la interacción con el usuario pueden darse situaciones que impidan el correcto funcionamiento del software. Es por esto que la totalidad de dichas situaciones deben ser contempladas, a los efectos de impedir la realización de cálculos sobre bases erróneas. Este tipo de situaciones problemáticas fueron tratadas mediante el *manejo de excepciones* que proporciona JAVA. Esta técnica, también presente en otros lenguajes Orientados a Objetos [1], permite dar una finalización controlada a situaciones no previstas por el programador y almacenar información de las mismas que ayude a su solución.

10.1. El package excepciones

Este package, contiene las excepciones que sirven de contingencia para aquellos códigos que por distintos motivos, no es posible llegar a ejecutar de forma adecuada. Desde el punto de la implementación, todas las excepciones creadas *heredan* de una excepción general, que nos permite capturar de forma sencilla todas las excepciones por igual, y recuperar la información que contienen. Esto se puede apreciar en el diagrama UML de clases de la figura 10.1

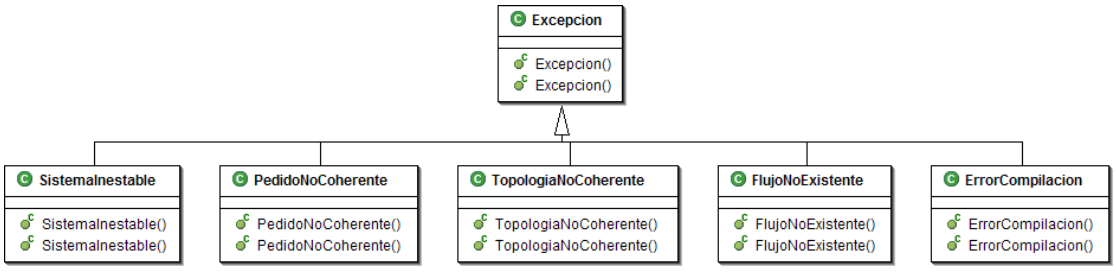

Figura 10.1: Diagrama de clases del package Excepciones

A continuación se enumeran las excepciones que fueron creadas, junto a una breve descripción del tipo de problemas al cual están orientadas, así como la información que almacenan.

[1] por ejemplo C++

10.2. Excepciones que detectan Errores en la Definición de la Red

10.2.1. Excepción ErrorCompilación

Cuando el archivo de comandos contiene una línea que no comienza con ninguna palabra reservada, o no sigue el formato especificado del lenguaje, o bien no está comentada[2], se interrumpe la ejecución del programa mediante una excepción del tipo ErrorCompilación. Esta excepción guarda e imprime en el archivo se salida, información referente a la línea del archivo leído en la cual se detectó el comando no válido.

10.2.2. Excepción TopologíaNoCoherente

Este tipo de excepción, interrumpe la ejecución del programa, cuando la definición de la red proporcionada por el usuario, presenta fallas graves, que impiden una correcta definición de la misma y por ende impiden la realización de cualquier tipo de cálculo. La excepción TopologíaNoCoherente previene entre otras las siguientes situaciones:

- Crear un *Enlace* con uno o ambos Nodos que no hayan sido definidos aún.
- Crear un *Lsp* con uno o varios Enlaces que no hayan sido definidos aún.
- Crear un *Lsp* cuyo *Nodo* de origen no haya sido definido aún.
- Crear un *Lsp* en el cual sus enlaces no tengan Nodos en común.
- Asociar un Flujo a un *Lsp* no definido.

En todos los casos anteriormente descritos, esta excepción imprime un mensaje de error en el archivo de salida, indicando cual fue la falla detectada y la línea donde se encontró.

10.2.3. Excepción FlujoNoExistente

Previo a la realización de cálculo alguno, el sistema verifica que todos los *Lsp*, tengan un Flujo asociado. Dada la posibilidad de que sean varios los *Lsp* que no tengan Flujo asociado y con el objetivo de acelerar la solución de este problema, el programa no interrumpe su ejecución con cada *Lsp* que no tenga Flujo asociado. Por el contrario, de encontarse un *Lsp* sin Flujo asociado, se guarda información de éste, que permita en una posterior instancia identificarlo y solucionar el problema, sin interrumpir la ejecución de modo que todos los *Lsp* pasen por esta revisión. Una vez analizados todos los *Lsp* y en caso de haber encontrado alguno sin flujo asociado, la excepción FlujoNoExistente, interrumpe la ejecución del programa desplegando en el archivo de salida, un mensaje de error. Dicho mensaje revela los nombres de aquellos *Lsp* a los cuales hay que necesariamente asociar un Flujo, para poder realizar cálculos y la línea del código en que se encuentran.

10.2.4. Excepción PedidoNoCoherente

Una vez asegurada la correcta definición de la red, puede que el cálculo solicitado sobre la misma no tenga sentido, es aquí donde actúa esta excepción, impidiendo situaciones como las que siguen:

[2]Para comentar una línea se debe iniciar la misma con el símbolo de porcentaje(%)

- Pretender resolver un *Enlace* con un *Nodo* que no pertenece a él.
- Pretender resolver un *Lsp* no definido.

Este tipo de excepciones alerta al usuario acerca de la imposibilidad de resolver su pedido con un mensaje de error en el archivo de salida.

10.3. Excepciones que detectan Inestabilidades en la Red

10.3.1. Excepción SistemaInestable

Una vez que la red está correctamente definida, es decir que pasó satisfactoriamente los chequeos descritos en el grupo anterior de excepciones, puede aun darse el caso que la misma no este dentro de nuestras hipótesis de trabajo. Como se discutió en la sección 3.4 es condición necesaria para la implementación de nuestras técnicas, que la suma de las medias de los tráficos que pasan a través de un *Enlace* sea menor que la capacidad de éste, ya que de lo contrario, el sistema se vuelve inestable, lo que se traduce en que la probabilidad de desborde es 1. En orden de evitar lo anterior previo a la realización de los cálculos solicitados por el usuario el sistema chequea la estabilidad de cada uno de los enlaces de la red. La excepción SistemaInestable guarda e imprime información de aquellos Enlaces que no verifican la condición de estabilidad anteriormente descrita. Así como en la excepción de la sección 10.2.3 puede que sean más de uno los Enlaces comprometidos, por lo cual, por un tema de eficiencia, conviene detectarlos en una misma instancia, para lo que es necesario que el programa siga corriendo. Pero aquí también es imperioso detener la ejecución ante la presencia de al menos un *Enlace* inestable, por lo cual se utiliza un procedimiento similar al de la sección citada anteriormente para resolver este problema. Esto es, analizar todos los enlaces, a medida que se detectan enlaces inestables, guardar información sobre estos y una vez finalizado este proceso, imprimir en el reporte de salida dicha información.

Parte IV

Validación y Conclusiones

Capítulo 11

Validación de los resultados

En este capítulo pondremos a prueba los resultados teóricos desarrollados a lo largo del proyecto. Para esto contrastaremos las estimaciones de la probabilidad de overflow dadas por nuestra aplicación, contra resultados obtenidos mediante simulaciones. Para probar la aplicabilidad de nuestra herramienta, se seleccionaron algunos casos representativos de redes, donde puede ser de utilidad determinar la probabilidad de overflow, con el objetivo de poder conocer las garantías de QoS.

Debido a las dificultades existentes para realizar mediciones sobre una red en funcionamiento, que nos permitan verificar nuestros resultados se optó por utilizar la herramienta de simulación NS-2, de extensa aceptación en la comunidad internacional.

Es así que en la primera sección se discuten las generalidades del proceso de validación, para luego pasar a la sección 11.2, donde se presentan las simulaciones y estimaciones para un enlace al cual convergen numerosas fuentes. Luego en la sección 11.3 se pasa a simular y contrastar contra nuestras predicciones, el ejemplo descrito en profundidad en la sección 4.3. Posteriormente en la sección 11.4 se extiende el caso de dos enlaces a un Lsp de mayor cantidad de saltos. Para finalizar, en la sección 11.5 se simulan una red con topología en forma de árbol, estudiando el enlace de la raíz.

11.1. Criterios generales para la validación

A los efectos de validar las estimaciones realizadas mediante la técnica desarrollada a lo largo del proyecto, se compara la probabilidad de overflow estimada con la obtenida mediante simulaciones variando el número de fuentes que ingresan a la red. De la sección3.2, se tiene que el comportamiento asintótico de la probabilidad de overflow verifica la siguiente ecuación:

$$\lim_{N\to\infty} \frac{1}{N} \log \mathbf{P}(\mathit{overflow}) = -\mathbf{I}$$

siendo N el número de fuentes que afectan al enlace.

Dicho comportamiento se debe ver reflejado en las simulaciones por lo que se graficó la magnitud correspondiente en cada uno de los casos. Además la ecuación anterior sugiere el uso de una escala logarítmica para comparar la probabilidad de overflow estimada y experimental ya que el logaritmo de dicha magnitud sigue un comportamiento aproximadamente lineal en relación al número de fuentes.

De lo anterior aparece la necesidad de estimar a partir de las simulaciones en NS la probabilidad de overflow en los diferentes enlaces de la red en estudio. Para ello se monitorearon los buffers correspondientes y se estimó la probabilidad de overflow como:

$$P_{\exp}(\mathit{overflow}) = \frac{\text{cant. de unidades de tiempo donde se registran pérdidas}}{\text{tiempo total de simulacion en unidades de tiempo}}$$

Debido a la hipótesis de ergodicidad de los procesos de llegada de trabajo lo anterior constituye un estimador de la verdadera probabilidad de overflow. Para el relevamiento del estado de los buffers se trabajó en tiempo discreto con la misma tasa de muestreo que las trazas de tráfico que el mismo simulador genera y alimenta a nuestra aplicación. Se simuló para diferentes tasas no encontrándose grandes diferencias en los resultados. La tasa de muestreo elegida para las simulaciones que se detallan a continuación es de una muestra cada 5ms.

En todos los casos se simulaban las redes elegidas durante un tiempo suficientemente largo según el tipo de red, que permita asegurar la convergencia del anterior estimador. Lo anterior trae aparejado un compromiso entre el tiempo de simulación necesario para obtener buenas estimaciones y la viabilidad de llegar a cabo dicha simulación ya que los tiempos requeridos por el NS-2 en el caso de redes de gran porte se vuelven inmanejables. La necesidad de los prolongados tiempos de simulación está dada por la baja frecuencia con que se pretende que ocurran estos eventos en las redes estudiadas.

Por un compromiso similar al anterior el número de fuentes generadoras de tráfico no puede hacerse arbitrariamente grande. Se observó que utilizando computadoras personales de última generación el límite práctico para el número de fuentes que generan tráfico simultáneamente ronda el millar.

Para reducir los efectos de la varianza de cada simulación, cada topología fue simulada en 5 ocasiones, y el promedio de las probabilidades de pérdida obtenidas fue el valor considerado real.

Para modelar el tráfico, se eligió simular en todos los casos con flujos markovianos ON-OFF de distintos ciclos de trabajo y niveles de tasas de arribo. En general se obtuvieron comportamientos similares en cada caso, por lo que para cada simulación se eligió un caso representativo.

Para medir la precisión de las estimaciones se eligió como magnitud de comparación la distancia en escala logarítmica dada por

$$E = \log_{10}\{\mathbf{P}_{estimada}(overflow)\} - \log_{10}\{\mathbf{P}_{experimental}(overflow)\} \tag{11.1}$$

Esta magnitud resulta adecuada para medir la precisión en el orden de la estimación. Muchas veces resulta de interés estimar el orden de la probabilidad de overflow y no su valor exacto. En términos del error elegido, una unidad representa un orden de magnitud de diferencia.

11.2. Evidencia en el caso de un único enlace

Para comenzar con el proceso de validación de los resultados, se optó por probar la precisión de la técnica en la topología de red más sencilla posible. Esto es un único buffer al cual arriban numerosas fuentes. Aparte este caso es de especial interés, ya que como se presentó en la sección 3.4.3 para el mismo existe un factor de corrección que permite estimar el ratio de pérdida de paquetes a partir de la probabilidad de overflow del buffer. Esta magnitud es de especial interés desde el punto de vista del usuario de la red, ya que le permite determinar que tipo de servicios puede utilizar sobre la misma. Esto se debe a que a partir de la probabilidad de desborde del buffer el usuario no puede obtener información sobre lo que le ocurrirá a sus paquetes.

A continuación, en la gráfica 11.1 se presenta la comparación de nuestra estimación de la probabilidad de overflow con la obtenida mediante la simulación. En este caso en particular también se guardó registro de la cantidad de paquetes perdidos del total de enviados en la simulación, con lo que se estimó el ratio de pérdida de paquetes de acuerdo con la siguiente ecuación:

$$\text{Tasa de pérdida} = \frac{\text{cant. de paquetes perdidos}}{\text{número total de paquetes enviados}}$$

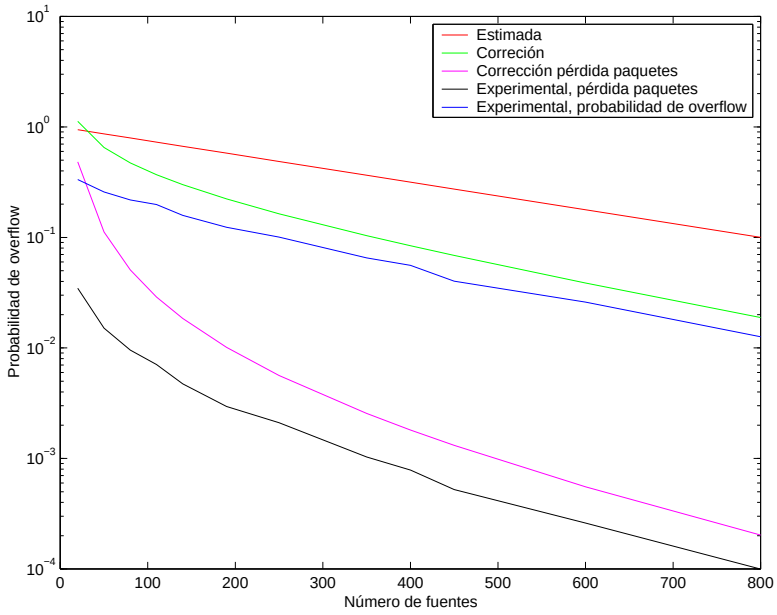

Figura 11.1: Probabilidad de overflow para el caso de un único enlace

En la siguiente figura se muestran los resultados obtenidos en la comparación para el caso de la tasa de pérdida (I).

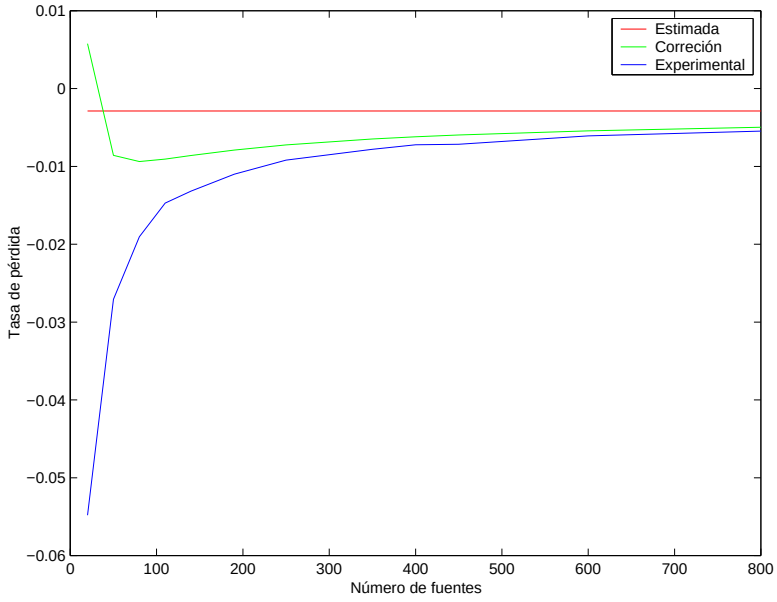

Figura 11.2: Tasa de pérdida para el caso de un único enlace

Como se observa en la figura 11.1 nuestra técnica sobreestima levemente la probabilidad de overflow, error que se aprecia con mayor exactitud en la figura 11.3. De todos modos se observa que dicho error siempre permanece muy por debajo de un orden de magnitud. En términos de la tasa de pérdida (figura 11.2), este error se desvanece al aumentar el número de fuentes, debido a la normalización utilizada.

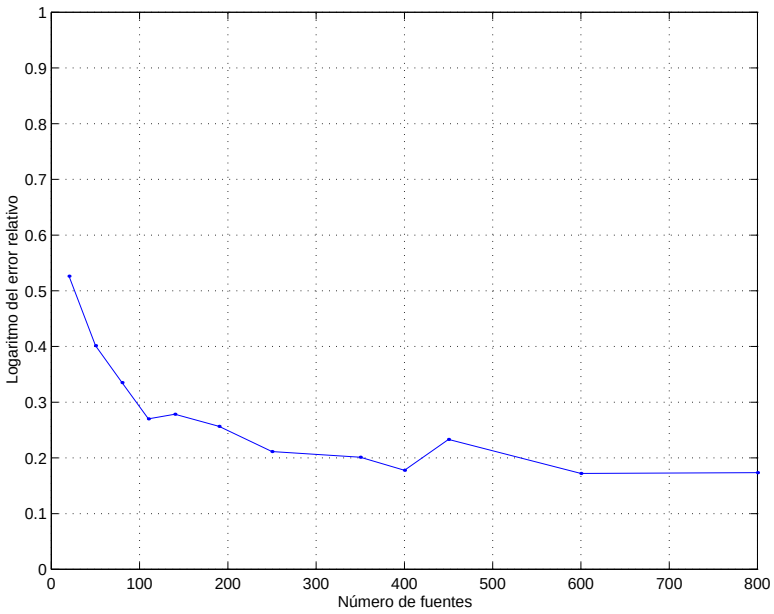

Figura 11.3: Error logarítmico para el caso de un único enlace

El marco en que se realizó la anterior simulación se describe a continuación

- **Tipo de Flujo:** Se utilizó tráfico del tipo Markoviano ON-OFF, con tiempo medio de permanencia en el estado ON de 40ms y 20ms para el estado OFF, y tasa pico de generación de trabajo de 5 paquetes por unidad de tiempo.
- **Capacidades de los enlaces:** La capacidad por fuente del enlace era 7 paquetes por unidad de tiempo.
- **Tamaño del buffer:** Se utilizó un tamaño de buffer capaz de almacenar hasta $2\sqrt{N}$ paquetes cuando se superponen N fuentes.
- **Tiempo de simulación:** La simulación se repitió 5 veces, con el objetivo de disminuir la varianza de la estimación, y el resultado que se presenta es el promedio de todas las corridas. En todos los casos, el tiempo de simulación utilizado era de 250 segundos.
- **Número de Fuentes:** La cantidad de fuentes de tráfico que conformaban el agregado se varió desde 20 hasta 800 tratando de cubrir todas las zonas intermedias. Vale la pena aclarar que en este caso fue posible un número de fuentes tan elevado ya que la simplicidad de la red así lo permitía, buscando lograr la mayor proximidad posible al comportamiento asintótico.

Para analizar la dependencia de los resultados con el tamaño del buffer utilizado, se repitió la simulación correspondiente a la situación anterior variando la tasa de crecimiento del buffer. Se optó por simular tomando los tamaños del buffer de la forma $b\sqrt{N}$ y variando el valor de b.

Según la teoría, si el buffer es despreciable frente a la capacidad, la probabilidad de overflow es insensible al tamaño del mismo. Además, el valor de la misma debería acercarse a las estimaciones, que corresponden al caso sin buffer.

En la figura 11.4 se aprecian los resultados obtenidos.

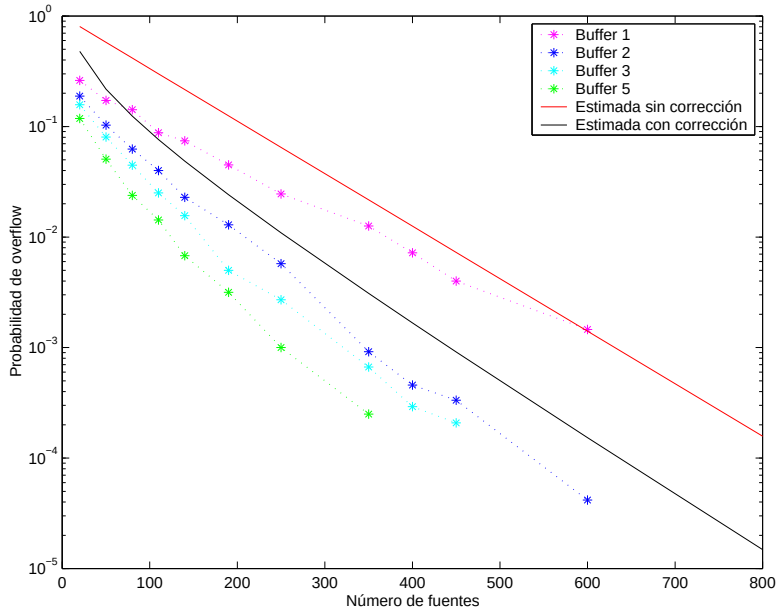

Figura 11.4: Variación de la probabilidad de overflow con el tamaño del buffer

Es de notar que, en concordancia con los resultados teóricos, la pendiente asintótica de las simulaciones corresponde a la tasa de pérdida calculada en la mayoría de los casos. Como es de esperar, el valor de la probabilidad de overflow decrece al aumentar el buffer disponible, debido a que el sistema es capaz de asimilar ráfagas de mejor manera, aumentando su capacidad efectiva. Esto está de acuerdo con lo anunciado en la observación 3.2.

Con el objetivo de realizar simulaciones que se acerquen lo más posible a las hipótesis de trabajo, lo que permitiría contrastar de la manera más adecuada, se intentó simular situaciones en las que el tamaño del buffer se aproximara a cero lo mas posible. Como se puede ver en la gráfica, esto no fue posible ya que nuestro modelo no se ajusta a las condiciones de la herramienta de simulación.

La explicación a esto se encuentra en la utilización de un modelo de tiempo discreto para aproximar el comportamiento de la cola (ecuación de Lindley). El anterior modelo calcula la ocupación del buffer en cada momento, a partir de la cantidad total de arribos de paquetes en un intervalo de tiempo. Esto de por sí introduce una perturbación, ya que no se considera la forma en que se producen dichos arribos. Esto se ve acentuado por el hecho de no considerar que los paquetes se envían de a uno a la vez, y que cada envío insume un determinado tiempo durante el cual el paquete aún permanece en el buffer. Para valores de b pequeños, esto genera problemas, ya que el tamaño del buffer no es capaz de soportar la simultaneidad de arribos, cuya cantidad media crece como la cantidad de fuentes.

En el caso en que se utilizó $b = 1$, se aprecian claramente las limitaciones del modelo antes mencionadas, donde las pérdidas no siguen el comportamiento previsto. En el extremo, es decir, para un valor de $b = 0$, se observa en las simulaciones que la totalidad de los paquetes se pierden, lo que no debería ocurrir según la ecuación de Lindley.

De todas formas, sigue siendo relevante verificar si los resultados son correctos, por lo que se optó por utilizar un simulador específico, que implementa la ecuación de Lindley para esta situación en particular. Los resultados se aprecian el las figuras 11.5 y 11.6 para diferentes ciclos de trabajo del proceso de entrada.

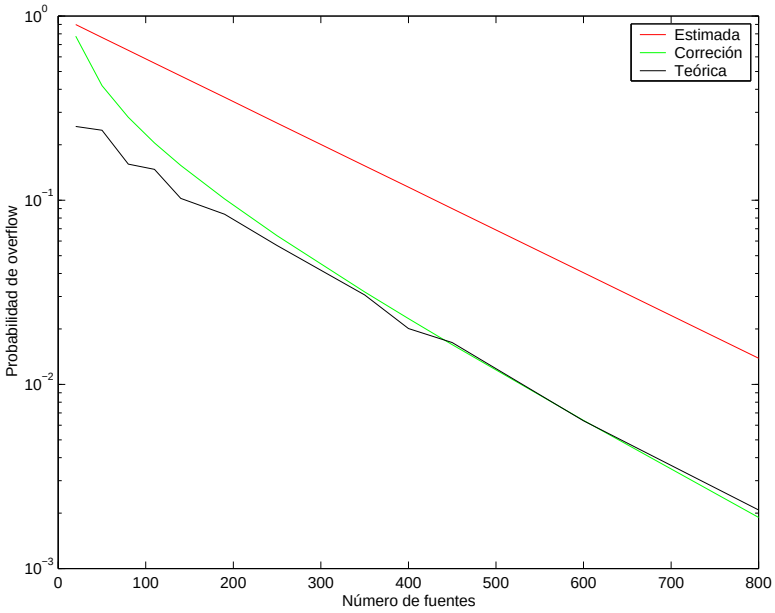

Figura 11.5: Probabilidad de overflow para el caso sin buffer

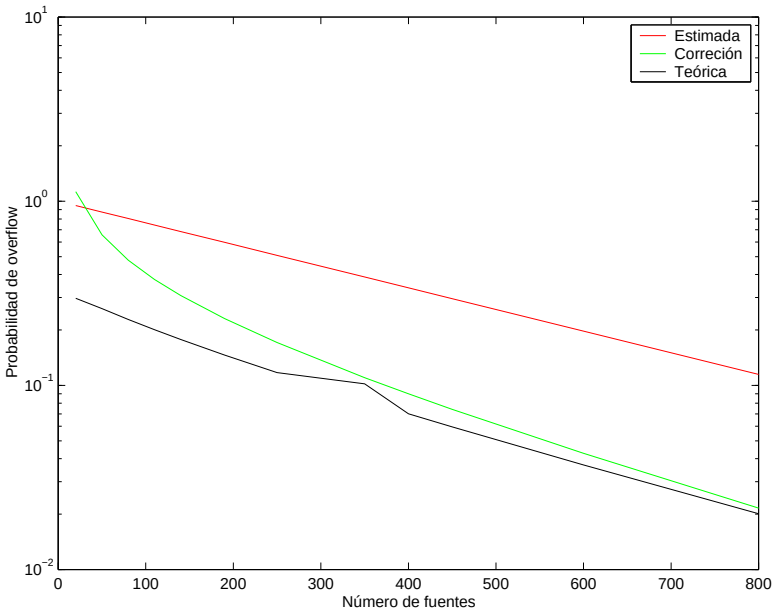

Figura 11.6: Probabilidad de overflow para el caso sin buffer

De lo anterior podemos concluir que los resultados obtenidos están de acuerdo con las estimaciones teóricas, y que el error cometido es atribuible al modelo.

11.3. Evidencia en el caso de dos enlaces

Este ejemplo, que ya fue presentado en la sección 4.3 nos permitirá obtener una primera aproximación cualitativa a las estimaciones que provee nuestra herramienta de software.

A su vez, dado que la resolución de este ejemplo deriva en un problema de pocas variables, nos permite verificar de manera analítica, algunos de los resultados obtenidos y explicar posibles discrepancias con la realidad.

En particular, resultó de gran utilidad para verificar el correcto funcionamiento de los principales algoritmos de nuestro programa, ya que muchos de los resultados obtenidos, pueden ser interpretados intuitivamente.

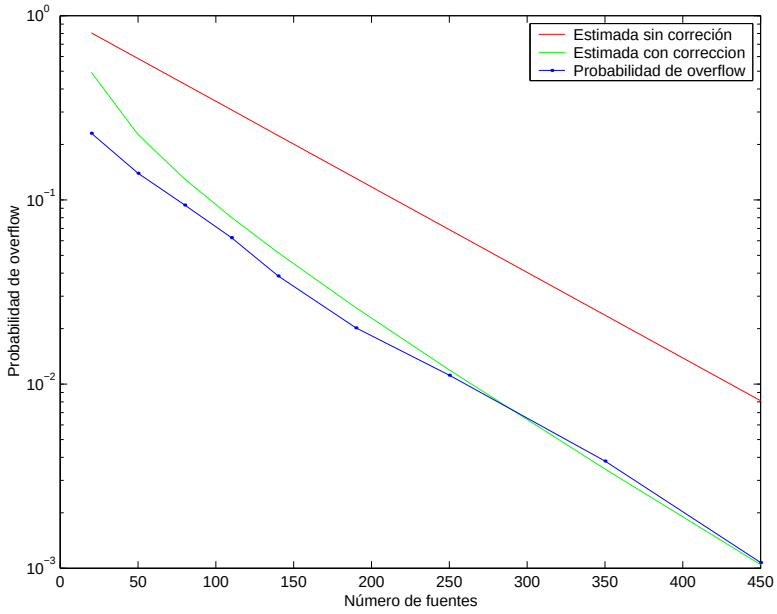

Figura 11.7: Probabilidad de overflow para el caso de 2 enlaces

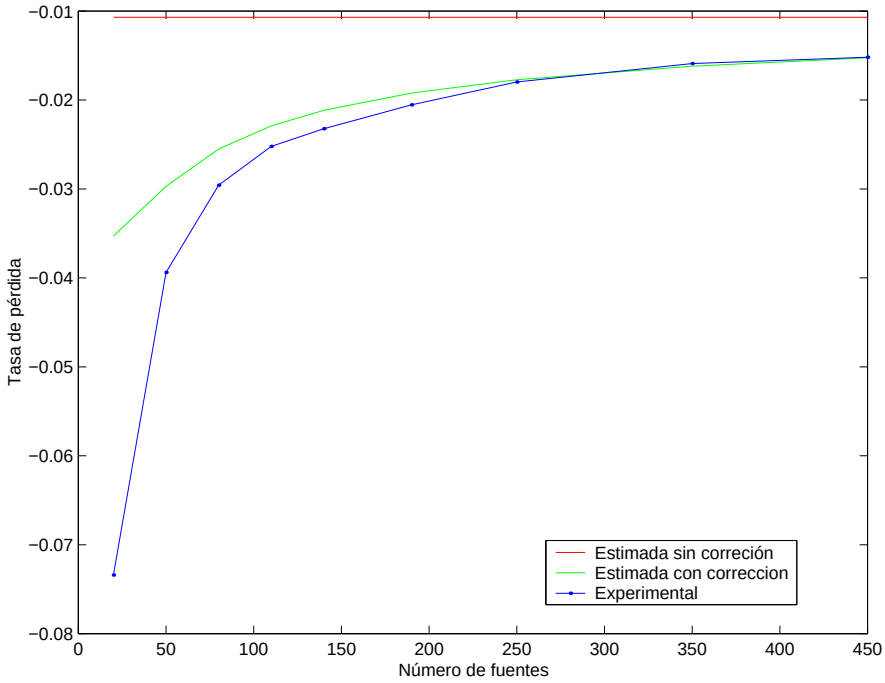

Figura 11.8: Tasa de pérdida para el caso de 2 enlaces

En la figura 11.7 se puede apreciar el resultado obtenido en las simulaciones, en donde se graficó la probabilidad de overflow del segundo enlace contra la cantidad de fuentes. En ella se observa la precisión alcanzada en este caso, precisión que también se aprecia en la figura 11.8 y se ve reflejada en el bajo error obtenido que se observa en la figura 11.9.

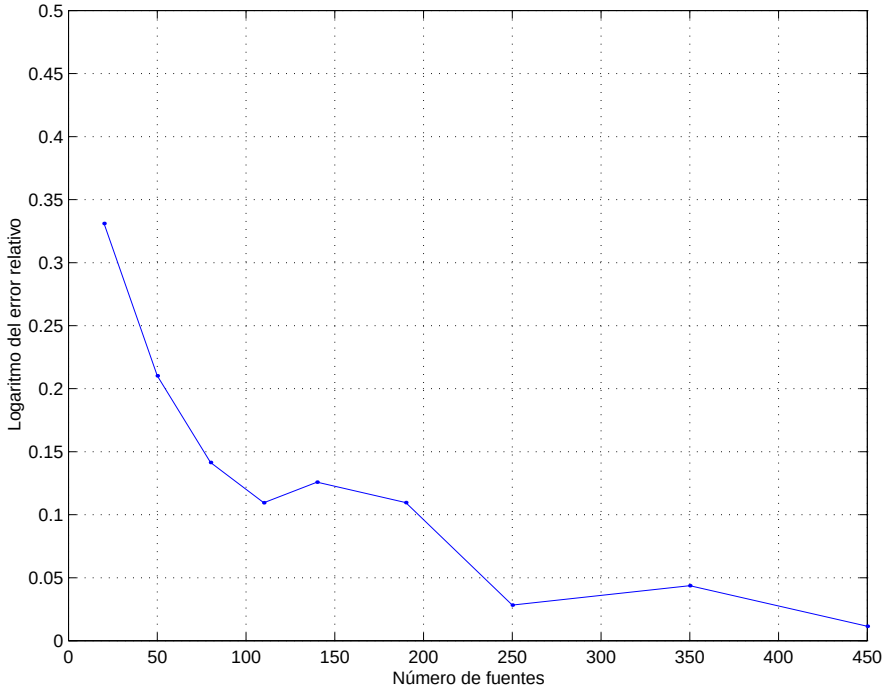

Figura 11.9: Error logarítmico para el caso de 2 enlaces

Las condiciones de la simulación presentada son:

- **Tipo de Flujo:** Se utilizó tráfico del tipo Markoviano ON-OFF, con tiempo medio de permanencia en el estado ON de 50ms y 50ms para el estado OFF, y tasa pico de generación de trabajo de 5 paquetes por unidad de tiempo.
- **Capacidades de los enlaces:** La capacidad por fuente del primer enlace era de 7 paquetes por unidad de tiempo mientras que para el segundo enlace era de 5.5, con una tasa máxima para cada enlace de 10 unidades.
- **Tamaño del buffer:** Se utilizó un tamaño de buffer capaz de almacenar hasta $2\sqrt{N}$ paquetes cuando se superponen N fuentes.
- **Tiempo de simulación:** La simulación se repitió 5 veces, con el objetivo de disminuir la varianza de la estimación, y el resultado que se presenta es el promedio de todas las corridas. En todos los casos, el tiempo de simulación utilizado era de 250 segundos.
- **Número de Fuentes:** La cantidad de fuentes de tráfico que conformaban el agregado se varió desde 20 hasta 450 tratando de cubrir todas las zonas intermedias.

En este caso se puede afirmar que los resultados son mejores, observándose una menor influencia del factor d_0.

11.4. Evidencia en el caso de 4 enlaces

Este ejemplo pretende modelar el caso de un flujo (LSP) que atraviesa una red, pasando por varios saltos, compartiendo recursos con otros muchos flujos en cada uno de ellos. En particular se estudia la probabilidad de pérdida en el último enlace de la red, de modo de analizar la influencia de compartir recursos en reiteradas ocasiones, así como poner a prueba la técnica en un enlace interior a una red.

La red que se simula es la que se observa en la figura 11.10. Sobre esa topología se definieron 5 LSP, uno de los cuales la recorre de punta a punta, mientras que los otros 4 comparten con el anterior cada uno de los enlaces de la red, representando tráfico interferente.

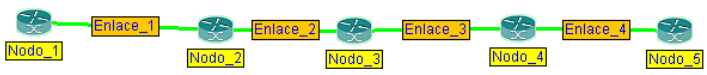

Figura 11.10: Topología de red simulada

Las condiciones en que se realizó la simulación son:

- **Tipo de Flujo:** Se utilizó tráfico del tipo Markoviano ON-OFF, con tiempo medio de permanencia en el estado ON de 50ms y 50ms para el estado OFF, y tasa pico de generación de trabajo de 5 paquetes por unidad de tiempo.
- **Capacidades de los enlaces:** La capacidad por fuente para los 4 enlaces era 7 6.5 6 5.5 paquetes por unidad de tiempo respectivamente, sobre un pico máximo en cada enlace de 10 unidades.

- **Tamaño del buffer:** Se utilizó un tamaño de buffer capaz de almacenar hasta $2\sqrt{N}$ paquetes cuando se superponen N fuentes.
- **Tiempo de simulación:** La simulación se repitió 5 veces, con el objetivo de disminuir la varianza de la estimación, y el resultado que se presenta es el promedio de todas las corridas. En todos los casos, el tiempo de simulación utilizado era de 250 segundos.
- **Número de Fuentes:** La cantidad de fuentes de tráfico que conformaban el agregado se varió desde 20 hasta 350 tratando de cubrir todas las zonas intermedias.

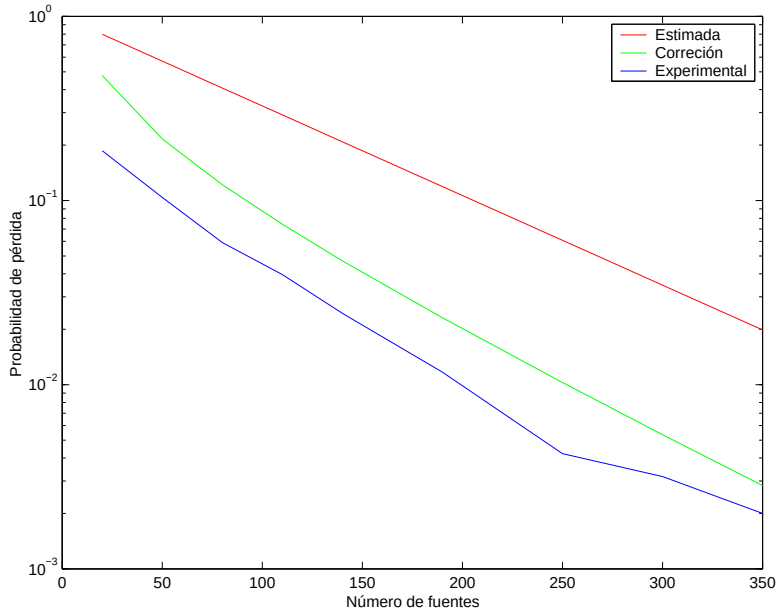

Figura 11.11: Probabilidad de overflow para el enlace más comprometido

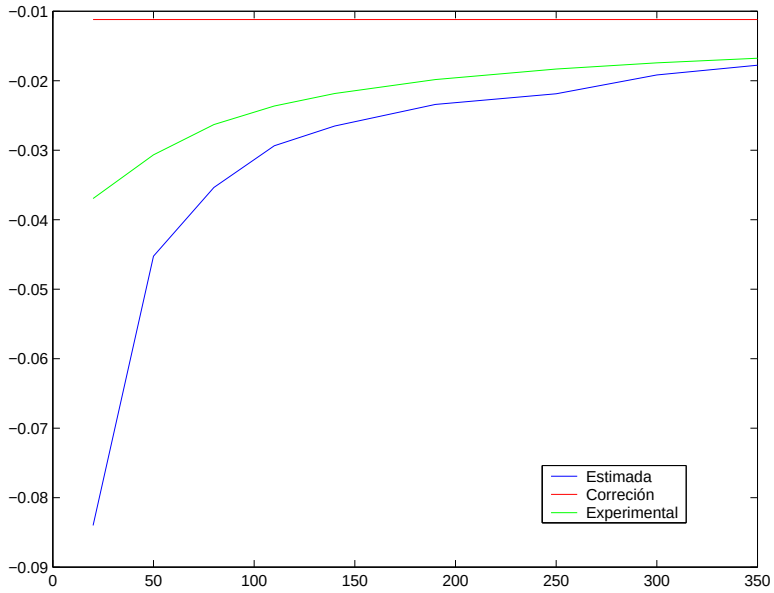

Figura 11.12: Tasa de pérdida para el enlace más comprometido

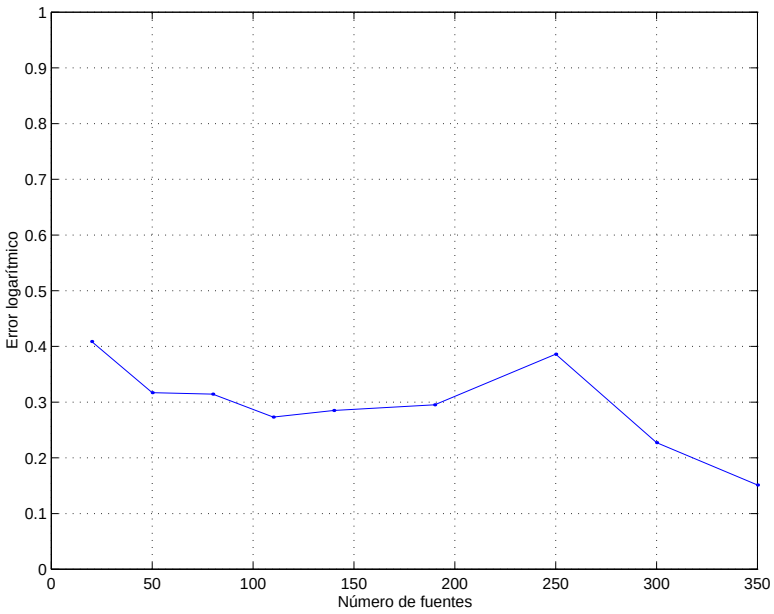

Figura 11.13: Error logarítmico para el enlace más comprometido

En las figuras se puede apreciar, como era de esperar, que a medida de las dimensiones de la red crecen, apareciendo mas buffers influyendo en el comportamiento del tráfico, la aproximación de $d_0 = 1$ comienza a carecer de precisión. Sin embargo las estimaciones obtenidas capturan apropiadamente la pendiente asintótica y muestran un error que disminuye al aumentar el número de fuentes.

11.5. Evidencia en el caso de la topología de árbol

Este caso, se pretende estimar el comportamiento en aquellos enlaces en los cuales se da una alta concentración de tráfico. El contar con una herramienta que permita estimar

dicho comportamiento puede ser de vital importancia, ya que el correcto dimensionamiento de este tipo enlaces, es clave para la calidad de servicio de punta a punta de la red, ya que puede ser el caso de la salida al mundo del tráfico proveniente de una red corporativa o de un país entero.

La topología simulada se puede apreciar en la figura 11.14.

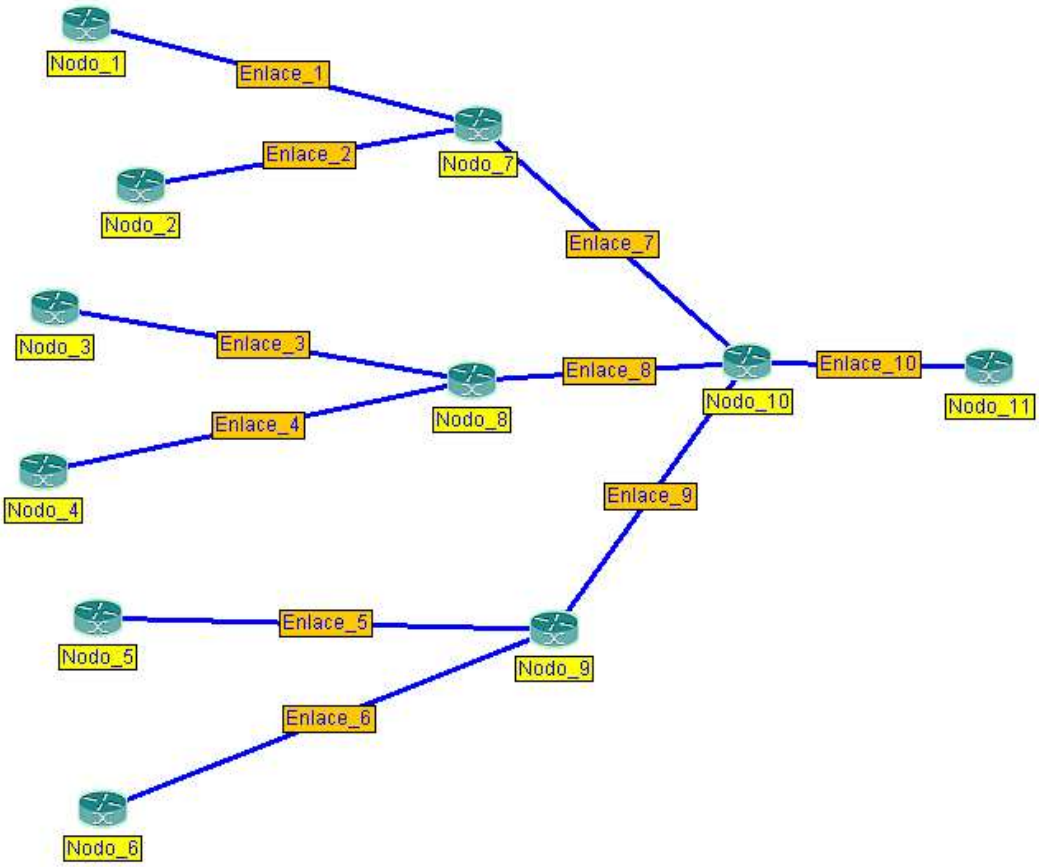

Figura 11.14: Topología de árbol simulada

Sobre esta topología se establecieron 6 LSP con origen en las ramas exteriores del árbol y destino el nodo raíz.

Las condiciones en que se realizó la simulación son:

- **Tipo de Flujo:** Se utilizó tráfico del tipo Markoviano ON-OFF. Para cada LSP se eligieron flujos con diferentes ciclos de trabajo. En todos los casos la tasa pico de generación de trabajo se tomó de 5 paquetes por unidad de tiempo.
- **Capacidades de los enlaces:** Se eligieron capacidades por fuente en cada nivel de forma de que éstas decrecieran hacia la raíz. Esto está de acuerdo con la idea de que a mayor cantidad de flujos multiplexados, se requiere menos capacidad por fuente.
- **Tamaño del buffer:** Los buffers se tomaron de la forma $b\sqrt{N}$ con N la cantidad de fuentes y, al igual que la capacidad, el valor de b se hizo decrecer hacia la raíz del árbol.
- **Tiempo de simulación:** La simulación se repitió 3 veces, con el objetivo de disminuir la varianza de la estimación, y el resultado que se presenta es el promedio

de todas las corridas. En todos los casos, el tiempo de simulación utilizado era de 150 segundos.

- **Número de Fuentes:** La cantidad de fuentes de tráfico que conformaban el agregado se varió desde 10 hasta 220 tratando de cubrir todas las zonas intermedias.

En las siguientes figuras se muestran los resultados obtenidos.

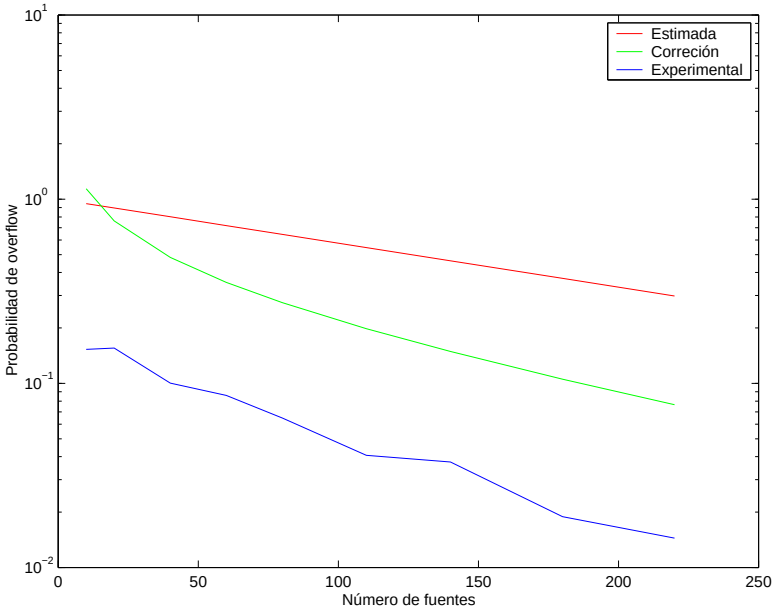

Figura 11.15: Probabilidad de overflow para el enlace raíz

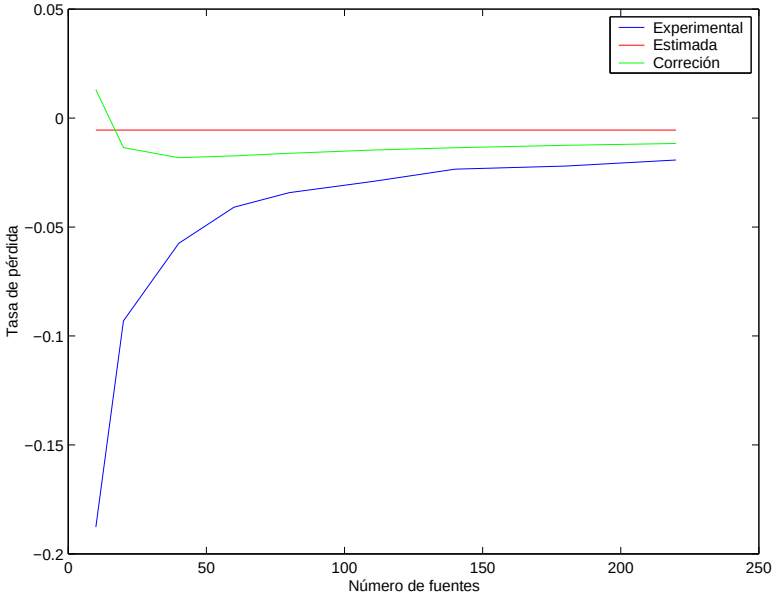

Figura 11.16: Tasa de pérdida para el enlace raíz

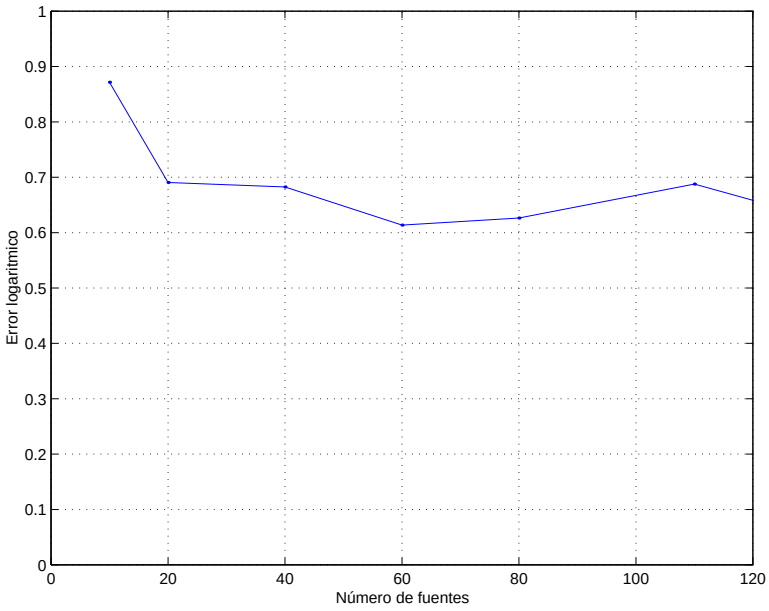

Figura 11.17: Error logarítmico para el enlace raíz

Se puede apreciar nuevamente, al igual que en el ejemplo anterior, que la aproximación de $d_0 = 1$ también comienza a carecer de precisión en este caso. Nuevamente las estimaciones capturan apropiadamente la pendiente asintótica, pero muestran un error importante, que igualmente en ningún caso supera el orden de magnitud.

11.6. Análisis de resultados

Los resultados más significativos que surgen de los ejemplos anteriores, indican que en todos los casos las estimaciones de la probabilidad de overflow constituyen una aproximación relativamente buena de su valor real. En particular se logró capturar de buena forma el comportamiento de la pendiente (tasa de pérdida).

Lamentablemente, en algunos casos, el modelo se aleja bastante de la realidad, brindando resultados que no son del todo precisos, sobre todo en aquellas redes en que los buffers adquieren gran relevancia. Con el estado actual de la teoría, no es posible mejorar dichas estimaciones, ya que no existe una expresión de mayor exactitud que la utilizada. Esto quedó comprobado en la simulación de un único enlace, donde se pudo estimar la probabilidad teórica de otro modo y se observó que el ajuste era bueno. Esto permitió atribuir el error cometido al modelo utilizado.

De todas maneras, podemos resaltar que, dados los cortos tiempos en que se obtienen los resultados, la gran generalidad de la técnica utilizada y la posibilidad de estudiar cualquier enlace de la red, las estimaciones obtenidas constituyen una muy buena aproximación de la probabilidad de overflow. Hay que resaltar que en todos los casos en que se trabajó, siempre se estuvo del lado de la seguridad, ya que en ninguno de ellos se subestiman las pérdidas. A su vez, el error obtenido en cada caso, nunca supera el orden de magnitud, que en muchos casos es el grado de aproximación que se necesita para realizar el diseño.

Capítulo 12

Aplicaciones de la técnica desarrollada

El objetivo del presente capítulo es presentar algunas casos de uso donde se pueden utilizan los resultados del proyecto en aplicaciones que requieran estimar parámetros de calidad de servicio. Como veremos existen diversas situaciones en las que es necesario estimar las condiciones de calidad de servicio para tomar decisiones.

Este tipo de situaciones se pueden dividir en dos categorías diferenciadas por la escala de tiempo en la que trabajan. En un caso puede ser interesante realizar estimaciones de performance en la etapa de diseño de una red, donde la escala de tiempo relevante es a largo plazo. En otro caso, las decisiones pueden ser necesarias en tiempos mucho menores, como es el caso de control de admisión de conexiones.

A continuación se presentan algunos ejemplos que ilustran las aplicaciones mencionadas y dan la pauta de la utilidad de una herramienta como la desarrollada en el presente trabajo.

12.1. Diseño de una red

En la actualidad el diseño de la red se realiza generalmente mediante predicciones sobre los requerimientos del tráfico que por la misma deberá pasar. En general dichas predicciones se basan en la experiencia previa o el conocimiento de situaciones similares, y en muchos casos se garantiza el cumplimiento de los Acuerdos de Nivel de Servicio (SLA por sus siglas en inglés) mediante el sobredimensionamiento de recursos. Si bien esto garantiza la calidad del servicio que los diferentes flujos de datos perciben, claramente no es un método eficiente de diseño.

Una práctica común hoy en día es el sobredimensionamiento del ancho de banda requerido por un determinado flujo reservando valores cercanos a la capacidad máxima que el mismo puede llegar a transmitir. Este enfoque no tiene en cuenta lo visto en el capítulo 2, donde se discutió que el multiplexado estadístico resulta beneficioso sobre todo desde el punto de vista de la economía de recursos. Sin embargo, para poder realizar correctamente el multiplexado estadístico, es necesario un análisis previo que tenga en cuenta el comportamiento aleatorio del tráfico.

Es aquí donde juegan un rol fundamental las herramientas que permiten realizar estimaciones de performance sobre la red a diseñar. A partir del conocimiento de la probabilidad de sobrecargas en cada uno de los enlaces de la red, uno puede determinar aquellas garantías que está en condiciones de ofrecer. Más aún, si se realizan ajustes sobre los parámetros de interés en la red se puede estudiar como varían dichas garantías

de calidad de servicio hasta obtener un diseño adecuado. A su vez se pueden tener en cuenta los diferentes requerimientos de los diferentes tipos de tráfico, y diseñar acorde a éstos.

La única alternativa para realizar lo anterior que existe hoy en día son las simulaciones. Esto presenta algunas desventajas importantes desde el punto de vista práctico, principalmente el tiempo necesario para obtener estimaciones razonables. Dado que se tienen muchos parámetros en juego en la etapa de diseño, y el hecho que cada variación de los mismos requiere nuevas simulaciones redunda en que esta etapa se torne tediosa lo que puede provocar soluciones subóptimas.

La herramienta desarrollada permite obtener estimaciones de la probabilidades de overflow, en tiempos significativamente menores que aquellos requeridos por simulaciones. La misma permite a su vez realizar estudios sobre tipos generales de tráfico siempre y cuando se disponga de muestras del mismo. En aquellos casos en que exista un modelo para el tráfico, el mismo puede ser tenido en cuenta para acelerar el tiempo de cálculo y obtener estimaciones mas apropiadas. Todo lo anterior la convierte en una herramienta de análisis de redes generales y resulta de particular valía en la etapa de diseño.

Debido al alcance acotado del proyecto, la herramienta propuesta presenta algunas limitaciones que podrán ser solucionadas en futuras versiones. En particular la herramienta actual permite un solo modelo teórico de tráfico. Teniendo en cuenta que la implementación de cada tipo diferente de flujo requiere un análisis teórico en profundidad de las técnicas de grandes desvíos involucradas, se optó por hacer hincapié en la extensibilidad de las bibliotecas desarrolladas y dejar la inclusión de otros modelos para posteriores versiones.

12.2. Ruteo basado en restricciones

El ruteo basado en restricciones aparece hoy en día como una alternativa para cumplir las garantías de calidad de servicio que requieren las aplicaciones modernas, como se mencionó en el capítulo 2. Algoritmos como el CSPF consisten en elegir como ruta a emplear aquella que presente el camino más corto entre origen y destino, luego de eliminar los caminos de la red que no provean las garantías adecuadas.

La implementación de estos algoritmos requiere conocer a priori cuáles son las garantías que cada enlace de la red ofrece. En general, al día de hoy esto se traduce en utilizar como restricción garantías de ancho de banda, es decir, eliminar aquellos enlaces que no puedan brindar el ancho de banda requerido por el flujo de datos. Este enfoque es equivalente a diseñar para el pico de tráfico en cada ruta, por lo que se pierden todas las ventajas mencionadas al discutir el multiplexado estadístico.

El disponer de otro método para evaluar la performance de cada enlace permitiría cambiar este enfoque conservador. Nuestra herramienta brinda la posibilidad de realizar una estimación eficaz de la tasa de pérdida de cada enlace, teniendo en cuenta las superposiciones existentes y el ancho de banda que efectivamente requiere cada fuente. Esto además, en un tiempo razonable como para acoplar la misma a técnicas existentes de ruteo basado en restricciones. Cuenta además con la posibilidad de realizar estimaciones a partir de medidas de tráfico, las cuales pueden ser realizadas periódicamente para seguir el comportamiento en el tiempo de los flujos y recalcular nuevas rutas cada cierto tiempo, obteniendo así un comportamiento adaptativo del ruteo.

En el estado actual de desarrollo, se cuenta con las clases básicas para la representación de la red y cálculo de las estimaciones pertinentes. La implementación de un algoritmo automático que realice ruteo basado en restricciones requiere además de la posibilidad de realizar mediciones sobre la red y alterar las tablas de ruteo en función de

los resultados, y por lo tanto su implementación depende de la tecnología a utilizar para la construcción de la red. Teniendo en cuenta esto, se tomó como objetivo el proveer una herramienta adecuada para representar la red independientemente de la tecnología que utilice, y obtener estimaciones rápidas sobre la performance de sus enlaces, lo que se logró satisfactoriamente.

12.3. Control de admisión de conexiones

Una de las áreas de mayor interés en la Ingeniería de Tráfico es el control de admisión de conexiones (CAC). El mismo consiste en condicionar la admisión del ingreso de un cierto tipo de tráfico a la red a que el mismo cumpla algunas características que permitan cumplir con los requerimientos del mismo, así como evitar que éste haga disminuir excesivamente los recursos disponibles para los otros flujos que atraviesan la red.

En general las características que se exigen consisten en limitaciones de media y tamaño de ráfaga del proceso de arribos, lo que no permite realizar estimaciones precisas del impacto en la red, debido a que aportan poca información sobre el comportamiento estadístico del mismo. Como contraparte, los parámetros no pueden ser complicados debido a que los cálculos que se realizan con ellos deben insumir poco tiempo.

El CAC originalmente se utilizaba en redes ATM, y estaba fuertemente relacionado al modelo IntServ, donde se pensaba en el control por fuente de tráfico como una alternativa de ingeniería de tráfico. La introducción del modelo DiffServ y el comenzar a pensar la ingeniería de tráfico desde el punto de vista de flujos agregados cambió la perspectiva del CAC. En redes de gran porte, puede ser interesante realizar CAC sobre agregados de tráfico. Es aquí donde es necesario realizar una estimación rápida de las características del tráfico a admitir, de modo de conocer su impacto en la red. La estimación inmediata de las características del tráfico es imprescindible para poder tomar las decisiones en línea.

La biblioteca desarrollada, a través del package topología, puede ser utilizada en este contexto. En nuestro modelo, la función de velocidad de cada tipo de tráfico constituye una medida de los recursos que el mismo necesita. Uno de los principales aportes realizados en dicha biblioteca es la introducción de una forma rápida de realizar una estimación de la misma, mediante interpolación polinómica. Se probó además que esta aproximación es suficientemente buena como para decidir los recursos del tráfico en base a ella. El package puede utilizarse entonces en un ambiente distribuido para realizar cálculos en línea, utilizando las funciones aproximadas.

A modo de ejemplo, puede diseñarse un algoritmo simple que permita implementar control de admisión sobre agregados de flujo. Supongamos que cada agregado está compuesto de una superposición de tráficos independientes y con la misma estadística. Entonces, todo lo que se necesita para evaluar el impacto en la red del mismo es la cantidad de fuentes que aporta y la función de velocidad. Durante la etapa de acuerdo de la conexión, puede utilizarse la función aproximada (polinómica) que se resume en un vector de coeficientes que el tráfico que solicita conexión brinda a la red. El punto de entrada a la red (el router LER en una arquitectura MPLS) calcula entonces las probabilidades de sobrecarga en toda la red y toma la decisión de aceptar o no el flujo agregado, de acuerdo a si la red puede cumplir con las garantías que este necesite, y no se dejan de cumplir las del resto de los flujos. Los tiempos de cálculo logrados por la biblioteca permiten por primera vez pensar en la aplicación de las estimaciones de la teoría de grandes desviaciones para este tipo de cálculos en línea.

Como complemento a este algoritmo, es necesario mantener un control sobre el comportamiento del tráfico que permita saber si los coeficientes utilizados en la admisión

siguen siendo válidos. Una forma de hacerlo es tomar mediciones periódicas sobre el tráfico y estimar la función de velocidad de manera exacta en concordancia, pero solo cada cierto tiempo. Esto permitiría actualizar las estimaciones de performance a intervalos regulares, aunque sin las exigencias temporales del control de conexiones. Los resultados de esta estimación pueden utilizarse para verificar que los flujos se comporten de acuerdo a lo declarado, y penalizar a aquellos que no lo hagan.

La única desventaja que se encuentra en este caso es que los índices de performance calculables mediante la técnica implementada se reducen a las probabilidades de sobrecarga. El desarrollo actual de la teoría no permite determinar otros índices a partir de la función de velocidad. Sin embargo, la herramienta actual constituye un importante primer paso en esa dirección.

12.4. Tarificación y multiplexado estadístico

Como se deduce de los resultados del capítulo 3 y observando las gráficas del capítulo 11, se aprecia que en la región de muchas fuentes el comportamiento tanto de la probabilidad de overflow como del ratio de pérdida de paquetes es aproximadamente lineal en escala logarítmica, con pendiente **I**. Este valor en sí constituye un índice de performance, que puede utilizarse para estimar la ganancia introducida por el multiplexado estadístico.

Supongamos que se conocen estimaciones de alguna de estas magnitudes, por ejemplo el ratio de pérdida, para un cierto número de fuentes N_0. Si se desea aumentar la cantidad de fuentes en un mismo agregado, debe reservarse más capacidad para el mismo. En términos de tarifas esto significa un aumento en la tarifa al menos lineal en la cantidad de fuentes.

Sin embargo, el aumento de la cantidad de fuentes produce una ganancia adicional en la performance, reduciendo las pérdidas. Dicha ganancia puede aproximarse por:

$$\log p_N - \log p_{N_0} \approx -(N - N_0)\mathbf{I}$$

donde p_N representa el valor del índice de performance para N fuentes.

En este marco, **I** juega el papel de la ganancia marginal en la performance debido al multiplexado estadístico. Por lo tanto, su valor puede utilizarse para calcular el costo marginal de agregar fuentes, lo que tiene especial interés en la fijación de tarifas. En particular, el aumento en la cantidad de fuentes y la capacidad ofrecida en la misma proporción conduce a un aumento de la performance, lo que permite afirmar que el costo marginal de agregar una fuente es decreciente. Por lo tanto, el valor de **I** puede utilizarse para introducir bonificaciones en el uso de un cierto enlace, al aumentar la cantidad de fuentes.

Capítulo 13

Líneas de trabajo futuro

Este capítulo está dedicado a escribir algunas de las áreas en que el proyecto admite extensiones.

13.1. Acercamiento de la herramienta a otras aplicaciones

Como se presentó en el capítulo 12, existen diferentes áreas donde la herramienta teórica expuesta resulta aplicable. En particular, la biblioteca desarrollada durante este proyecto permite pensar en nuevas aplicaciones, como la ingeniería de tráfico en línea.

Sin embargo, la versión actual de la aplicación tiene una clara orientación hacia la primera de las aplicaciones comentadas, es decir, el diseño de una red.

A pesar de esto, las componentes actuales pueden ser reutilizadas para acercar las técnicas de estimación de performance mediante grandes desviaciones a las aplicaciones restantes. Creemos que es realmente importante dar estos pasos porque la teoría existente es muy potente, en particular debido a la generalidad de modelos de tráfico con que permite trabajar. Esta cualidad es una de las mayores ventajas de esta teoría, y es menester aplicarla a las redes de datos actuales, donde precisamente la diversidad de modelos de tráfico es una característica deseada.

Creemos que a partir de las herramientas, tanto teóricas como prácticas, desarrolladas en este proyecto es posible comenzar esta integración de la teoría con las aplicaciones.

13.2. Extensión de la metodología a otros índices de performance

Como se desprende del análisis teórico realizado, el único índice de performance para el cual se ha logrado desarrollar estimaciones de buena calidad ha sido la probabilidad de desborde de la capacidad de un enlace.

Claramente, esto resulta insuficiente para tener una idea completa de la calidad de servicio que una red puede ofrecer, ya que se dejan de lado otros parámetros que también son de importancia en este aspecto, como ser el retardo de cola, el retardo de punta a punta, el jitter y la tasa de pérdida de paquetes que observa un flujo determinado, entre otros.

Hasta el momento, se ha dado un primer paso en este sentido, con la aproximación dada por Mazumdar para el caso de la tasa de pérdida de paquetes en un enlace, discutida en la sección 3.4.3. Esta aproximación permite resolver completamente el problema de un único enlace al que arriban muchas fuentes.

Resta aún mucho camino por recorrer en esta área, en particular creemos que la extensión del resultado anterior a los enlaces interiores de la red es posible, y constituye una importante línea de trabajo a seguir.

Para el resto de los índices de performance, la tarea aun está poco avanzada, pero creemos que la potencialidad de la teoría de grandes desviaciones no se ha desarrollado aun completamente, por lo que es de esperar que investigaciones en esta área abran nuevos caminos de trabajo futuro.

13.3. Análisis de otros modelos de tráfico

Uno de los puntos más débiles del estado actual de la teoría utilizada es que se dispone de pocos modelos teóricos manejables para la representación de distintos tipos de tráfico.

En primer lugar es necesario contar con análisis más profundos sobre la naturaleza estadística de los tipos de tráfico de interés en las redes modernas. Es necesario avanzar en la formulación de modelos que sean a la vez suficientemente descriptivos de los fenómenos en estudio, y que permitan un análisis completo desde el punto de vista de la teoría de grandes desviaciones.

Al momento de la realización del proyecto, el único modelo de importancia que permitió un análisis completo fue el modelo de Flujo Markoviano, que ha sido explotado a lo largo de todo el proyecto. Existen otros modelos, que también son tratables desde el punto de vista teórico, pero tienen grandes limitaciones para representar tipos reales de tráfico.

La investigación de nuevos modelos es una línea importante a seguir en el futuro. En particular, los modelos de tráfico basados en dependencias largas, o generalizaciones del Flujo Markoviano ya se encuentran en estudio, y es de esperar que a la brevedad se cuente con resultados que permitan extender la herramienta desarrollada para incorporarlos. La arquitectura de la aplicación fue pensada para que esto sea posible de manera sencilla.

Es importante aquí resaltar que estos análisis no pueden hacerse sin realizar mediciones en redes de gran porte, que son las que interesa modelar. A su vez, también es importante hacer notar que est tarea requiere de la intervención de diferentes disciplinas como la estadística, el tratamiento de señales y el análisis de redes de datos.

Capítulo 14

Conclusiones

La primera conclusión de carácter general es que los objetivos planteados al inicio del proyecto fueron alcanzados satisfactoriamente. En primer lugar, se logró compilar un conjunto de herramientas teóricas, que incluyen las técnicas de grandes desviaciones, herramientas estadísticas, modelos y algoritmos que permiten obtener estimaciones de performance en todos los puntos de una red a partir del conocimiento del tráfico que circula por la misma.

En segundo lugar, se consiguió plasmar dichas herramientas en una metodología práctica que permite efectuar el cálculo de los estimadores teóricos. En particular, la aplicación de software desarrollada permite relevar el funcionamiento de una red dada a partir de muestras del tráfico, estimando la probabilidad de overflow en cada uno de sus enlaces.

Se generó una biblioteca de clases que permite construir una topología de red arbitraria y realizar cálculos sobre la misma. Dicha biblioteca fue diseñada de forma de que fuera reutilizable por otras aplicaciones para realizar estimaciones de performance diferentes a las ya implementadas, utilizando la misma como base.

Como aportes adicionales a los objetivos iniciales, cabe mencionar que los tiempos de cálculo obtenidos en la versión final de la herramienta superaron ampliamente las expectativas que se tenían en un principio. La performance final de los algoritmos permite por primera vez pensar en aplicar estas técnicas con el objetivo de realizar ingeniería de tráfico en línea. La eficiencia en el cálculo se basa en que se ha logrado desarrollar un algoritmo de minimización específico para el complejo problema de optimización inherente a la estimación. A su vez, se propuso un modelo para aproximar las características estadísticas del tráfico que redujo significativamente el tiempo de procesamiento.

Otro aporte constituye el haber definido un lenguaje de base para utilizar las bibliotecas, y un conjunto de clases que permite actuar como interfaz entre el usuario y las herramientas de cálculo. En particular, si bien no estaba planteado como objetivo inicial, se desarrolló una interfaz gráfica que simplifica notoriamente el uso de la herramienta, permitiendo visualizar las topologías en estudio y realizar de manera sencilla modificaciones a la misma. Esto permite analizar el impacto en la performance producido por la variación de los distintos parámetros de la red, lo que resulta útil en la etapa de diseño.

Para nosotros los autores, este trabajo significó un crecimiento personal en diferentes aspectos. En particular, emprender una tarea de largo aliento, cuyos objetivos y plazos, así como la metodología de trabajo debieron ser definidas paso a paso a medida que se iba comprendiendo el problema planteado, constituyó una experiencia enriquecedora que nos acercó a la forma de trabajo de la vida profesional.

Una de las vetas más gratificantes del trabajo realizado es haber logrado incorporar al mismo conocimientos provenientes de una amplia gama de disciplinas de estudio. En

particular, fue necesario profundizar en el área de probabilidad, estadística, tratamiento de señales, ingeniería de redes, desarrollo de software, entre otras. Nos encontramos con que la formación previa nos aportó las bases necesarias para emprender la tarea, y al concluir la misma, habíamos enriquecido nuestros conocimientos en todas las áreas mencionadas.

Cabe resaltar que la herramienta desarrollada constituye un auspicioso primer paso en el desarrollo de una aplicación que permita realizar estimaciones de performance en redes. Como se observó en la validación de resultados los mismos no cuentan aún con la precisión suficiente como para dar por cerrado el problema. Sin embargo, como primer paso consideramos que es satisfactorio, quedando aún mucho por hacer. Los lineamientos discutidos en el capítulo 13 constituyen un punto de partida para el trabajo futuro.

Bibliografía

[1] L. Andersson, P. Doolan, N. Feldman, A. Fredette, B. Thomas, "LDP Specification", Internet Engeneeirng Task Force RFC 3036, Enero 2001.

[2] L. Aspirot; "Procesos condicionalmente débilmente dependientes y su aplicación a la estimación de anchos de banda efectivos", Monografía de Lic. en Matemática, Fac. de Ciencias, 2003.

[3] P. Bermolen; "Ancho de banda efectivo para flujos markovianos", Monografía de Lic. en Matemática, Fac. de Ciencias, 2003.

[4] C.S. Chang; *Performance Guarantees in Communication Networks*, Springer, 2000.

[5] C. Courcoubetis, R. Weber; "Buffer overflow asymptotics for a buffer handling many traffic sources", *Journal of Applied Probability* 33, pp. 886-903, 1996.

[6] C. Courcoubetis, V. Siris; "Procedures and tools for analysis of network traffic measurements", Performance Evaluation, 48, pp. 5-23, 2002.

[7] G. Dahlquist, B. Björck, I. Andersson; "Numerical Methods", Prentice Hall, 1974

[8] A. Dembo, O. Zeitouni; *Large Deviations Techniques and its Applications*, Bartlett and Jones, 1991.

[9] B. Eckel; *Thniking in Java* 2nd. Ed., Prentice-Hall, 2001.

[10] Grupo ARTES; "Quality of service parameters and link operating point estimation based on effective bandwidths", *Proceedings of the 3rd Confernce on Hetergeneous Networks (HetNet03)*, 2003.

[11] IEC Trillium, "Multiprotocol Label Swiching".

[12] M. Iltis; "Sharp asymptotics of large deviations in $\mathbb{R}^d$. *Journal of Theoretical Probability* **8**(3), pp. 501-524, 1995.

[13] F. Kelly; "Notes on Efective Bandwiths"in *Stochastic Networks: Theory and Applications*, pp. 141-168. Oxford, Oxford University Press, 1996.

[14] G. Kesidis, J. Walrand, C.S. Chang; "Effective Bandwidths for Multiclass Markov Fluids and Other ATM Sources", IEEE/ACM Trans. Networking, No.1, pp. 424-428, 1993.

[15] N. Likhanov, R. Mazumdar; "Cell loss asymptotics in bufferws fed with a large number of independent stationary sources", *Journal of Applied Probability*, **36**(1), pp. 86-96, 1999.

[16] N. Likhanov, R. Mazumdar,R., O. Özturk; "Many sources Asymptotics for Networks with Small Buffers", Queueing Systems (QUESTA), **46** (1-2), pp. 129-147, 2004.

[17] J.M. Ortega, W.C. Rheinboldt; *Iterative solution of nonlinear equations in several variables*, Academic Press New York, 1970.

[18] J. Pechiar, G. Perera, M. Simon; "Effective Bandwidth estimation and testing for Markov sources", Performance Evaluation 48, pp. 257-175, 2002.

[19] Rabinovitch, P. (2000) "Statistical estimation of effective bandwidth", M.Sc.thesis, University of Cambridge.

[20] E. Rosen, A. Viswanathan,R. Callon, "Multiprotocol Label Switching Architecture", Internet Engeneering Task Force RFC 3031, Enero 2001.

[21] Wischik, D. The Output of a switch, or, effective bandwidths for networks. *Queuing Systems* 32, pp. 383-396, 1999.