Tarifación de Redes

Proyecto de Fin de Carrera de Ingeniería Eléctrica, plan 97, perfil Telecomunicaciones

Rodrigo de Andrés[1], Claudina Rattaro[2], Priscilla Severgnini[3]

Tutores
Ing. Andrés Ferragut
Ing. Pablo Belzarena

Facultad de Ingeniería
Universidad de la República

Mayo 2008

[1]grdamail@gmail.com
[2]claudira@gmail.com
[3]prissever@gmail.com

Resumen

El presente documento contiene la documentación final del Proyecto de Fin de Carrera titulado "Tarifación de Redes". Los autores del mismo: Rodrigo de Andrés Lucas, Claudina Rattaro Eugui y Priscilla Severgnini Hernández son estudiantes de la carrera Ingeniería Eléctrica perfil Telecomunicaciones de la Facultad de Ingeniería - UdelaR.

El trabajo se llevó a cabo en el período comprendido entre marzo de 2007 y mayo de 2008 bajo la tutoría de los ingenieros Pablo Belzarena y Andrés Ferragut.

Este proyecto de fin de carrera forma parte del proyecto PDT de tarifación de redes multiservicio, cuya finalidad es analizar el problema que se presenta cuando se desean agregar servicios como voz, datos y telefonía sobre una misma red, haciendo énfasis en como tarifar dichos servicios dependiendo de los requerimientos de los usuarios.

El aporte de este proyecto al grupo de tarifación PDT es la implementación de un simulador de redes a escala de flujos que permita aplicar diferentes políticas de tarifación y así poder estimar ganancias de los operadores y comportamiento de los usuarios de la red. Tomando como base el estudio de la performance de la red frente a distintas políticas de tarifación, es posible emplear técnicas de Ingeniería de Tráfico para poder brindar cierta Calidad de Servicio (QoS).

Para la implementación del software se eligió hacer un simulador de eventos discretos a escala de flujos. Dicho simulador tiene como entradas la topología de la red, datos de los usuarios y diferentes políticas de tarifación, y como salida, el comportamiento de la red y ganancias que se obtendrían de la aplicación de dichas políticas.

El simulador no fue diseñado pensando en un tipo específico de arquitectura sino que se puede armar la topología que se desea. Sobre la topología se distinguen dos tipos de flujos: *flujos elásticos* (representando el tráfico de datos, flujos que no necesitan un rate fijo) y *flujos streaming* (flujos de voz y video) que sí precisan de un rate fijo para ser enviados.

Al conjunto de servicios de voz, datos y video que brinda un mismo operador sobre una misma red física se le llama Triple Play. Para familiarizarse con este tema se dedicó un capítulo para explicar las componentes y tecnologías que intervienen en las arquitecturas Triple Play en desarrollo actualmente.

El documento consta de cuatro partes. La *Parte I* describe el problema planteado y las etapas que llevaron a su implementación. Además se dedicó un capítulo

para explicar la solución Triple Play. La *Parte II* explica la solución implementada y en la *Parte III* se indican los resultados teóricos utilizados. Por último, en la *Parte IV* se enumeran las conclusiones obtenidas en las simulaciones realizadas.

Con la presente documentación se adjunta un CD conteniendo:

- Este documento
- JavaDoc
- Ejemplos de prueba para el simulador
- Archivos correspondientes a las clases implementadas

Agradecimientos

- A nuestras familias que nos han apoyado a lo largo de toda la carrera
- A Verónica y a Marco por apoyarnos en todo momento, por darnos toda la energía que necesitamos para realizar un trabajo de esta envergadura.
- A nuestros tutores por haber depositado su confianza en nosotros y haber dirigido este proyecto.
- A integrantes de ALCATEL por brindarnos información valiosa para el desarrollo del proyecto
- A Mario de Oliveira, de Nuevo Siglo, por brindarnos su tiempo
- A los integrantes del grupo PDT
- A Julia Demasi por ayudarnos en la organización del documento.

Índice general

I Descripción del Problema 15

1. Introducción 17
   1.1. Objetivos 17
   1.2. Planificación 17

2. Motivación 19
   2.1. Introducción 19
   2.2. Limitaciones de las redes actuales 20
   2.3. Propuesta de Alcatel 20
      2.3.1. Arquitectura y funcionalidades de la red Triple Play 20
   2.4. Propuesta de empresas cableras 22
      2.4.1. Cable data transport 22
      2.4.2. Telefonía por cable 22
   2.5. Conclusión 23

II Desarrollo de la Herramienta de Software 25

3. Proceso de desarrollo de la herramienta de simulación 27

4. Descripción del Software 31
   4.1. Frontera 31
   4.2. Requerimientos del Sistema 31
   4.3. Actores 32
   4.4. Casos de Uso 32
      4.4.1. Caso de Uso: Configuración del simulador 32
      4.4.2. Caso de Uso: Correr Simulación 32
   4.5. Diagramas de Interacción 36
      4.5.1. DSS: Solicitar Elástico 36
      4.5.2. DSS: Comienza Elástico 37
      4.5.3. DSS: Termina Elástico 38
      4.5.4. DSS: Solicitar Streaming 39
      4.5.5. DSS: Comienza Streaming 40
      4.5.6. DSS: Termina Streaming 41
      4.5.7. DSS: Llamada a controlador de gestores 42
      4.5.8. DSS: Oferta insuficiente para streaming 43
      4.5.9. DSS: Fin de la simulación 43
   4.6. Descripción y diagramas de las clases implementadas 44
      4.6.1. Package Topología 44
      4.6.2. Package Eventos 45
      4.6.3. Package Exceptions 46

4.6.4. Package Flujos 46
4.6.5. Package Main 47
4.7. Obtención de estadísticas 48

III Fundamentos teóricos utilizados 49

5. Algoritmo de Max - Min Fairness 51
5.1. Introducción 51
5.2. Implementación 52

6. Políticas de Tarifación 55
6.1. Ingeniería vs Economía 55
6.2. Herramientas de la teoría económica 56
6.3. Implementación 57

IV Descripción de las simulaciones 61

7. Introducción 63

8. Simulaciones con flujos streaming 65
8.1. Simulaciones con topologías formadas únicamente por un enlace . . 65
8.1.1. Simulación 1 65
8.1.2. Simulación 2 68
8.2. Simulaciones con topologías formadas por dos enlaces 71
8.2.1. Simulación 1 71
8.2.2. Simulación 2 75

9. Simulaciones con flujos elásticos 77

10. Simulaciones con flujos elásticos y flujos streaming 81
10.1. Simulación 1 81
10.2. Simulación 2 82

11. Conclusiones 85

Bibliografía 85

A. Triple Play 91
A.1. Introducción 91
A.2. Propuesta de ALCATEL 92
A.2.1. Limitaciones de las redes actuales 92
A.2.2. Arquitectura y funcionalidades de la red Triple Play 92
A.2.3. Tecnologías Asociadas 94
A.3. Propuesta de empresas cableras - Nuevo Siglo 99
A.3.1. Cable Data Transport 99
A.3.2. Telefonía por Cable 105
A.4. Conclusión 108

B. Descripción de las clases implementadas 109
B.1. Package Topología . . . 109
B.1.1. Clase Agente . . . 109
B.1.2. Clase ClienteElastico . . . 110
B.1.3. Clase ClienteStreaming . . . 111
B.1.4. Clase Nodo . . . 113
B.1.5. Clase Enlace . . . 113
B.1.6. Clase Ruta . . . 115
B.1.7. Clase ServidorElastico . . . 116
B.1.8. Clase ServidorStreaming . . . 117
B.1.9. Clase ControladorDeGestores . . . 118
B.1.10. Clase Gestor . . . 119
B.1.11. Clase Topologia . . . 121
B.2. Package Eventos . . . 122
B.2.1. Clase Evento . . . 122
B.2.2. Clase ComienzaElastico . . . 122
B.2.3. Clase ComienzaStreaming . . . 123
B.2.4. LlamadaAControladorDeGestores . . . 124
B.2.5. Clase SolicitarElastico . . . 124
B.2.6. Clase SolicitarStreaming . . . 125
B.2.7. Clase TerminaElastico . . . 125
B.2.8. Clase TerminaStreaming . . . 126
B.2.9. Clase OfertaInsuficienteParaStreaming . . . 127
B.2.10. EventoDeFin . . . 127
B.2.11. Clase ListaDeEventos . . . 128
B.3. Package Flujos . . . 129
B.3.1. Clase Flujo . . . 129
B.3.2. Clase FlujoElastico . . . 130
B.3.3. Clase FlujoStreaming . . . 131
B.4. Package Main . . . 132
B.4.1. Clase GeneradorAleatorio . . . 132
B.4.2. Clase MaxMinFairness . . . 132
B.4.3. Clase Principal . . . 133
B.4.4. Clase Reader . . . 134
B.5. Package Exceptions . . . 134
B.5.1. RutaNoContinua: . . . 134
B.5.2. DestinoNoAlcanzable . . . 134

C. Plan de Trabajo - Entregado en Mayo de 2007 135
C.1. Objetivo general del Proyecto . . . 135
C.2. Actores, supuestos y restricciones . . . 136
C.3. Objetivos específicos . . . 136
C.4. WBS . . . 137
C.5. Objetivos planteados para los entregables intermedios . . . 137
C.6. Análisis de riesgos . . . 138
C.7. Cronograma del proyecto - Listado de tareas . . . 139

D. Evaluación de la planificación 141

E. Simulaciones 143
E.1. Verificación del primer prototipo - flujos streaming . . . 143
E.2. Verificación de la implementación del algoritmo de Max Min Fairness 147

F. Manual de Usuario 149
F.1. ¿Cómo definir un nodo? 149
F.2. ¿Cómo definir un enlace? 150
F.3. ¿Cómo definir un cliente que solicite flujos streaming? 151
F.4. ¿Cómo definir un cliente que solicite flujos elásticos? 152
F.5. ¿Cómo definir un ServidorElastico? 152
F.6. ¿Cómo definir un ServidorStreaming? 153
F.7. ¿Cómo definir una ruta? 153
F.8. ¿Cómo definir un Gestor? 153
F.9. ¿Cómo definir un ControladorDeGestores? 154
F.10. ¿Cómo definir el Evento Fin? 154
F.11. ¿Cómo habilitar las trazas a los distintos elementos de la red? . . . 154

Índice de figuras

2.1. Arquitectura Triple Play propuesta por Alcatel 20
2.2. Calidad de servicio en los diferentes niveles de agregación 21

4.1. Solicitar Elástico 36
4.2. Comienza Elástico 37
4.3. Termina Elástico 38
4.4. Solicitar Streaming 39
4.5. Comienza Streaming 40
4.6. Termina Streaming 41
4.7. Llamada a controlador de gestores 42
4.8. Oferta insuficiente para streaming 43
4.9. Fin de la simulación 43
4.10. Relaciones importantes entre clases del paquete topología 45
4.11. Clases del paquete Eventos 46
4.12. Clases del paquete Flujos 47

5.1. Asignación de recursos en una red sencilla 52

6.1. Ejemplo de función de utilidad 56
6.2. Ejemplo de función de utilidad de un Gestor de la red 58
6.3. Función utilidad de un Gestor de la red 59

8.1. Representación de la topología simulada 65
8.2. Ganancias por remate, media de duración de los flujos 100 66
8.3. Ganancias por remate, media de duración 300 68
8.4. Ganancias por remate G1 69
8.5. Ganancias por remate G2 69
8.6. Ganancias por remate G3 69
8.7. Representación de la topología simulada 71
8.8. Cantidad de ofertas aceptadas por remate - G1 73
8.9. Cantidad de ofertas aceptadas por remate - G2 73
8.10. Cantidad de ofertas aceptadas por remate - G3 74

9.1. Representación de la topología simulada 77

10.1. Distribución de la capacidad del enlace 1 82
10.2. Distribución de la capacidad del enlace 2 82
10.3. Representación de la topología simulada 83

A.1. Arquitectura Triple Play propuesta por Alcatel 93
A.2. Calidad de Servicio en los diferentes niveles de agregación 94
A.3. Sistema FSK 100

A.4. Sistema de transmisión BPSK 101
A.5. Sistema básico QPSK 102
A.6. Integración del DLC 106
A.7. Modificación de la arquitectura de DLC para dar Telefonía por Cable 107
A.8. Elementos del sistema telefónico 107
A.9. Interfaz telefónica a nivel del usuario 108

C.1. Análisis de Riesgos 138
C.2. Tareas Planificadas 139

E.1. Comparación de los datos experimentales con los teóricos dados por Erlang B 144
E.2. Histograma 145
E.3. Comparación de los datos experimentales con los teóricos dados por Erlang B 146
E.4. Histograma 146
E.5. Histograma 147
E.6. Histograma 148

Índice de cuadros

8.1. Solicitudes en el primer remate 66
8.2. Solicitudes en el segundo remate 67
8.3. Ganancias medias obtenidas por cada Gestor 70
8.4. Solicitudes para el G1 - ruta 1 (azul) 72
8.5. Solicitudes para el G2 - ruta 2(verde) 72
8.6. Solicitudes para el G3 - ruta 3(roja) 72
8.7. Solicitudes para el G1 (Segundo Remate) - ruta 1(azul) 74
8.8. Solicitudes para el G2 (Segundo Remate) - ruta 2(verde) 74
8.9. Solicitudes para el G3(Segundo Remate) - ruta 3(roja) 74

9.1. Tasas asignadas en el tiempo 7,46 78
9.2. Tasas asignadas en el tiempo 7,58 - después de terminar de enviar un flujo 79

A.1. Características del downstream RF 103
A.2. Parámetros para 64 QAM 104
A.3. Parámetros para 256 QAM 104
A.4. Características del upstream RF 104
A.5. Parámetros para QPSK 104
A.6. Parámetros para 16 QAM 104

Parte I

Descripción del Problema

Capítulo 1

Introducción

En el presente capítulo se presentan los objetivos planteados al inicio del proyecto.

1.1. Objetivos

El objetivo general planteado para el proyecto, fue implementar un entorno de simulación donde se pueda definir una red con características de red Triple Play, tanto en su topología como en el tipo de conexiones que demandan los usuarios. A partir de este entorno, se estudiaría el comportamiento de la red debido a diferentes perfiles de usuario aplicando las diferentes políticas de tarifación que surjan del proyecto PDT y así poder evaluar los recursos del operador.

Se consideraron como criterios de éxito:

- Llegar al fin del proyecto en el tiempo estipulado cumpliendo con los requisitos propuestos, entre ellos, el más importante, que el software implementado permita simular el tráfico de una red Triple Play considerando:

    1. Perfiles de Usuarios
    2. Tipos de tráfico
    3. Políticas de tarifación e Ingeniería de Tráfico

- Que el simulador cuente con una interfaz de usuario con manual detallado incluyendo ejemplos se uso típico.

- Realizar un resumen de las soluciones Triple Play.

1.2. Planificación

Para cumplir con la meta propuesta, se dividió el objetivo general en objetivos específicos que se detallarán a continuación.

Objetivos Específicos

1. Estudiar la tecnología Triple Play y las posibles formas de implementarla en el simulador. Presentar un documento conteniendo el resumen de lo concluido.
2. Estudiar los diferentes tipos de arquitectura para extraer las características esenciales de las redes a simular.
3. Realizar un software que permita simular flujos en una red con arquitectura Triple Play, y que permita aplicar políticas de tarifación y asignación de recursos.
4. Verificar funcionamiento del simulador y realizar ajustes en caso de ser necesario.
5. Implementar una interfaz de usuario
6. Implementar políticas de tarifación que surjan del proyecto PDT
7. Realizar documentación total del proyecto
8. Preparar entregables intermedios

Siguiendo con la planificación, para cada objetivo específico se definió una serie de tareas a realizar y a dichas tareas se les asignaron horas de ejecución. Se realizó un Diagrama de Gantt con todos los hitos a cumplir ordenados cronológicamente.

Por más detalles, ver Apéndice C.

Durante la ejecución del proyecto se realizaron dos entregables intermedios.

Para el primer entregable[1] se propuso como objetivos:

- Completar un primer prototipo del simulador que represente una arquitectura y flujos sencillos. Entregar ejemplos de prueba y evaluar los tiempos de ejecución de las simulaciones.
- Realizar un documento describiendo las principales componentes de una arquitectura Triple Play y cómo se planea integrar éstas al simulador.
- Entregar un documento con el plan de diseño del software final, informando que se implementó hasta el momento.

Para la segunda entrega[2], se planteó como meta tener pronta la versión de pruebas del simulador, una primera versión de la documentación del software e informar sobre el diseño de una Interfaz de Usuario para la herramienta.

Para la entrega final, una vez cumplidos todos los objetivos específicos, se planteó la posibilidad de realizar una interfaz gráfica amena al usuario con su manual correspondiente.

[1] Setiembre de 2007
[2] Febrero de 2008

Capítulo 2

Motivación

2.1. Introducción

La motivación principal para desarrollar un simulador como el propuesto y realizado durante este proyecto, es que actualmente no se dispone de simuladores que permitan implementar políticas de tarifación. Para ello, es necesario adaptar un simulador existente, o crear uno. La mayoría de los simuladores existentes de código abierto trabajan en la escala de tiempo de paquetes, es decir, simulan el intercambio de cada paquete en la red. Esta escala no es la adecuada para aplicar políticas de tarifación ya que es técnicamente inviable controlar paquete a paquete la aplicación de precios. Más aún, los usuarios no perciben la calidad a nivel de los paquetes enviados sino de los flujos.

Un flujo es un intercambio del usuario con la red, como por ejemplo el envío de un archivo, la descarga de una página web, ver un video en tiempo real, etc.

Actualmente no hay disponibles simuladores de red a escala de flujos, que es una escala de interés para la tarifación, ya que permite mayor granularidad que las políticas de tarifación aplicadas hoy en día. Actualmente las diferentes formas de tarifar se implementan a escala de tiempo de conexión o según los bytes transferidos (en un mes por ejemplo).

Por lo expuesto, se decidió desarrollar un simulador propio, que permita definir diferentes topologías y tipos de tráfico que aparecen en una red del tipo Triple Play y que permita aplicar políticas de tarifación adecuadas, que serán parte del trabajo que desarrolla el grupo del Proyecto PDT.

Como se mencionó, el software debe simular características de una red Triple Play. Para ello se estudiaron diferentes formas de implementar esta solución.

La solución Triple Play surge con la finalidad de ofrecer un servicio mucho más atractivo para los clientes, su objetivo es transmitir voz, datos y video en una única red IP.

El interés por los servicios de convergencia es una constante en los últimos encuentros de telecomunicaciones de Latinoamérica, como parte de una tendencia mundial que parece irreversible. Telcos, "cableras", proveedores de servicios de Internet y hasta las compañías eléctricas se ven ahora como posibles rivales en un

mercado que suma complejidad y dispares protagonistas.

En las siguientes secciones se resumen brevemente dos arquitecturas Triple Play diferentes, explicando sus componentes esenciales. En el Apéndice A se brindan más detalles de las mismas.

2.2. Limitaciones de las redes actuales

Las redes actuales son inadecuadas para la implementación de la nueva tecnología ya que no fueron diseñadas para la optimización de la transmisión de voz y video en tiempo real (en gran parte ocasionado por el jitter[1] y por ser IP un servicio best effort[2]), además el ancho de banda disponible para la transmisión de video no garantiza un servicio óptimo.

2.3. Propuesta de Alcatel

Se eligió explicar la solución de Alcatel, debido a que es un proveedor de varias Telcos y con fuerte presencia en Uruguay.

2.3.1. Arquitectura y funcionalidades de la red Triple Play

La arquitectura Triple Play se basa en la división de la red de acceso en dos regiones para hacer un gerenciamiento de tráfico, lo que permite hacer un control de tráfico en diferentes escalas (tanto a nivel de usuarios como a escala de servicios).

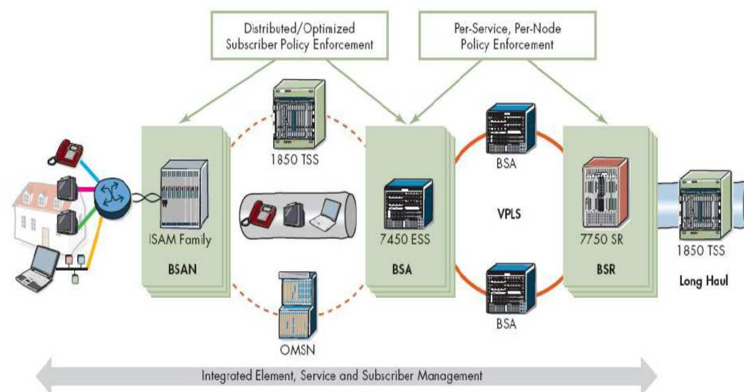

Figura 2.1: Arquitectura Triple Play propuesta por Alcatel

La propuesta de Alcatel consiste en utilizar diferentes equipos en diversas regiones de la red para garantizar una buena calidad de servicio.

[1]Jitter: variación en el retardo
[2]Best Effort: "el mejor esfuerzo", define la forma de prestar aquellos servicios para los que no existe una garantía de QoS. Esto implica que no existe una preasignación de recursos, ni plazos conocidos, ni garantía de recepción correcta de la información.

A continuación, se detallan los nodos que se muestran en la Figura 2,1.

BSA (Broadband Service Aggregator)

Como se mencionó antes, para garantizar calidad de servicio, Alcatel propone la división del acceso, el BSA es el nodo encargado de agregar el tráfico por usuario en tráfico por servicio.

Este equipo es un switch Ethernet con soporte de QoS y policing [3], que se conecta via Ethernet/VPLS al BSR.

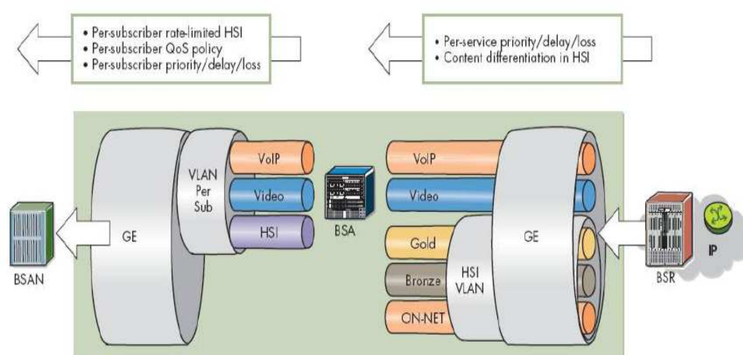

Figura 2.2: Calidad de servicio en los diferentes niveles de agregación

BSR (Broadband Service Router)

Es el equipo que se encuentra entre la red Ethernet/VPLS e IP/MPLS. Asigna vía DHCP direcciones IP (en lugar de conexiones PPPoE) a los usuarios. Junto con el BSA y la red VPLS sustituyen a la vieja arquitectura BRAS/ATM de agregación de las redes de acceso.

BSAN (Broadband Service Access Node)

Es el elemento de la red que se encarga de realizar el filtrado, autenticación de usuarios e IP Multicast. [3]

Alcatel propone una solución para la red de acceso que se apoya en tecnologías[4] como IP Multicast, MPLS, VPLS, etc, para la correcta implantación de una red Triple Play que garantice buenos niveles de QoS.

[3]policing: aplicación de políticas sobre los flujos
[4]En el Apéndice A se explica un resumen de algunas de las tecnologías asociadas

2.4. Propuesta de empresas cableras

2.4.1. Cable data transport

No sólo las empresas de telefonía han adoptado la nueva tendencia de proveer a sus clientes de servicios de datos y televisión agregados a los ya tradicionales como la voz, sino que empresas de televisión para abonados se han sumado a este gran desafío.

Los factores que hacen al cable un medio atractivo son su precio, velocidad y que no es orientado a conexión.

La tecnología asociada a la transmisión de datos en una red de cable es el CABLE MODEM y el protocolo utilizado es el DOCSIS.

Protocolo Docsis en Cable MODEM

Docsis es el protocolo utilizado para el transporte de datos en cable modem. El estándar le permite al consumidor cambiar de sistema de cable, utilizando el mismo modem.

Capa Física

La capa física se encarga de la modulación de los datos y de la correcta sincronización entre el transmisor y receptor.

Capa MAC

La capa MAC se encarga de la asignación de ancho de banda, es decir, de la determinación de la frecuencia y tasa de datos a utilizar. Esto es controlado por el CMTS (Cable Modem Termination System) de acuerdo a parámetros programados y condiciones existentes.

Debido a que los "upstream messages"son bastante cortos, el protocolo permite, subdividir los time slots permitiendo el acceso a una mayor cantidad de módems, y así mejorar la eficiencia.

Docsis agrega una subcapa de seguridad, adicional a la de capa Mac, SSI (Security System Interface). Cable Modems tienen la libertad de implementarla o no.

2.4.2. Telefonía por cable

El servicio de telefonía en una red de cable no se encuentra aún estandarizado, por lo que se verá una limitada representación de lo que se está usando.

En un principio los sistemas de telefonía estaban basados en tecnologías de switching circuits, pero ahora cada vez más la tendencia es la orientación a servicios sobre IP. En el Apéndice A se detalla un poco de ambas.

Telefonía IP

La tecnología IP esta generando un gran interés en las empresas de cable como alternativa al servicio convencional de telefonía.

El mismo, es visto como un servicio de bajo costo para grandes distancias con una interfaz local donde se utiliza Internet.

Las señales de voz son digitalizadas, comprimidas y enviadas como paquetes de datos por la red IP. En la práctica se presentan algunas limitaciones, como por ejemplo, que solo puede hablar uno de los lados por vez. [7]

2.5. Conclusión

Los proveedores a nivel mundial están optando por la venta de paquetes de servicios para asegurar la fidelidad de los clientes. Los servicios más atractivos parecen ser video, voz y datos. Los operadores para poder ofrecerlos necesitan realizar cambios a sus redes actuales, que no se encuentran diseñadas y optimizadas para este propósito.

Actualmente cada servicio tiene su forma distinta de tarifa, al surgir ésta convergencia se necesitará también, una convergencia en las políticas de tarifación. La gran motivación del proyecto radica en la falta de una herramienta que permita simular una red multiservicio y además permita estudiar los distintos impactos obtenidos al aplicar (en dicha red) diferentes políticas de tarifación.

Este proyecto constituye un primer paso en esta dirección.

Parte II

Desarrollo de la Herramienta de Software

Capítulo 3

Proceso de desarrollo de la herramienta de simulación

Como se explicó en el capítulo anterior, el objetivo principal del proyecto era elaborar una herramienta de simulación que permitiera aplicar diferentes políticas de tarifación para una topología de red con arquitectura Triple Play. Para ello, se optó por utilizar el lenguaje de programación Java, ya que todos los integrantes del proyecto estaban familiarizados con el mismo. Otro criterio usado a la hora de elegir el lenguaje fue, que al ser Java un lenguaje tan conocido, es bastante sencillo encontrar documentos, bibliotecas, etc.

Se eligió implementar un simulador de eventos discretos. Para comprender que se entiende por "Simulador de eventos discretos", en una primera instancia se estudió el libro "Discrete - Event System Simulation"[1].

Un simulador de eventos discretos se utiliza para modelar aquellos sistemas en los cuales el estado de sus variables cambia únicamente en ciertos instantes de tiempo. Nuestro caso, es claramente un sistema discreto ya que el estado de las variables, la cantidad de flujos enviándose en la red, la cantidad de ofertas solicitadas, entre otras cosas, varían únicamente cuando un flujo comienza a ser enviado, o cuando un flujo termina de enviarse, cuando un cliente realiza una petición de un flujo, etc. [1]

Como objetivo inicial, se planteó la implementación de un primer prototipo que simule una arquitectura y flujos sencillos. Se decidió implementar una red formada únicamente por un enlace[1] y varias fuentes de flujo. Objetivo cumplido para el primer entregable. Para llegar al primer prototipo se realizaron varias etapas:

Lo primero que se implementó fue un sistema de colas **FIFO** para flujos de rate fijo. La filosofía de este primer sistema era dejar pasar todos los flujos que permitía la capacidad del enlace y luego encolar los restantes, estos últimos eran enviados en orden de llegada a medida que quedaba capacidad libre en el enlace.

Acercándose un poco más al comportamiento de los flujos streaming en la realidad, se implementó un sistema basado en pérdidas. Al comenzar a enviar un flujo, si la capacidad libre del enlace en ese momento no era suficiente, el flujo

[1] Elemento de la red que permite el envío de flujos limitado por su capacidad

era descartado. El servidor implementado enviaba flujos de rate fijo de manera aleatoria con distribución exponencial. Se realizaron simulaciones para esta implementación, chequeando los datos experimentales con los teóricos dados por la fórmula de **Erlang B**[2].
Las simulaciones realizadas se encuentran en el Apéndice E. Además se evaluaron los tiempos de ejecución de esta primer versión del software, los cuales resultaron aceptables.

Con lo anterior, se tenía un simulador de un enlace y un servidor que generaba flujos que debían ser enviados a una velocidad fija, como lo son flujos de video y de voz. Para el entregable de setiembre se logró además, implementar otro tipo de Servidor, un **ServidorElastico**. Los flujos generados por este tipo de servidor tratan de emular el tráfico de internet[3].

En el caso de existir flujos streaming y elásticos presentes en la red, se tomó el siguiente criterio:

El 80 % de la capacidad total del enlace podía ser ocupada por flujos Streaming. El rate asignado a cada flujo elástico enviándose en la red era igual a la capacidad libre del enlace[4] sobre la cantidad de flujos elásticos enviándose en ese instante.

Cabe resaltar, que este primer prototipo llevó más tiempo del planificado, razón que llevó a re planificar las tareas que se atrasaron. El no poder culminar la versión final del documento de Triple Play (para el entregable de setiembre), fue preponderante a la hora de decidir la nueva planificación, ya que no se consiguió el material apropiado en tiempo y forma.

Para el primer entregable se tenían implementados dos tipos de servidores[5]: **Servidor Streaming** que generaba flujos que debían ser enviados a velocidad constante a lo largo del enlace (como son flujos de Voz y Video) y **Servidor Elástico** que generaba flujos que simulaban el comportamiento del tráfico de internet (flujos que se enviaban a rate varible, según era la capacidad disponible en el enlace).

En una segunda instancia, al optimizar el prototipo y extenderlo a una topología arbitraria (más de un enlace), se llegó a la conclusión de que no servía la asignación de la tasa de transmisión para los flujos elásticos implementada anteriormente. Es por esto que se decidió implementar un algoritmo de asignación de recursos, se eligió: **Max-Min Fairness**[6]. Una vez implementado dicho algoritmo se realizaron simulaciones para probar su funcionamiento las cuales dieron resultados satisfactorios. Además se evaluaron los tiempos de simulación y estos dieron mejores de los esperados (Ver Apéndice E).

Un parámetro importante que indica el desempeño de un simulador es su escalabilidad, en particular el tiempo que toma simular una topología comple-

[2]Fórmula que permite determinar la probabilidad de bloqueo para un sistema basado en pérdidas donde los arribos tengan distribución Poisson
[3]Flujos Elásticos - flujos que **no** necesitan un rate fijo para ser enviados
[4]capacidad libre en el enlace = capacidad total-capacidad ocupada por lo flujos streaming
[5]Servidores: Generadores de flujos
[6]Más adelante en el documento se realizará una descripción del algoritmo y de la forma de implementación

ja, utilizando algoritmos complejos de asignación como los ya mencionados. El hecho de que el simulador fuera realizado en Java, que es un lenguaje basado en máquinas virtuales, podía influir en este aspecto. Sin embargo, como se estudia en el Capítulo 10, los resultados en cuanto a tiempo de ejecución fueron muy buenos.

También para el entregable de Febrero, se realizó una *Interfaz de Usuario* que permitía crear los elementos de la topología de la red a través de un archivo de texto.

Para la entrega final del proyecto se perfeccionó dicha interfaz. Se adjuntan varios ejemplos de posibles archivos de entrada. En el Apéndice F se adjunta un instructivo de la misma y también ejemplos de uso típico.

Por último, se implementó una política de tarifación indicada por el grupo PDT, junto con distintos perfiles de usuarios. Dicha política se realizó para el caso de flujos streaming.

Para la entrega final, se perfeccionó el simulador. Hasta el segundo entregable, los servidores creaban flujos de manera aleatoria (sorteaban el tamaño para el caso de flujos elásticos, la duración para el caso de flujos streaming, etc). Los clientes, simplemente recibían dichos flujos y actualizaban contadores. En la versión final del simulador, los clientes son los encargados de solicitar los distintos flujos a la red. Además es posible definir diferentes caminos o rutas a seguir por los flujos, y aplicar políticas de tarifación complejas. Existen elementos llamados Gestores que se encargan controlar dichas políticas.

El simulador final cuenta con una *Interfaz Gráfica* para que el uso del mismo sea más amigable al usuario.

Capítulo 4

Descripción del Software

4.1. Frontera

Consideramos como sistema al **simulador**, eso incluye: los enlaces, los servidores, los nodos, los clientes, la lista de eventos, etc.

El **usuario del simulador** es el actor que interactúa con dicho sistema.

El simulador contendrá clases que definen las diferentes componentes de la topología y mantendrá una lista de eventos, dicha lista, es la que lleva el orden cronológico de los hechos que ocurren en el sistema e inicia la ejecución de los métodos apropiados para actualizar el estado de la red.

4.2. Requerimientos del Sistema

Requerimientos Funcionales:

- crear una topología a simular (enlaces, nodos, Clientes Streaming[1], Clientes Elásticos[2], servidores (tanto generadores de flujos elásticos como streaming), rutas, etc.
- manejar una lista de eventos ordenada cronológicamente
- crear y destruir eventos que serán incluidos en dicha **lista de eventos**; Al recorrer la lista, estos descencadenarán acciones que darán paso a la evolución de la simulación
- crear, enviar y recibir flujos de distinto tipo (tipo Elásticos o tipo Streaming, estos serán detallados cuando se expliquen las clases correspondientes)
- aplicar políticas de tarifación para el caso de Flujos Streaming
- recalcular las tasas de envío de flujos del tipo FlujoElástico al ingreso o egreso de un flujo al enlace aplicando el algoritmo de Max-Min Fairness
- guardar estadísticas

[1]Clientes que solicitan Flujos Streaming
[2]Clientes que solicitan Flujos Elásticos

4.3. Actores

El único actor es el **usuario del simulador**. La persona que a partir de la Interfaz Gráfica o de la Interfaz de Usuario crea la topología y hace correr la simulación.

4.4. Casos de Uso

Existen dos casos de uso bien diferenciados.

4.4.1. Caso de Uso: Configuración del simulador

Actor: Usuario del simulador

Escenario Principal: El **usuario** configura en la Interfaz Gráfica (o en la Interfaz de Usuario (archivo de texto)) la topología de la red. Además setea los parámetros de la simulación como: capacidades de enlaces, medias de arribo de peticiones de flujos, medias de oferta, etc. También, el usuario es el encargado de configurar el tiempo en que un cliente comience a solicitar flujos. Luego de esto la simulación está pronta para correr.

4.4.2. Caso de Uso: Correr Simulación

Actor: Usuario del simulador.

Escenario Principal: El **usuario** da comienzo a la simulación presionando el botón de **Start** de la Interfaz Gráfica. Corre la simulación
Escenario Alternativo: El **usuario** da comienzo a la simulación presionando el botón de **Start** de la Interfaz Gráfica. Si existe algún error en la configuración de la topología o en la asignación de algún parámetros, la simulación no se ejecutará enviando un mensaje al usuario.

Para explicar en más detalle el segundo Caso de Uso, se dividió en "sub Casos de Uso", estos casos de usos los determinan los distintos tipos de eventos que intervienen en el simulador. Los Sub Casos de Usos son:

- Solicitar Elástico (Indica que un cliente solicita un flujo elástico)
- Comienza Elástico (Indica el comienzo del envío de un flujo elástico)
- Termina Elástico (Indica el fin del envío de un flujo elástico)
- Solicitar Streaming (Indica que un cliente solicita un flujo streaming)
- Comienza Streaming (Indica el comienzo del envío de un flujo streaming)
- Termina Streaming (Indica el fin del envío de un flujo streaming)
- Llamada a Controlador de Gestores (Indica el turno de realizar los remates)
- Oferta insuficiente para Streaming (En el caso de no aceptarse una solicitud de algún flujo streaming, se le devuelve al cliente un evento llamado igual que el caso de uso.)

- Fin (Termina la simulación)

Al correr la simulación, se va recorriendo la lista de eventos. Cada evento llama a su agente destino para que realice la acción correspondiente dependiendo del evento en cuestión. Esto se va a repetir hasta la llegada de un evento de la clase **EventoDeFin**, en ese momento se termina la simulación.

Sub Caso de Uso: Solicitar Elástico

Actor: Usuario

Escenario Principal: El caso de uso comienza cuando al recorrer la lista de eventos se encuentra un Evento de la clase **SolicitarElastico**[3]. Las instancias pertenecientes a esta clase tienen asociados los parámetros del flujo a solicitar. En este caso al tratarse de un flujo que simula el tráfico de Internet, el criterio es, dejar pasar siempre este tipo de flujos asignándoles el rate dado por el algoritmo de **Max- Min Fairness**. Al turno de este evento, éste llama a su agente destino para que genere el flujo solicitado por el cliente. El servidor elástico correspondiente, además de crear el flujo, crea el evento correspondiente de la clase **ComienzaElastico**.

Sub Caso de Uso: Comienza Elástico

Actor: Usuario

Escenario Principal: El caso de uso comienza cuando al recorrer la lista de eventos se encuentra un Evento de la clase **ComienzaElastico**[4]. Al turno de este evento, éste llama a su agente destino (Cliente Elastico)para que vuelva a solicitar un flujo[5]. Además, éste evento, agrega el flujo en la lista de flujos que se están enviando y crea el evento **TerminaElastico** correspondiente. Luego de esto, llama al algoritmo de Max - Min Fairnes para que recalcule las tasas de envío de los demás flujos del tipo **FlujoElastico** que se están enviando y por lo tanto deberá invocar a que se actualicen los respectivos eventos del tipo **TerminaElastico**.

Sub Caso de Uso: Termina Elástico

Actor: Usuario

Escenario Principal: El caso de uso comienza cuando al recorrer la lista de eventos se encuentra un Evento de la clase **TerminaElastico**. Al turno de este evento, éste llama a su agente destino (elemento de la clase ClienteElastico) para que obtenga el flujo desde la lista de flujos que se están enviando e incremente el contador de flujos recibidos. Luego se llama al algoritmo de asignación de tasas

[3]El evento **SolicitarElastico** es previamente creado por un cliente perteneciente a la clase **ClienteElastico**(su agente origen) y tiene como agente destino a un Servidor correspondiente a la clase **ServidorElastico**

[4]El evento **ComienzaElastico** es previamente creado por un servidor perteneciente a la clase **ServidorElastico**(su agente origen) como se explicó anteriormente, y tiene como agente destino al cliente que solicitó el flujo

[5]El cliente crea un nuevo evento del tipo **SolicitarElastico** indicando las características del flujo que desea

(Max-Min Fairness) y se actualizan los eventos del tipo **TerminaElastico** presentes en la lista de eventos.

Sub Caso de Uso: Solicitar Streaming

Actor: Usuario

Escenario Principal: El caso de uso comienza cuando al recorrer la lista de eventos se encuentra un Evento de la clase **SolicitarStreaming**[6]. Al turno de este evento, éste, según su ruta y rate asociado, busca el gestor que le corresponde y se ingresa a la lista de peticiones del mismo. Al ingresar a la lista inmediatamente queda ordenado según su oferta.

Sub Caso de Uso: Comienza Streaming

Actor: Usuario

Escenario Principal: El caso de uso comienza cuando al recorrer la lista de eventos se encuentra un Evento de la clase **ComienzaStreaming**[7]. Al turno de este evento, éste actualiza la capacidad ocupada por los flujos streaming en los enlaces involucrados en la ruta del flujo en cuestión y se llama a que se ejecute el algoritmo de Max - Min Fairnes. Luego da paso a la actualización de los eventos **TerminaElastico** existentes. Agrega el flujo en la lista de flujos que se están enviando y crea el evento **TerminaStreaming** correspondiente.

Sub Caso de Uso: Termina Streaming

Actor: Usuario

Escenario Principal: El caso de uso comienza cuando al recorrer la lista de eventos se encuentra un Evento de la clase **TerminaStreaming**. Al turno de este evento, éste llama a su agente destino (elemento de la clase ClienteStreaming) para que obtenga el flujo desde la lista de flujos que se están enviando, incremente la cantidad de flujos recibidos y realice una nueva petición de un flujo de este tipo. Antes de llamar al algoritmo de asignación de tasas (Max-Min Fairness) se actualizan las capacidades ocupadas por los flujos streaming en los enlaces pertenecientes a la ruta del flujo que termina su envío. Por último se actualizan los eventos del tipo **TerminaElastico** presentes en la lista de eventos.

[6]El evento **SolicitarStreaming** es previamente creado por un cliente perteneciente a la clase **ClienteStreaming**(su agente origen) y tiene como agente destino a un Servidor correspondiente a la clase **ServidorStreaming**

[7]El evento **ComienzaStreaming** es previamente creado por un servidor perteneciente a la clase **ServidorStreaming**(su agente origen) como se explicó anteriormente, y tiene como agente destino al cliente que solicitó el flujo y ganó el remate

Sub Caso de Uso: Llamada a controlador de gestores

Actor: Usuario

Escenario Principal: El caso de uso comienza cuando al recorrer la lista de eventos se encuentra un evento de la clase **LlamadaAControladorDeGestores**[8]. Al turno de este evento, éste llama al controlador de gestores. El controlador lo primero que hace es crear una nueva instancia de la clase **LlamadaAControladorDeGestores** y luego le indica a los gestores de la red que es hora de rematar. Una vez realizada la subasta cada gestor da la orden de comenzar el envío de los flujos streaming correspondientes a las ofertas aceptadas[9], también se crean los correspondientes eventos del tipo **OfertaInsuficienteParaStreaming** dirigidos a los clientes cuyas ofertas no fueron aceptadas en el remate.

Sub Caso de Uso: Oferta insuficiente para Streaming

Actor: Usuario

Escenario Principal: El caso de uso se da a lugar, cuando al recorrer la lista de eventos se encuentra un evento del tipo **OfertaInsuficienteParaStreaming**. Éste evento llama a su agente origen, un cliente streaming, para que vuelva a ofertar, avisándole que su oferta fue rechazada.

Sub Caso de Uso: Fin de la simulación

Actor: Usuario

Escenario Principal: El caso de uso se da a lugar, cuando al recorrer la lista de evento se encuentra un evento correspondiente a la clase **EventoDeFin**, este evento es creado al inicio de la simulación y el usuario es el encargado de setearle el tiempo asociado. Al ocurrir éste evento se da por finalizada la simulación.

[8]Esta llamada al controlador de gestores sólo tiene sentido cuando existen clientes del tipo **ClienteStreaming** activos en la red

[9]Los servidores además de crear los flujos streaming solicitados, crean los respectivos eventos **ComienzaStreaming**

4.5. Diagramas de Interacción

A continuación se presenta la especificación del comportamiento de cada sub Caso de Uso por medio de los DSS. Para todos, se realiza dicho diagrama para el flujo básico.

4.5.1. DSS: Solicitar Elástico

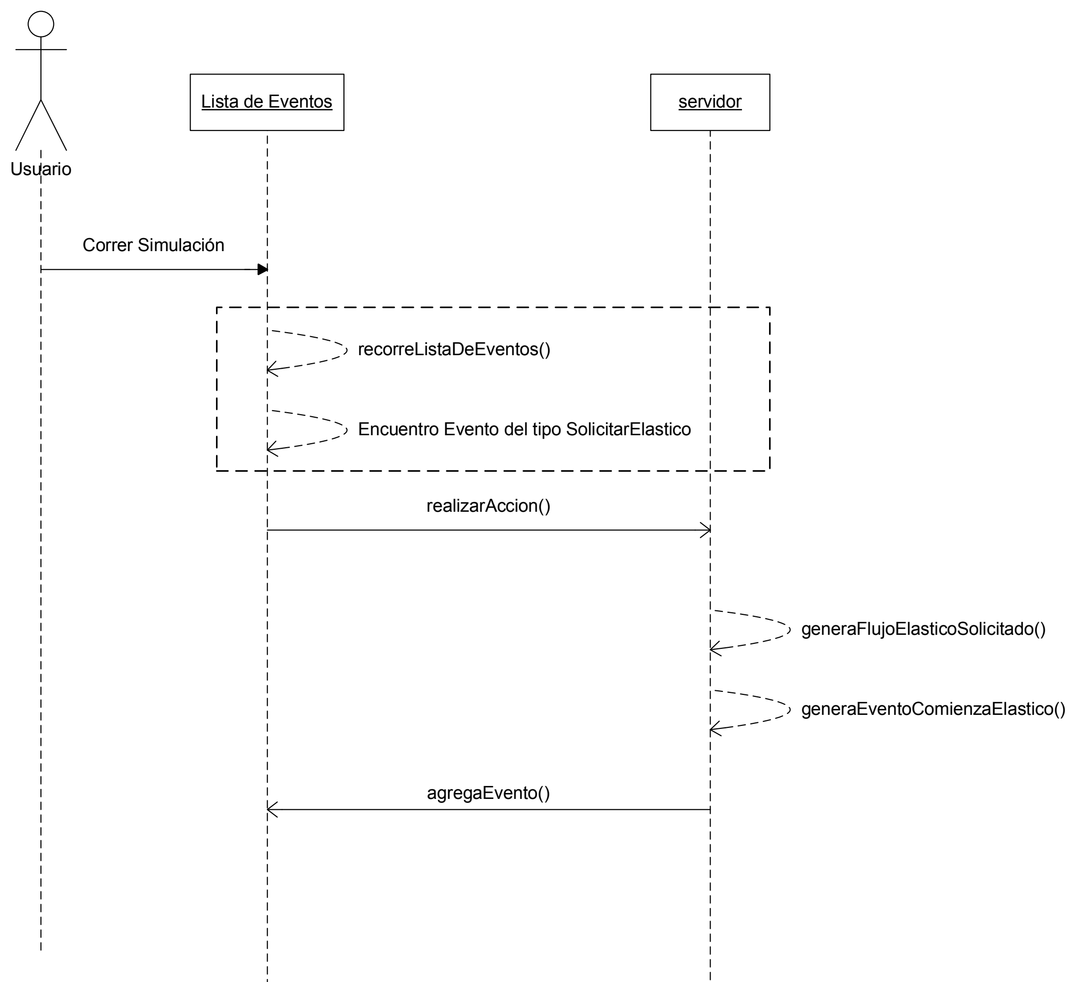

Figura 4.1: Solicitar Elástico

El caso de uso corresponde al turno de un evento de petición de un flujo elástico a la red.

Obs: El elemento **servidor** es una instancia de la clase **ServidorElastico**.

4.5.2. DSS: Comienza Elástico

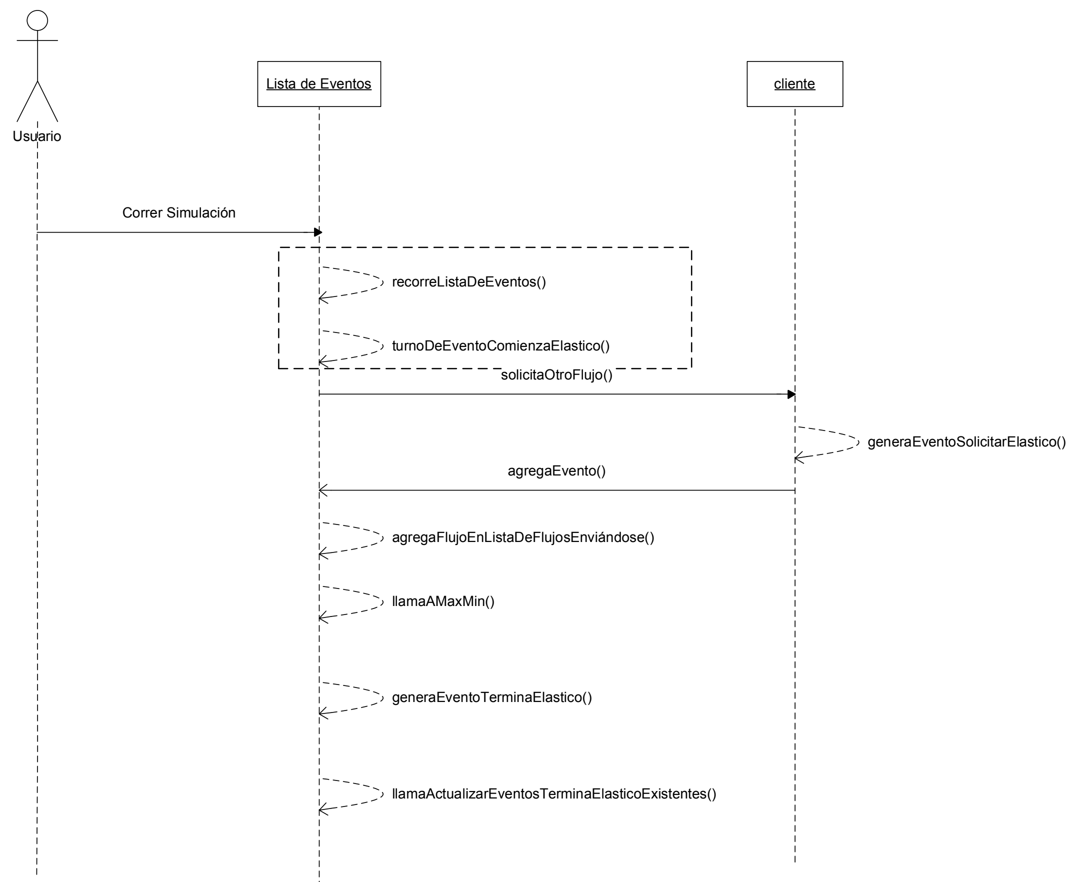

Figura 4.2: Comienza Elástico

El caso de uso corresponde a un evento que indica el comienzo de la transmisión de un flujo elástico. El cliente mostrado es un cliente que solicita flujos elásticos.

4.5.3. DSS: Termina Elástico

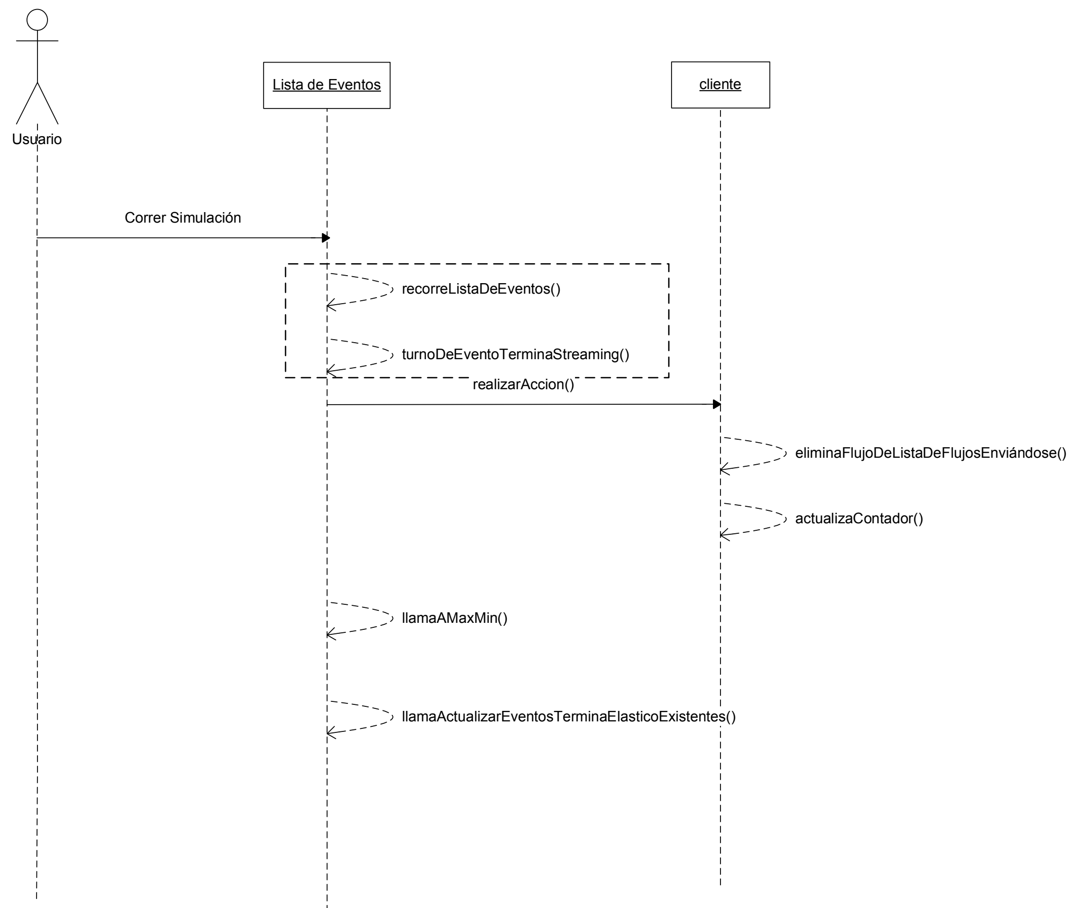

Figura 4.3: Termina Elástico

4.5.4. DSS: Solicitar Streaming

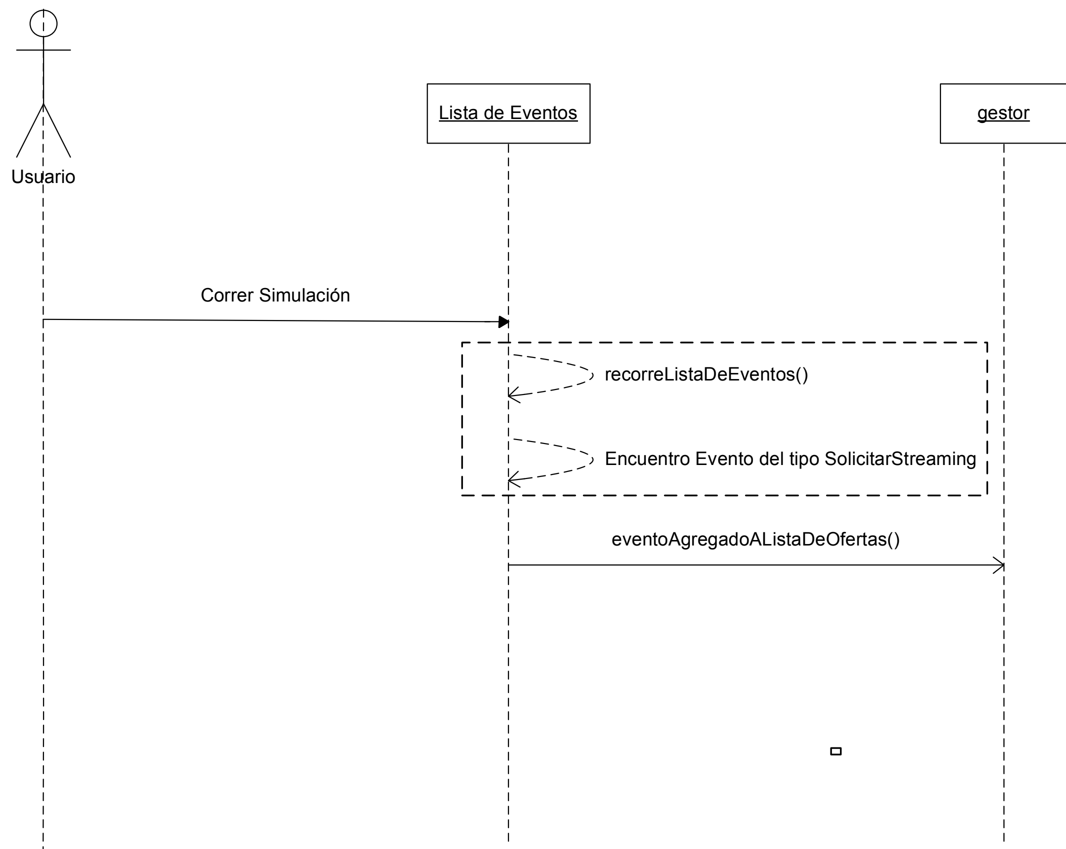

Figura 4.4: Solicitar Streaming

4.5.5. DSS: Comienza Streaming

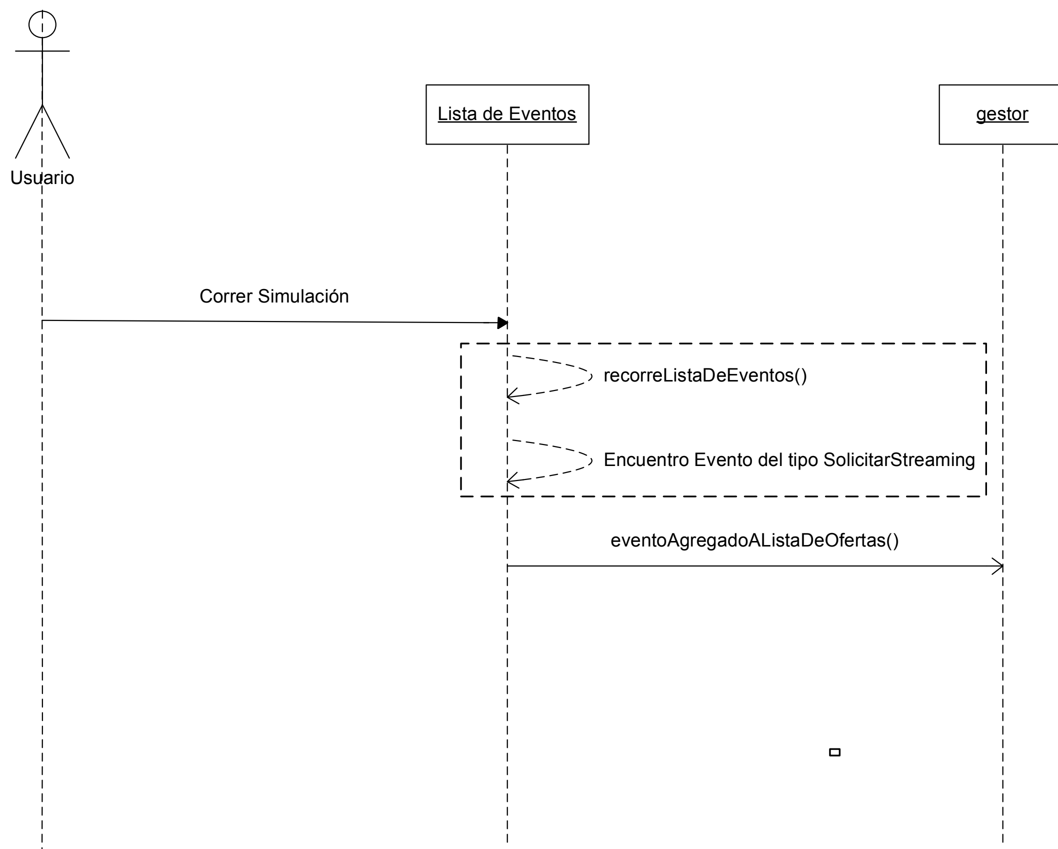

Figura 4.5: Comienza Streaming

Obs: El elemento del diagrama llamado **ruta** representa la ruta del flujo streaming que se comienza a transmitir al ocurrir el evento **ComienzaStreaming**.

4.5.6. DSS: Termina Streaming

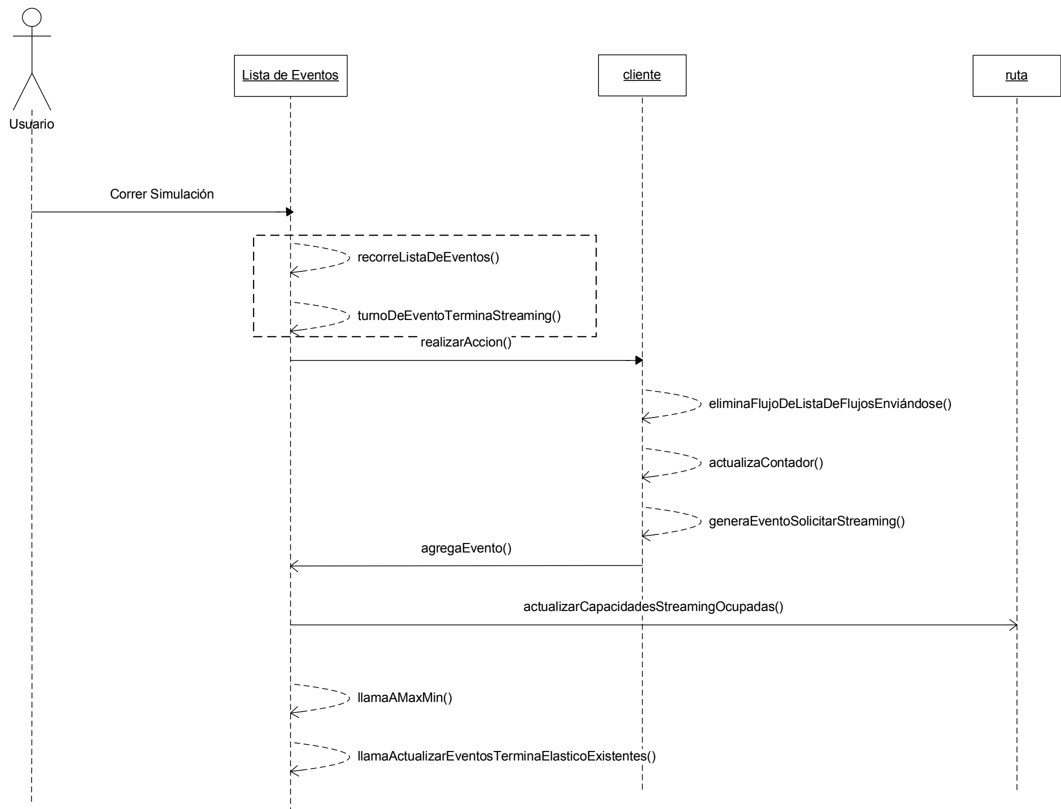

Figura 4.6: Termina Streaming

Obs: El elemento del diagrama llamado **ruta** representa la ruta del flujo streaming que se terminó de enviar asociado al evento **TerminaStreaming**.

4.5.7. DSS: Llamada a controlador de gestores

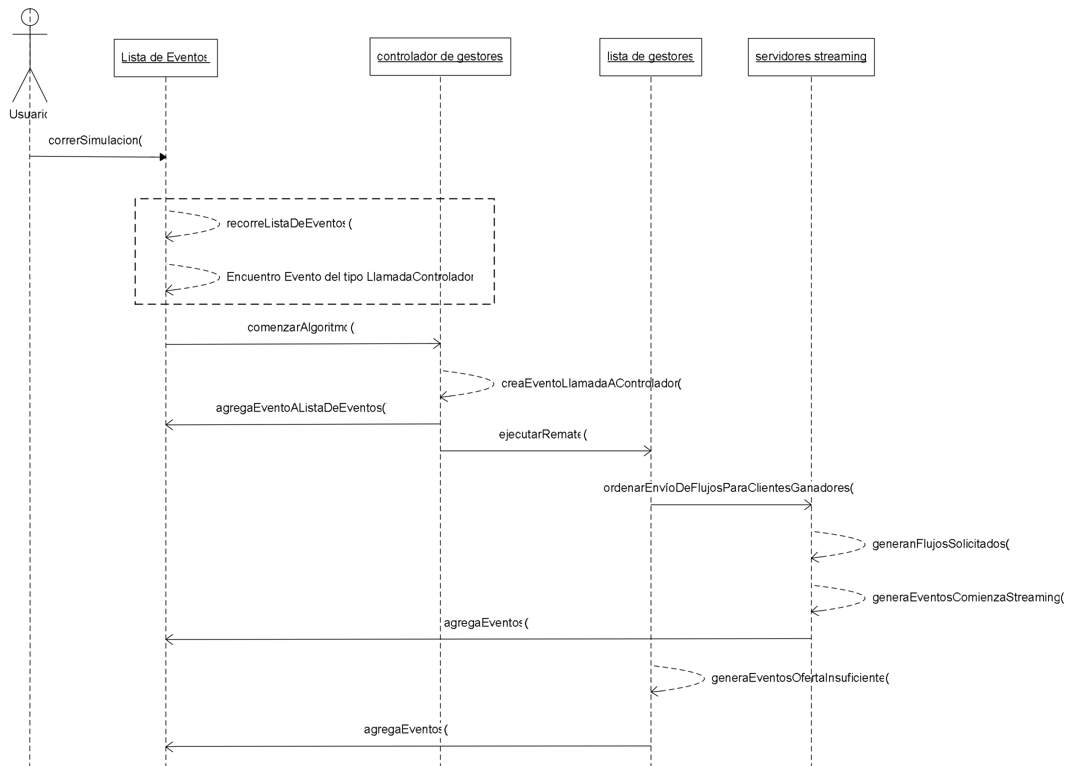

Figura 4.7: Llamada a controlador de gestores

4.5.8. DSS: Oferta insuficiente para streaming

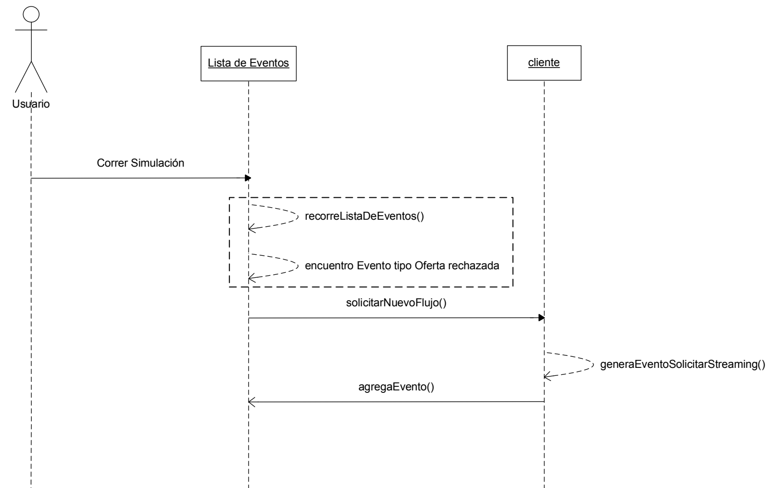

Figura 4.8: Oferta insuficiente para streaming

4.5.9. DSS: Fin de la simulación

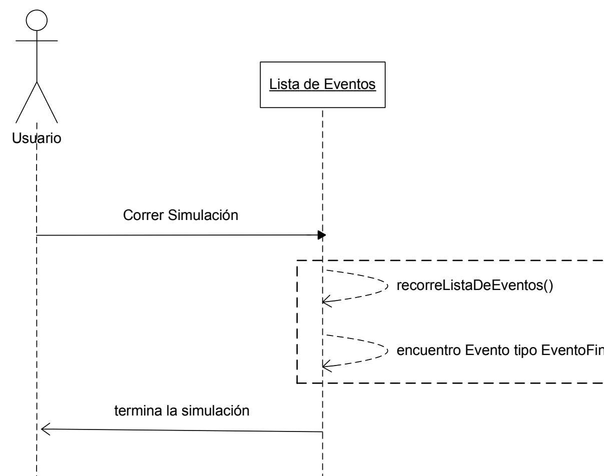

Figura 4.9: Fin de la simulación

4.6. Descripción y diagramas de las clases implementadas

A continuación se muestran los diagramas de algunas relaciones entre las clases implementadas.
Se omiten en el diagrama los método get(), set() y los constructores.

En capítulo **Descripción de las clases implementadas** del Apéndice B se detallan todas las clases en profundidad.

4.6.1. Package Topología

Para implementar la topología de la red se crearon dos clases: **Enlace** y **Nodo**. Las instancias de la clase **Nodo** juegan el papel de elementos de interconexión entre las instancias de la clase **Enlace**. Dos enlaces estarán conectados si tiene un nodo en común. En la representación de la clase **Enlace** se observan dos atributos que son instancias de la clase **Nodo**, éstos son `nodoInicio` y `nodoFin`.

Las instancias de la clase **Enlace** tienen asociadas una `capacidad` (b/s), dicho campo es el limitante a la hora de asignar la tasa de transmisión a los flujos elásticos y también es preponderante para decidir la entrada o no de los flujos streaming a la red.

Las clases mencionadas anteriormente heredan de la clase abstracta **Agente**[10]. También heredan de dicha clase, las clases: **ServidorStreaming**, **ServidorElastico**, **ClienteStreaming** y **ClienteElastico**.

Las instancias de las clases **ServidorElastico** son las encargadas de generar instancias de la clase **FlujoElastico**, representando el tráfico de datos. Dichos flujos son solicitados a la red por algún objeto de la clase **ClienteElastico**.

Las instancias de la clase **ServidorStreaming** y **ClienteStreaming** se comportan análogamente a las instancias de las clases **ServidorElastico** y **ClienteElastico** respectivamente.

Otras clases que resultaron conveniente incluirlas en el paquete Topología son las clases: **Ruta** (representando los posibles caminos en la red), **Gestor** y **ControladorDeGestores**. Las instancias de éstas dos últimas clases, son las encargadas de ejecutar la política de tarifación implementada.

[10] Agente: elemento de la red que maneja eventos

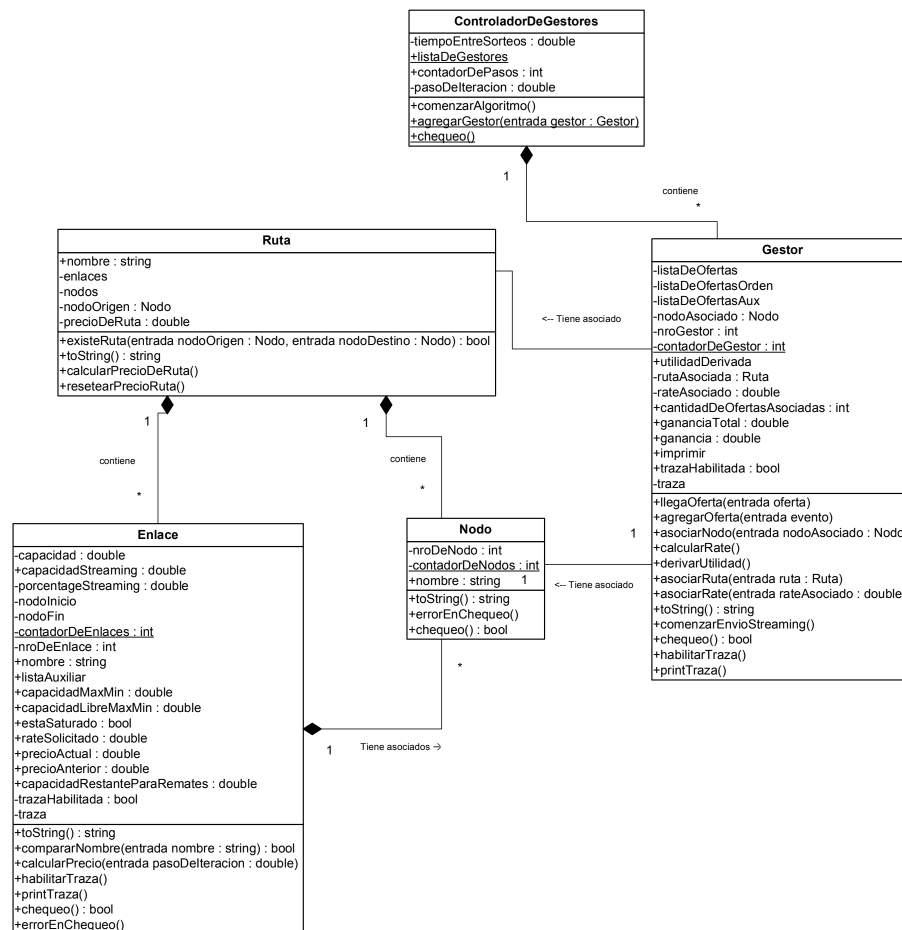

Figura 4.10: Relaciones importantes entre clases del paquete topología

4.6.2. Package Eventos

La herramienta de simulación está basada en eventos discretos, para ello se implementaron varias clases cuyas instancias representan los posibles eventos que desencadenan la simulación.

Dichas clases heredan de una clase abstracta denominada **Evento**.
La clase abstracta, tiene un método abstracto llamado **realizarAccion()** el cual, al ser invocado desencadena una serie de acciones representativas de cada tipo de evento (Ver Casos de Uso).

En este paquete también se incluyó la clase **ListaDeEventos**.

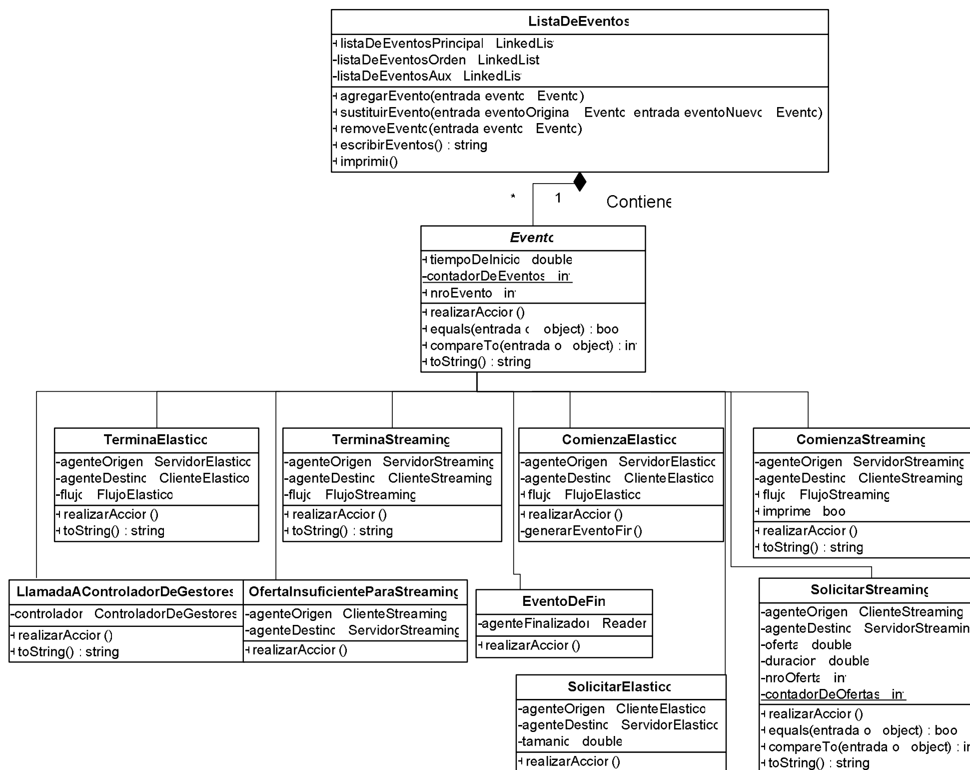

Figura 4.11: Clases del paquete Eventos

4.6.3. Package Exceptions

Dicho paquete contiene dos clases que implementan dos tipos de excepciones a considerar.

RutaNoContinua

Esta excepción ocurre cuando al crear una ruta se indica un conjunto de enlaces que no forma un camino continuo, es verificado en el constructor de la clase **Ruta** según nodos asociados a cada instancia de la clase **Enlace**.

DestinoNoAlcanzable

Esta excepción ocurre al momento de asignarle la ruta a un flujo si no existe ninguna ruta que vaya del inicio al fin deseado.

4.6.4. Package Flujos

Este paquete contiene las clases que representan los distintos flujos en la red. Tratando de emular el tráfico en una red Triple Play se crearon dos clases que representan los dos tipos de flujos, los flujos elásticos y los flujos streaming.

Las instancias de estas clases guardan las características de los flujos que están en curso por la red y permiten calcular los recursos asignados a cada uno.

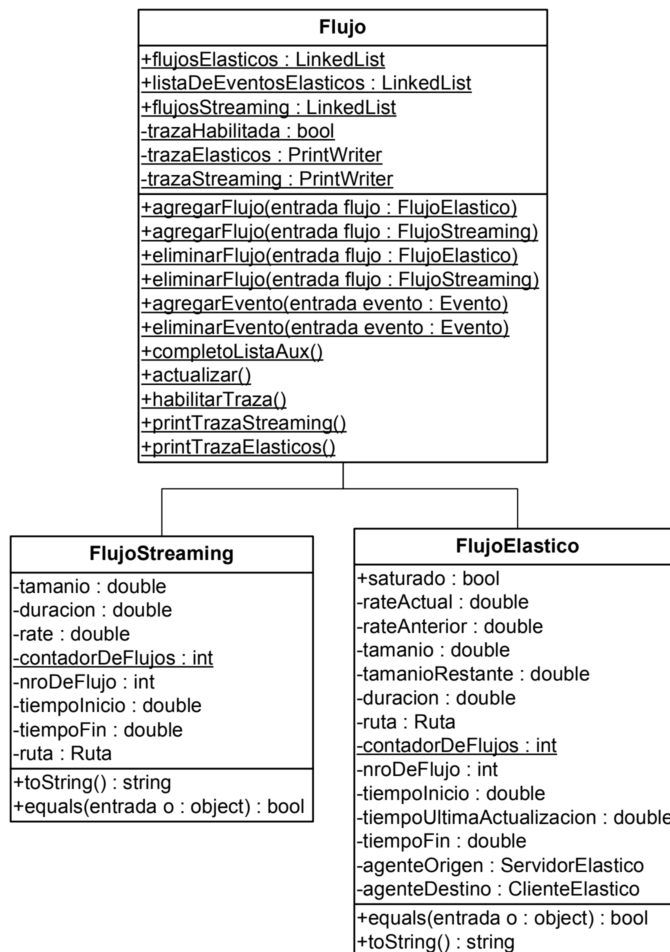

Figura 4.12: Clases del paquete Flujos

4.6.5. Package Main

En el paquete se incluyen las clases:

- **Principal:** Contiene el método *main(String[] args)* y tiene como atributo una instancia de la **ListaDeEventos** donde se alojan todos los eventos que desencadenan la simulación.

- **Reader:** Clase encargada de interpretar el archivo de texto configurado por el usuario antes de correr la simulación.

- **GeneradorAleatorio:** Las instancias de las clases **ClienteElastico** y **ClienteStreaming** hacen referencia a dicha clase para obtener números aleatorios con distribución exponencial.

- **MaxMinFairness:** Clase que implementa el algoritmo de asignación de recursos Max - Min Fairness. En el capítulo siguiente se explica dicho algoritmo en detalle.

4.7. Obtención de estadísticas

Existen clases cuyas instancias se encargan de almacenar sus estadísticas en los momentos en que la red cambia de estado.

Dichas clases son: Gestor, Enlace, Flujo y ClienteElastico. La habilitación del trazado de los datos de cada uno de los elementos de la red, debe ser dada por el usuario al momento de definir el archivo de texto de entrada (en el caso de la interfaz de usuario) o al momento de crear la topología usando la Interfaz Gráfica. Por más información, ver detalles en el manual correspondiente.

Las instancias de la clase **Gestor** expulsan un archivo de texto con el mismo nombre de la instancia, en él se guardan, al ocurrir cada remate, las siguientes trazas: **Tiempo de Remate**, **Ganancia de Remate**, **Ganancia Total**, **Ofertas aceptadas en el correspondiente remate**.

El formato para las trazas de las instancias pertenecientes a la clase Enlace es: **tiempo**, **rate streaming** y **rate elástico**. Los últimos dos campos significan la capacidad destinada para flujos streaming y flujos elásticos respectivamente.

Los clientes elásticos almacenan el tiempo en que termina de recibir un flujo, el nº identificatorio de dicho flujo y la duración del mismo.

La clase Flujo, crea dos archivos de texto, uno con el nombre "FlujosStreaming.txt"y el otro "FlujosElasticos.txt". Cada vez que se aplica el algoritmo de **Max- Min Fairness** se guarda en cada archivo el tiempo, el número del flujo enviándose y el rate asignado por la red a dicho flujo.

Las trazas se guardan automáticamente en una carpeta llamada **Trazas** que debe ser creada por el usuario antes de la simulación [11].

[11] Ver Manual de Usuario en el Apéndice F

Parte III

Fundamentos teóricos utilizados

Capítulo 5

Algoritmo de Max - Min Fairness

5.1. Introducción

El determinar cuanto tráfico de cada flujo debe ser admitido por la red satisfaciendo los requerimientos de alta utilización de la misma y garantizando justicia a los usuarios, es uno de los retos más grandes del diseño de las redes de telecomunicaciones de hoy en día.

¿Qué principios se deben seguir para asignar el ancho de banda entre los distintos flujos presentes en la red, de manera de cumplir algún criterio de justicia?

Una posible respuesta sería utilizar el algoritmo llamado `Max - Min Fairness`. Este algoritmo realiza una asignación justa de los recursos de ancho de banda a los diferentes flujos elásticos que circulan por la red. En una red real esto se logra mediante políticas de scheduling en los routers, como se propone en [13]. A su vez, constituye una buena aproximación de la asignación de recursos que realiza TCP en la escala de flujos, en una situación en que los Round - Trip Times (RTT) de los diferentes flujos que intervienen son similares [14].

La idea básica del algoritmo es incrementar lo máximo posible el ancho de banda (rate) de una demanda (flujo), sin que sea a expensas de otra demanda.

Matemáticamente, el objetivo del algoritmo es hallar el vector de asignación de recursos $x* = (x_1*, x_2*, x_3*, ..., x_N*)$ donde N es la cantidad de flujos enviándose en la red. Dicho vector indica cuanto BW se llevará cada demanda.

Para entender en que consiste este algoritmo, se considerará por ejemplo, el siguiente problema:

La red consta de dos enlaces, tal como lo muestra la Figura 5,1. Dichos enlaces son de distinta capacidad, y existen tres posibles rutas[1].

Se supone que en un determinado momento existen en la red tres flujos, flujos elásticos (aquellos que su tasa de transferencia se adapta a lo disponible en la

[1] Se denominará **Ruta 1** a la que solo ocupa el enlace de $1Mbps$, **Ruta 2** pasa por los dos enlaces y por consecuencia la otra ruta será la **Ruta 3**.

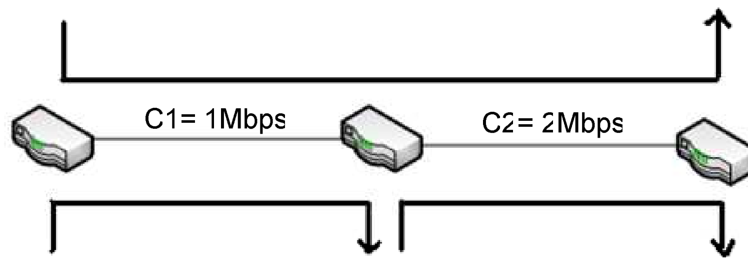

Figura 5.1: Asignación de recursos en una red sencilla

red). Cada uno de los flujos viajando en una ruta distinta.

La idea a la hora de repartir el ancho de banda, es tratar de ser lo más justo posible. Con este criterio, se asignaría $0{,}5Mbps$ a cada flujo, quedando saturado el enlace de $1Mbps$. Pero, otro criterio a cumplir es maximizar la utilización de los recursos de la red. Esto último, con la repartición anterior, no se está cumpliendo, ya que queda $1Mbps$ sin utilizar en el enlace de $2Mbps$. Entonces al flujo que circula por la **Ruta 3** se le asigna una tasa de transmisión de $1{,}5Mbps$, esto no afecta a ninguna de las otras demandas y es más beneficioso tanto para el usuario que haya solicitado el flujo como para la red (ya que va a terminar de ser enviado más rápido).

Es importante observar, que según la asignación anterior, el tráfico en la red otorgado a los flujos presentes en el ejemplo, suma un total de 2.5Mbps. Si sólo consideramos que existen dos flujos, uno en la **Ruta 1** y el otro en la **Ruta 3**, aplicando el mismo criterio, se le asigna 1Mbps y 2Mbps respectivamente (totalizando 3Mbps de tráfico otorgado). Esto se ve reflejado teniendo en cuenta que el flujo que pasa por los dos enlaces necesita recursos de ambos [9].

A continuación se explica como se llevó a cabo la implementación del algoritmo.

5.2. Implementación

La implementación se basó en el algoritmo de **Water Filling**.[11]

En nuestra red, los enlaces tienen capacidades fijas y rutas prefijadas de ante mano.

Básicamente la idea del algoritmo, es incrementar de a *pasos* el rate de cada flujo (tipo FlujoElastico) hasta que se saturan todos los enlaces presentes en la red[2].

El *paso*, denominado en el programa como **minimoViejo**, se determina como:

$$minimoViejo = \min_i \frac{capacidad}{cantidadDeFlujosQuePasanPorEl}$$

[2]Saturar un enlace es análogo a que un balde se llene en el algoritmo de Water Filling

i: recorre los enlaces de la red.

Los flujos considerados, son sólo aquellos que no pertenecen a un enlace que haya saturado en algún paso de la iteración. La capacidad considerada es la **capacidadMaxMin**.

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

PASO 1:

Inicialización en 0 de todos los **rates** de los flujos tipo **FlujoElastico** presentes en la red.

Inicialización en FALSE la variable **saturado** de todos los flujos tipo **FlujoElastico** presentes en la red.

Inicialización de la variable **capacidadMaxMin** de cada enlace como la diferencia entre la capacidad del enlace y la capacidad utilizada por los flujos tipo streaming. La **capacidadMaxMin** se irá actualizando en cada paso de la iteración.

Inicializo el **minimoViejo** y el **delta** en 0. **delta** es el rate final que se le asigna a cada flujo, se va incrementando en **minimoViejo** en cada paso de la iteración.

Se inicializa la variable termino en FALSE.

PASO 2:

Se llama al método *hallarMinimo()* que determina el valor del **minimoViejo**, junto con este valor, almacena el enlace que lo determinó. Además de realizar esto, en el caso de que todos los enlaces estén saturados, o los que quedan no tengan flujos pasando por él, setea en TRUE la variable **termino** para finalizar la iteración.

PASO 3:

En el caso de que **termino** sea FALSE, se da paso al método *asignarRateIntermedio(delta)*, el cual se encarga de setear **rate = delta** a cada flujo que no tenga la variable saturado en TRUE.

Luego de esto se llama al método *actualizarCapacidades()*. Éste se encarga de actualizar la **capacidadMaxMin** para el siguiente paso de la iteración, y además llama a *setearFlujosSaturados()*, Este último método se encargará de setear la variable saturado de aquellos flujos que pertenecen al enlace que saturó en dicho paso de la iteración. Luego vuelvo al paso 2.

La iteración continúa hasta que **termino** sea TRUE.

Obs: Por detalles de los métodos o atributos de las clases mencionadas ir al capítulo correspondiente en el Apéndice B.

Capítulo 6

Políticas de Tarifación

Dado que las leyes de la economía son cada vez más determinantes en las redes convergentes, en el presente capítulo se desarrollarán algunas de las herramientas de la teoría económica, y su relación con los sistemas de control y asignación de recursos en la red.

6.1. Ingeniería vs Economía

A continuación se muestran los servicios de telecomunicaciones brindados actualmente junto con su relación con la economía.

Telefonía Tradicional

La red telefónica tradicional está dimensionada con multiplexado estadístico y se basa en conmutación de circuitos.
La Ingeniería logra proveer a los abonados de los servicios garantizando cierto Grado de Servicio (GoS) y cierta QoS.

Por otro lado la tarifación de este servicio, al ser monopolio, está basada en la recuperación de los costos y a veces con subsidios internos.

Internet

La ingeniería y la economía se encuentran desacopladas a la hora de brindar y cobrar este servicio.

El operador brinda sólo conectividad, qué hacer con ello es problema del usuario.

Por otro lado, las tarifas existentes son planas por el acceso sin garantías de calidad.

Televisión por cable

La tarifación de este servicio es similar al caso de Internet. Se cobran tarifas planas y se ofrece la posibilidad de contratar diferentes paquetes con un pago adicional.

Claramente se tienen para cada servicio su forma de brindarlo y también su forma de cobrarlo.

El objetivo de la convergencia de las redes, es unificar todos los servicios en la misma red y por lo tanto unificar la política de tarifa, esto es lo mismo que decir: "acoplar más ingeniería y economía en las redes".

6.2. Herramientas de la teoría económica

Funciones de utilidad

"Para distribuir una cantidad **C** de un bien divisible entre consumidores heterogéneos, se asigna a cada uno la función de utilidad $U_i(x_i)$. La función anterior, mide en unidades compatibles (por ejmplo: dinero) el valor que el consumidor asigna a la cantidad x_i".

Obs: $U_i(x_i)$ es en general creciente y cóncava.

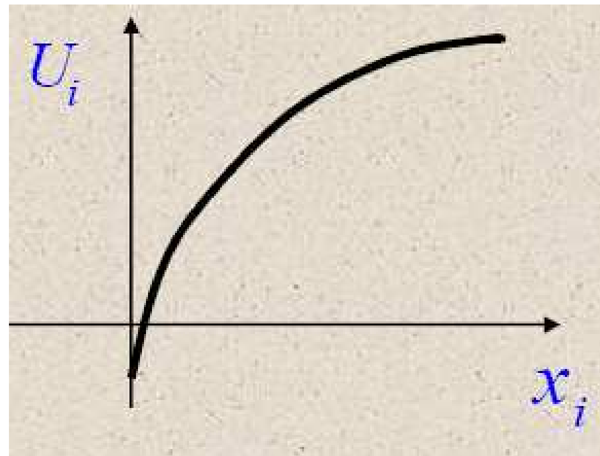

Figura 6.1: Ejemplo de función de utilidad

El punto que asegura la máxima ganancia para el proveedor del bien en cuestión, es aquel que cumple:

$$\text{máx} \sum_i U_i(x_i) \tag{6.1}$$

Sujeto a las restricciones de capacidad:

$$\sum_i x_i \leq C \tag{6.2}$$

Par resolver este problema de forma descentralizada se usan **Mecanismos de precio**.

Cada consumidor resuelve $\max[U_i(x_i) - px_i]$, donde p es un precio unitario.

La solución de lo anterior es: $U'(x_i) = p$.

"Existe un p tal que los x_i resultantes satisfacen la condición $\sum_i x_i < C$ ".

"Un mecanismo dinámico para hallar p es: $\frac{dp}{dt} = \gamma(\sum_i x_i - C)$ ".

El razonamiento anterior es para demandas inélasticas de ancho de banda. Como lo es el tráfico de voz y video en tipo real.

Para el caso de flujos elásticos, como lo son los datos, la idea cambia. [10]

6.3. Implementación

La política de tarifación implementada se ocupa únicamente de los flujos del tipo streaming presentes en la red. El objetivo de la misma es decidir que ofertas (generadas por los clientes) aceptar para garantizar la mayor ganancia al operador.

Haciendo la analogía de lo explicado anteriormente con la implementación de dicha política en el simulador:

Se buscó resolver el problema de forma descentralizada, para ello se creó la clase **Gestor**.

Las instancias de dicha clase gestionarán un **rate** y una única **ruta** definida en la red.

Por lo tanto, el usuario del simulador, deberá definir tantos *Gestores* como rutas en las que se prevea el pasaje de flujos inelásticos. Además si en dichas rutas pueden existir flujos streaming de diferente tasa de transmisión asociada, se deberá definir un gestor por cada tasa de transmisión.

Los elementos de la clase **ClienteStreaming**, tal como se explicó anteriormente en el documento, solicitarán distintos flujos streaming a la red. Cada una de estas ofertas se almacenará en la lista de ofertas del gestor correspondiente.

El gestor ordena la lista de solicitudes de flujo según la oferta (dinero dispuesto a pagar), de mayor a menor y construye su curva de utilidad.

Haciendo la analogía con el caso anterior, el gestor toma el papel del consumidor, ya que por cada gestor tengo una curva de utilidad distinta.

Como se explicó anteriormente, la clase **ControladorDeGestores** crea instancias de la clase **LlamadaAControladorDeGestores**. Esta última representa un tipo de los posibles eventos presentes en el simulador, que tiene como fin pedirle a la instancia de **ControladorDeGestores** que ejecute la política de tarifación.

Al ejecutarse, se realizan los siguientes pasos:

PASO 1:

Instancia de Inicialización.
Se inicializan los precios de los enlaces y los precios de las rutas en 0. Para cada enlace, se setea la **capacidadRestanteParaRemates** en un número muy grande. En este paso del algoritmo, se le pide a todos los gestores que obtengan la derivada de su curva de utilidad.
La función de utilidad de un gestor cualquiera es de la forma:

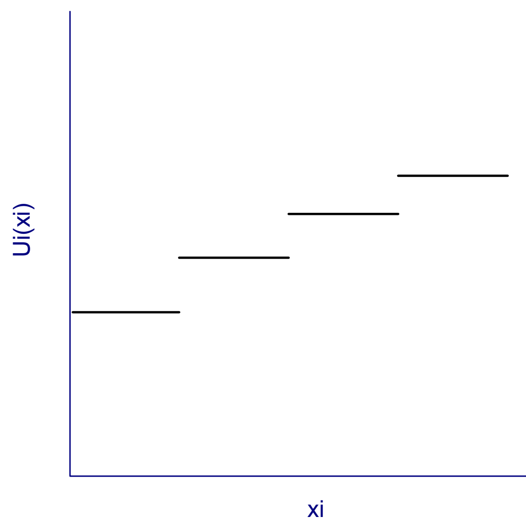

Figura 6.2: Ejemplo de función de utilidad de un Gestor de la red

El primer escalón corresponde a la mayor oferta recibida, el segundo a la suma de la mayor oferta y la que sigue en valor, y así sucesivamente.
Los valores de cambio en el eje de las abscisas son equiespaciados ya que cada gestor maneja una única tasa de transmisión.
Claramente se observa que dicha función no cumple las condiciones de validez del algoritmo. Para ello se utiliza como función utilidad a la siguiente función continua y derivable a trozos (curva roja). Dicha función cumple las condiciones de validez en cada tramo.

A partir de la nueva curva, es posible que cada gestor calcule su derivada, resultando una función escalonada decreciente.

También, en esta etapa de inicialización se resetea la cantidad de ofertas aceptadas por cada gestor.

Antes de comenzar con la iteración se inicializa la variable **finalizador** en FALSE.

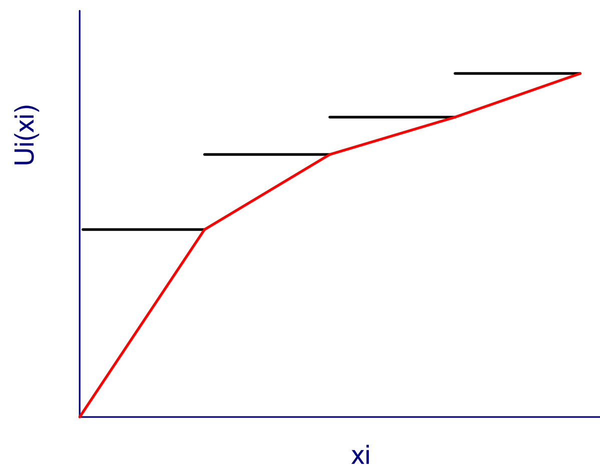

Figura 6.3: Función utilidad de un Gestor de la red

PASO 2:

Se setea la variable **rateSolicitado** de cada enlace en 0, dicho campo hace referencia a la capacidad pedida al enlace en cada paso de la iteración.

PASO 3:

A partir de los precios de las rutas, cada gestor, según la ruta que le corresponda y en función de la curva de su utilidad derivada decide cuántas ofertas aceptar.
En la primer iteración, al estar los precios inicializados en 0 se aceptarán todas las ofertas.
Una vez que el gestor encontró la cantidad de ofertas aceptadas, **agrega** a la variable **rateSolicitado** de cada enlace de la ruta gestionada, la capacidad resultante de permitir la transmisión de los flujos aceptados (**Sumando y no sustituyendo el valor de rateSolicitado** se soluciona el hecho de que dos gestores manejen rutas que tengan enlaces en común).

PASO 4:

En esta etapa del algoritmo, se le pide al enlace que calcule su precio. Ésto se realiza a partir de la variable **rateSolicitado**.
Para calcular el precio para dicha capacidad pedida, se realiza la discretización de la ecuación $\frac{dp}{dt} = \gamma(\sum_i x_i - C)$, obteniendo:

$$p(k+1) = p(k) + h\frac{(rateSolicitado - capacidadStreaming)}{capacidadStreaming}$$

En la fórmula anterior, h representa un pequeño factor de ganancia (es definido por el usuario al momento de la configuración de la simulación). Este factor define la velocidad de convergencia del algoritmo, cuanto menor sea este valor, mas

demorara en converger.

El cálculo del precio de cada enlace se realiza en el método de la clase **Enlace** denominado *calcularPrecio()*.

Un vez ejecutado este procedimiento, se controla el signo de la variable **CapacidadRestanteParaRemates**. Si en alguno de los enlaces dicha variable es positiva el algoritmo continuará, en caso de que todos tengan dicho valor negativo, el algoritmo se terminará seteando **finalizador** en TRUE.

Esta última estrategia es la solución encontrada para lograr **siempre** la convergencia del algoritmo. Con ello se logra llegar a una solución, pero que muchas veces no es la óptima, pero dicha solución se encuentra en la región factible.

PASO 5:

Con los precios de los enlaces cada ruta determina el suyo. Dicho valor es la suma de los precios de todos los enlaces que conforman la misma.

Luego se vuelve al *Paso 2*, siempre que **finalizador** esté en FALSE.

Obs: Por detalles de los métodos o atributos de las clases mencionadas ir al Apéndice B.

Parte IV

Descripción de las simulaciones

Capítulo 7

Introducción

En ésta parte del documento se pondrá a prueba el simulador. Para ello se eligieron algunos ejemplos representativos, y en base a estos, se intentará mostrar y evaluar los resultados del mismo en distintas simulaciones.

En las primeras secciones se mostrarán simulaciones únicamente con flujos **streaming** presentes en la red. Se realizaron pruebas con topologías varias: un enlace, dos enlaces y topologías constituidas por ‟muchos″enlaces.
Dichas simulaciones se realizaron con el fin de probar y mostrar el funcionamiento de la política de tarifación implementada: ‟Remates″.

También se realizaron simulaciones sólo con flujos **elásticos** transmitidos en la red con el fin de estudiar la performance del algoritmo Max- Min Fairness.

Por último se muestran simulaciones con los dos tipos de flujos circulando por la red y también un ejemplo donde se evalúa la performance del software al exigir una simulación extensa y sobre una topología más compleja..

Para todas las simulaciones, se adjuntan en el CD: los archivos de entrada, y las trazas obtenidas de cada uno de los elementos.

Capítulo 8

Simulaciones con flujos streaming

8.1. Simulaciones con topologías formadas únicamente por un enlace

8.1.1. Simulación 1

La primer simulación consiste en una topología formada por un enlace y dos nodos. En ella existe un servidor del tipo streaming y diez clientes (instancias de la clase ClienteStreaming) que solicitan flujos a dicho servidor. Como se observa en la Figura 8.1, el servidor está asociado al **Nodo 1** y los clientes al **Nodo 2**.

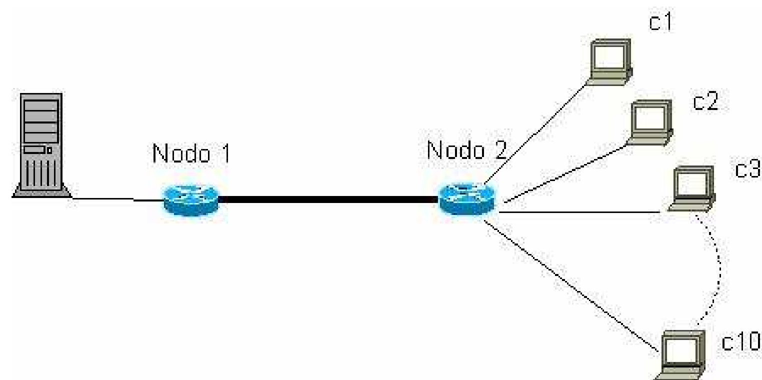

Figura 8.1: Representación de la topología simulada

En el caso de la simulación, los clientes tienen los mismos parámetros salvo la **media de oferta**.
Al existir clientes que solicitan flujos streaming a la red, se deben crear instancias de la clase **Gestor**. En este caso, existe una única ruta, la que va del **Nodo 1** al **Nodo 2** y como todos los clientes tienen asociados el mismo rate, es necesario crear una única instancia de la clase **Gestor**.

El archivo de entrada correspondiente a ésta simulación se encuentra en la carpeta: `Simulaciones\FlujosStreaming\Un Enlace\Simulacion 1\`.

También en dicho directorio se encuentran los datos obtenidos al correr la simulación.

Datos obtenidos

El tiempo de ejecución de la simulación propuesta fue de $485mseg$.
A continuación se muestra un gráfico en el que se representan las ganancias en cada remate.

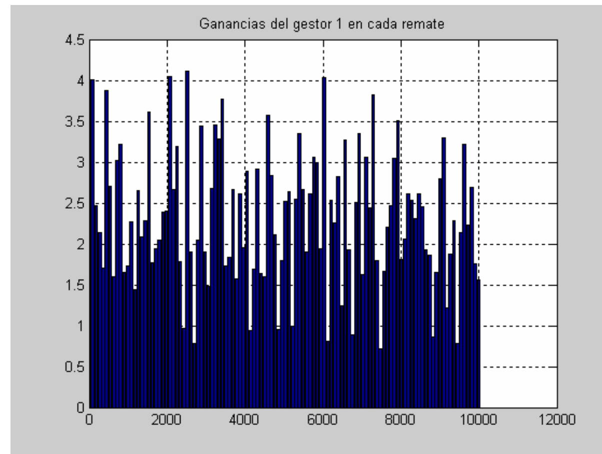

Figura 8.2: Ganancias por remate, media de duración de los flujos 100

En la simulación anterior la ganancia total fue de 257,53. El tiempo entre remates fue de 90 y la media de la duración de los flujos fue 100 (estos dos últimos parámetros son fijados en el archivo de texto que se toma como datos de entrada).

El resultado obtenido al aplicar el algoritmo es consistente con el esperado.

Lo anterior se explicará tomando como referencia los dos primeros remates:

oferta	duración
0,8852	164,57
0,8501	7,11
0,7726	70,21
0,7626	62,44
0,7321	210,44
0,6375	5,67
0,5117	124,07
0,4350	110,14
0,3002	75,319
0,0461	33,27

Cuadro 8.1: Solicitudes en el primer remate

Al aplicar el algoritmo por primera vez, siendo la capacidad del enlace $500b/s$

y los flujos solicitados son de rate $100b/s$, se aceptaron las 5 mejores ofertas dando una ganancia de 4,002858498123234. Ver cuadro 8.1 .

oferta	duración
0,9920	184,39
0,7740	69,00
0,7026	9,86
0,6415	31,79
0,5722	119,68
0,4647	10,83
0,1779	69,14
0,1268	91,30

Cuadro 8.2: Solicitudes en el segundo remate

Como se observa en cuadro 8.2, hay 8 ofertas, esto se debe a que se tomó como criterio, que los clientes vuelven a ofertar en el momento de recibir la noticia de que su oferta fue rechazada, o al momento de terminar de recibir el flujo solicitado.
En el caso del ejemplo, quienes realizaron las 8 ofertas son los cinco clientes cuyas ofertas fueron rechazadas y los tres que recibieron el flujo pedido antes del segundo remate (estos últimos corresponden a las ofertas 2, 3 y 4 del primer remate).

Al momento del segundo remate, existen dos flujos enviándose en la red (flujos ganadores del primer remate asociados a las ofertas 1 y 5), dichos flujos ocupan $200b/s$ del enlace. Por ello en el segundo remate sólo se aceptarán las tres mejores ofertas. Dando una ganancia de 2,468748677937593.

Los valores correspondientes a las ganancias mencionadas se observan claramente en el gráfico 8,2.

A continuación se muestra el resultado de simular exactamente lo mismo, pero con media de duración de los flujos 300.

Claramente se observa lo esperado: existen varios remates en los que no hay capacidad suficiente para rematar (no se remata nada). Dado que el tiempo entre remates es bastante menor que la media de duración de los flujos, en varias ocasiones los flujos correspondientes a las ofertas aceptadas continúan enviándose a la llegada del próximo remate. Eso se nota en el gráfico, en los tiempos en que la ganancia del remate es 0. Además la ganancia total es de 93,28, como se esperaba, es menor que la de la simulación anterior.

La simulación tiene un tiempo de ejecución de $281mseg$, dicho tiempo es menor que el de la simulación anterior. Este resultado coincide con lo esperado, ya que en los remates donde no se aceptan ofertas, la simulación avanza más rápido (que si se hubiese aceptado alguna oferta) pues no se consume tiempo en crear los flujos de las ofertas aceptadas, tiempo en crear los eventos de comienzo de envío de dichos flujos, tiempo en recorrer dichos eventos en la lista de eventos, etc.

Los datos obtenidos correspondientes a las ganancias del gestor en cada remate en esta última simulación se encuentran en la misma carpeta que la anterior

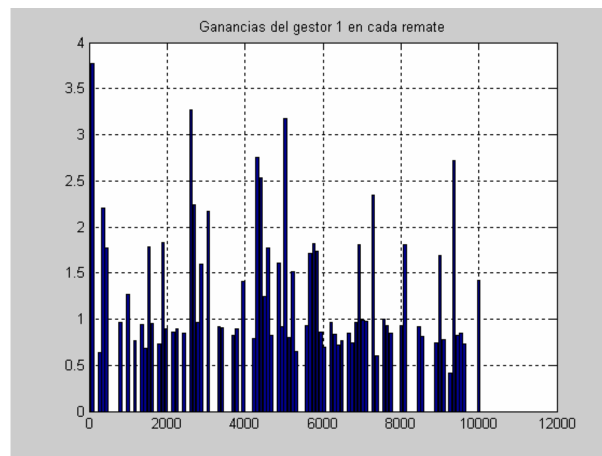

Figura 8.3: Ganancias por remate, media de duración 300

pero con el nombre `g12.txt`.

8.1.2. Simulación 2

Se realizó también una simulación, con la misma topología que la anterior, pero con 20 clientes.

Diez de dichos clientes solicitando flujos streaming de rate $512b/s$ con una media de duración distinta. Cinco de los clientes realizando peticiones de flujos que se transmitan a tasa de $256b/s$ y los cinco restantes a $64b/s$. A todos los clientes se les configuró la media de oferta en 1.

El enlace tiene una capacidad de $1024b/s$ y el tiempo entre cada subasta es de 90.

Al existir una única ruta posible y tres tipos de rate distintos a solicitar, se tuvieron que crear tres gestores. G1 el encargado de los flujos de $512b/s$, G2 el encargado de los de $256b/s$, y G3 el gestor de las ofertas de los flujos de $64b/s$.

A continuación se muestran las curvas correspondiente a la ganancia en cada remate de los tres gestores.

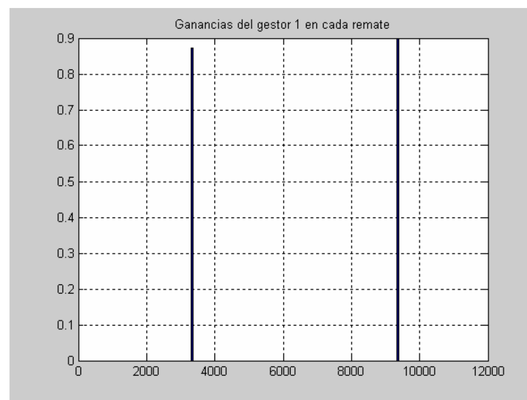

Figura 8.4: Ganancias por remate G1

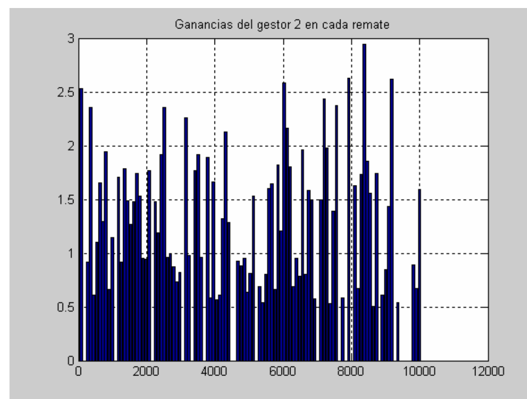

Figura 8.5: Ganancias por remate G2

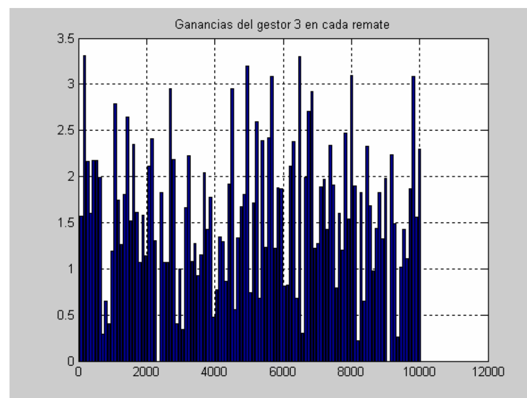

Figura 8.6: Ganancias por remate G3

Como se mencionó, los clientes creados en la simulación anterior tienen la misma **media de oferta** (mediaDeOferta = 1 en éste caso particular). En el momento de solicitar un flujo, cada cliente sortea su oferta correspondiente a un número aleatorio con distribución U[0,1] (en este caso).
Los datos obtenidos al graficar resultan razonables. Es bastante acertado pensar,

que si lo que se busca es obtener la mayor ganancia del operador, conviene más, aceptar más ofertas de menor rate, que llenar el enlace con ofertas de tasa de transmisión mayor, siendo las ofertas muy similares en su valor.

Gestor	Ganancia media
Gestor 1	0,885
Gestor 2	0,779
Gestor 3	0,585

Cuadro 8.3: Ganancias medias obtenidas por cada Gestor

8.2. Simulaciones con topologías formadas por dos enlaces

8.2.1. Simulación 1

Se realizó una simulación con la topología que se muestra en la figura siguiente.

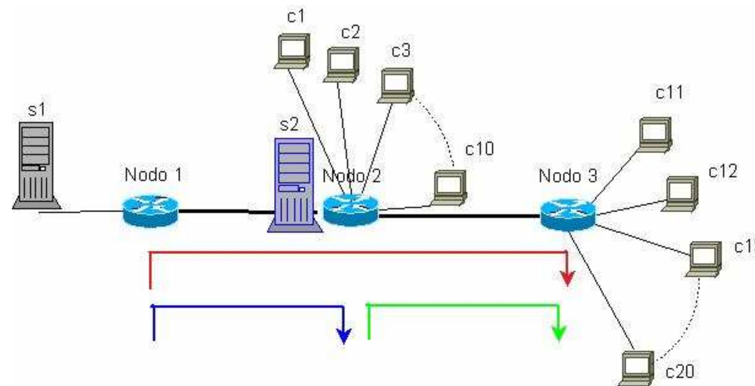

Figura 8.7: Representación de la topología simulada

Como se observa, existen dos servidores, uno ubicado en el **Nodo 1** y el otro en el **Nodo 2**. En la simulación participan 10 clientes, distribuidos como se muestra en la figura.

Los clientes asociados al **Nodo 2** solicitan flujos al servidor indicado en el gráfico como **s1**. Por otro lado los clientes del **Nodo 3** piden flujos al **s1** o al **s2**. Se deben definir tres rutas distintas (las rutas se muestran en la figura con los colores: rojo, verde y azul).

Para el enlace que va del **Nodo 1** al **Nodo 2** se definió la capacidad igual a $1000b/s$ y para el otro $500b/s$.

Para simplificar, todos los clientes solicitarán flujos de rate $100b/s$, por lo tanto se deberán crear tres elementos de la clase **Gestor**, uno para cada ruta.

Al igual que las demás simulaciones, se adjunta el archivo con los datos de configuración.

Datos de la simulación

Al simular, se obtuvo un tiempo de ejecución del sistema : $937mseg$.

Al igual que se hizo para la **simulación 1** de un enlace, se probará el algoritmo verificando los primeros dos remates. A continuación se observan tres tablas antes de ejecutar el primer remate, una para cada gestor, conteniendo las ofertas, junto con su duración y el cliente que la solicitó.

oferta	duración	cliente
0,7618	11,36	c3
0,5605	44,83	c6
0,4395	12,10	c1
0,4244	330,67	c8
0,3927	14,72	c7
0,2606	22,71	c4
0,1717	385,52	c9
0,0418	115,63	c10
0,0333	232,20	c2
0,0264	2,45	c5

Cuadro 8.4: Solicitudes para el G1 - ruta 1 (azul)

oferta	duración	cliente
0,6683	175,66	c15
0,5141	43,07	c11
0,3345	319,95	c14
0,2898	192,25	c17
0,2417	77,23	c12
0,2181	30,98	c16
0,1430	83,41	c13

Cuadro 8.5: Solicitudes para el G2 - ruta 2(verde)

oferta	duración	cliente
0,6121	21,80	c19
0,5163	85,07	c20
0,3140	63,11	c18

Cuadro 8.6: Solicitudes para el G3 - ruta 3(roja)

Como todos los flujos solicitados son para enviarse a $100b/s$, el **enlace 1** dejará pasar las mejores 10 ofertas de los gestores 1 y 3. Pero, por otro lado, el **enlace 2** aceptará las mejores 5 entre el gestor 2 y 3. Todo esto tratando de obtener la mayor ganancia en total.

Si se toman las ofertas de G1 y G3, las diez mejores estarían formadas por las 7 primeras de G1, y todas las de G3. Con esa elección, sólo se podrían aceptar las dos mejores ofertas de la lista de G2.
Dicha elección da una ganancia total de: 5,63.

Por otro lado, si se eligen las mejores 5 ofertas entre G2 y G3, éstas son las primeras 3 de G2 y las 2 primeras de G3. Con ésta elección se aceptarían las 8 mejores ofertas de G1.
Esta elección da una ganancia de 5,70.

La última forma de seleccionar los flujos a aceptar, es la que da la mayor ganancia total. El algoritmo obtiene el resultado correcto. Las figuras siguientes muestran la cantidad de ofertas aceptadas por cada gestor (datos obtenidos en la simulación). Se observa claramente, que en el primer remate, el gestor 1 (G1) aceptó 8 ofertas, el G2 aceptó 3 y el G3 sólo le permitió el envío a dos de sus ofertas de flujo.

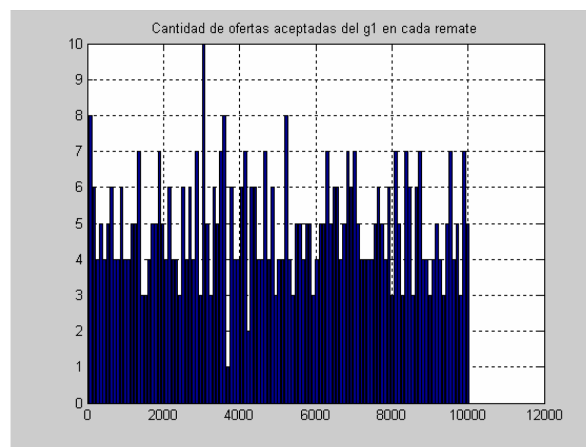

Figura 8.8: Cantidad de ofertas aceptadas por remate - G1

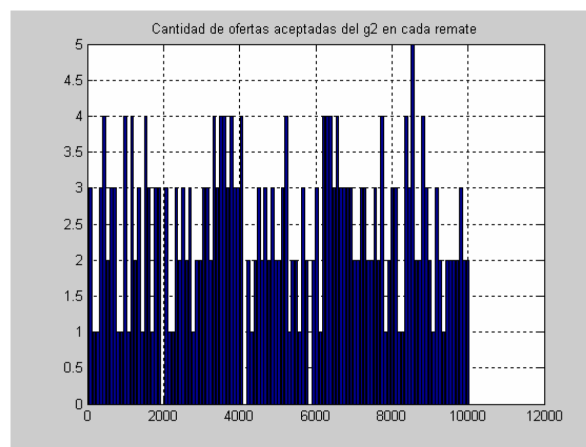

Figura 8.9: Cantidad de ofertas aceptadas por remate - G2

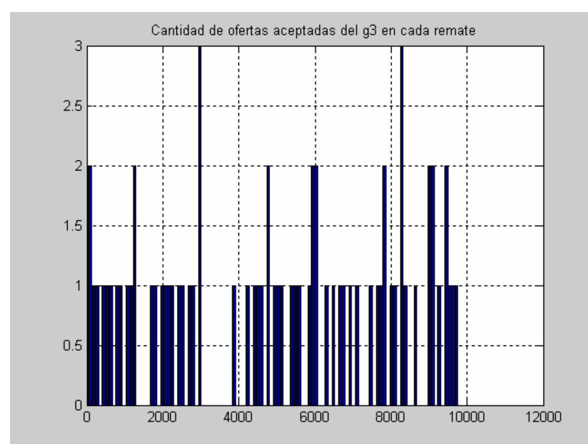

Figura 8.10: Cantidad de ofertas aceptadas por remate - G3

A continuación se observan las ofertas recibidas por cada gestor en el segundo remate.

oferta	duración	cliente
0,9214	85,48	c4
0,6890	8,08	c6
0,5466	149,29	c7
0,4200	157,66	c2
0,3598	90,31	c3
0,2221	14,38	c1
0,1204	4,01	c5

Cuadro 8.7: Solicitudes para el G1 (Segundo Remate) - ruta 1(azul)

oferta	duración	cliente
0,9267	122,24	c12
0,5020	19,53	c17
0,3966	118,74	c13
0,2547	78,45	c11
0,1125	467,97	c16

Cuadro 8.8: Solicitudes para el G2 (Segundo Remate) - ruta 2(verde)

oferta	duración	cliente
0,7791	44,43	c19
0,6719	30,95	c20
0,3747	41,19	c18

Cuadro 8.9: Solicitudes para el G3(Segundo Remate) - ruta 3(roja)

Al observar las tablas correspondientes a las ofertas de primer remate, se observan que 5 de las solicitudes aceptadas tienen un flujo asociado con duración mayor que el tiempo entre remates.

Por ello, en el enlace 1, la capacidad disponible para rematar en la segunda instancia es $700b/s$ y en el enlace 2 es de $300b/s$.

Según los gráficos anteriores, en el segundo remate, G1 acepta 6 solicitudes, y los otros dos 1. Este es un ejemplo en que el algoritmo no llega a la solución óptima. Esto pasa en ocasiones, consecuencia de la forma que se encontró para forzar la convergencia del algoritmo.

8.2.2. Simulación 2

A modo de ejemplo se adjunta una simulación con la misma topología que la anterior pero con clientes que solicitan flujos de distinta tasa de transferencia.

Los clientes ubicados en el **Nodo 2** solicitan flujos al servidor **s1**, algunos piden flujos de rate $50b/s$ y otros de tasa $100b/s$. Esto lleva a crear dos elementos gestores para la ruta 1 (azul), cada uno gestionando un rate distinto. En el archivo de entrada que se adjunta en la carpeta Simulaciones\FlujosStreaming\Dos Enlaces\Simulación 2, dichos gestores son "g1"y "g4".

Los clientes streaming asociados al **Nodo3**, al igual que en la Simulación 1, algunos solicitan flujos al **s1** y otros al **s2**. Los flujos solicitados al **s2** necesitan dos posibles tasas de transferencia ($80b/s$ o $100b/s$).

En la carpeta mencionada del CD, se encuentra el archivo de texto (usado para cargar los datos de configuración) y también los datos obtenidos al correr la simulación.

El tiempo ejecución de la simulación mencionada fue de $1094mseg$.

Capítulo 9

Simulaciones con flujos elásticos

Para los flujos elásticos, se implementó el algoritmo de **Max - Min Fairness**. El funcionamiento de dicho algoritmo fue verificado para una topología de un enlace en al realizar las simulaciones mencionadas en el capítulo 3. Se adjuntan algunas simulaciones que muestran el buen funcionamiento del algoritmo para el caso de un enlace en Apéndice E.

Además, resultó interesante adjuntar al documento, en la carpeta de las simulaciones correspondiente a los flujos elásticos, una simulación con una topología constituida por dos enlaces, y tres nodos (igual que la topología de las simulaciones del capítulo anterior).

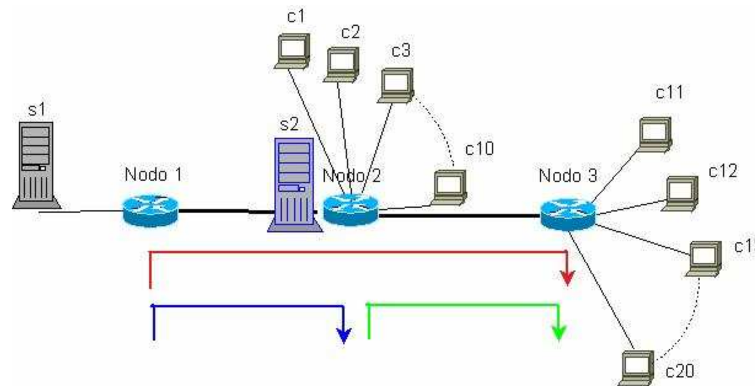

Figura 9.1: Representación de la topología simulada

Como lo muestra la figura, existen 20 clientes, dichos clientes son del tipo **ClienteElastico** y consecuentemente los servidores presentes son instancias de la clase **ServidorElastico**.

En ésta simulación, no se necesitan gestores.
Las capacidades de los enlaces se fijaron en $1000b/s$ y $500b/s$ para los enlaces 1 y 2 respectivamente[1].

[1]enlace1: nodo inicio = **Nodo 1** y nodo fin = **Nodo 2**

Los resultados de la simulación están en:
Simulaciones\FlujosElasticos\.

El tiempo de ejecución de la simulación es $72062 mseg$.

Si se observan los datos del archivo de texto "FlujosElasticos.txt"se puede analizar lo siguiente:

El primer flujo en enviarse tiene la **Ruta 3** como ruta asociada. Dicha ruta es la que va desde el **Nodo1** al **Nodo3**. Al correr el algoritmo de **Max - Min Fairness**, se le asocia al flujo un rate de $500b/s$. Dicha tasa coincide con la capacidad total del enlace2 que es el limitante en éste caso.

Al comenzar el envío del próximo flujo, éste tiene asociada la **Ruta 1** y por lo tanto se le asigna $1000b/s$ como rate, dado que hasta el momento no hay flujos enviándose por el enlace 1.

Comienza el envío del segundo flujo en el tiempo 0,47, en ese tiempo el flujo anterior ya se había terminado de enviar. El nuevo flujo está asociado a la **Ruta 2** y como no hay ningún flujo usando la capacidad del enlace 2, el algoritmo le asigna todo el ancho de banda a dicho flujo ($500b/s$).

El algoritmo comienza a ser interesante al llegar el sexto flujo, pues en ese momento el flujo 5 todavía está siendo enviado. Ambos tienen la misma ruta asociada. El algoritmo de **Max - Min Fairness** asigna una tasa de $250b/s$ a cada uno, tal como se esperaba. También se observa que se actualiza el tiempo de fin del flujo que cambió su tasa de transmisión.

Más interesante aún, es en el tiempo 7,46. En ese tiempo existen 5 flujos enviándose en la red. Tres por la **Ruta 2**, uno por la **Ruta 3** y uno por la **Ruta 1**. En el siguiente tiempo (7,58), uno de los flujos correspondientes a la **Ruta 3** termina su envío. El algoritmo actúa perfectamente. A continuación se muestran dos tablas con los rates asignados antes y después de la partida del flujo.

rate	ruta asociada
875	$ruta1$
125	$ruta2$
125	$ruta2$
125	$ruta2$
125	$ruta3$

Cuadro 9.1: Tasas asignadas en el tiempo 7,46

rate	ruta asociada
883,333333	$ruta1$
166,666666	$ruta2$
166,666666	$ruta2$
166,666666	$ruta3$

Cuadro 9.2: Tasas asignadas en el tiempo 7,58 - después de terminar de enviar un flujo

Capítulo 10

Simulaciones con flujos elásticos y flujos streaming

10.1. Simulación 1

Se realizó una simulación con la topología formada por dos enlaces utilizada en simulaciones anteriores. La diferencia es que se crearon dos instancias de la clase **ServidorElastico** y dos de **ServidorStreaming**.

Se ubicaron dos servidores en el **Nodo 1** (uno de cada clase) y los restantes en el **Nodo 2**. En la simulación participaron 20 clientes, diez que solicitaban flujos elásticos y los demás flujos streaming.

Los datos obtenidos junto al archivo de entrada se adjuntan en:
`Simulaciones\Los dos tipos de flujos\2 Enlaces\`.

La simulación realizada tuvo un tiempo de ejecución de $22453 mseg$, se considera dicho tiempo razonable teniendo en cuenta la cantidad de clientes y los parámetros de configuración de la simulación.

A continuación se muestran dos gráficas, una para cada enlace. En dichos gráficos se refleja la capacidad destinada para flujos streaming y la capacidad destinada para flujos elásticos para cada tiempo de remate.

Si se observa el archivo de configuración usado, se seteó como capacidad máxima para ser ocupada por los flujos streaming, el 80 % de la capacidad total en el enlace 1 y el 90 % en el enlace 2.

Si se observa la gráfica correspondiente al enlace 1, los resultados obtenidos son coherentes ya que todos los clientes streaming definidos están asociados al nodo 3 y sólo dos de ellos solicitan flujos al servidor ubicado en el nodo 1. (Ver topología en el capítulo anterior.)

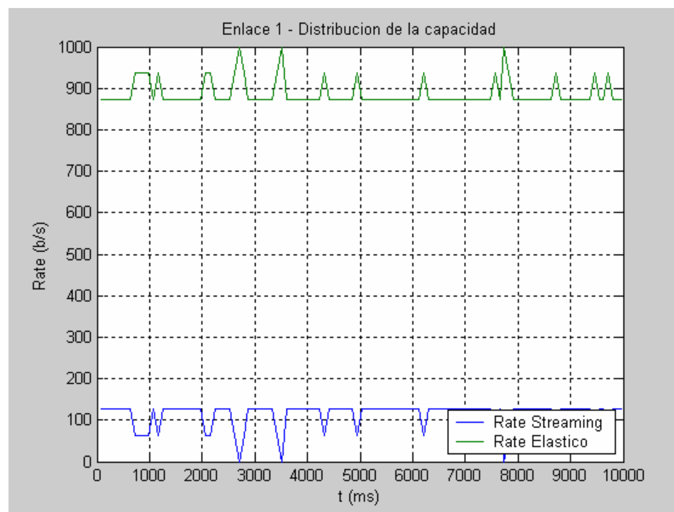

Figura 10.1: Distribución de la capacidad del enlace 1

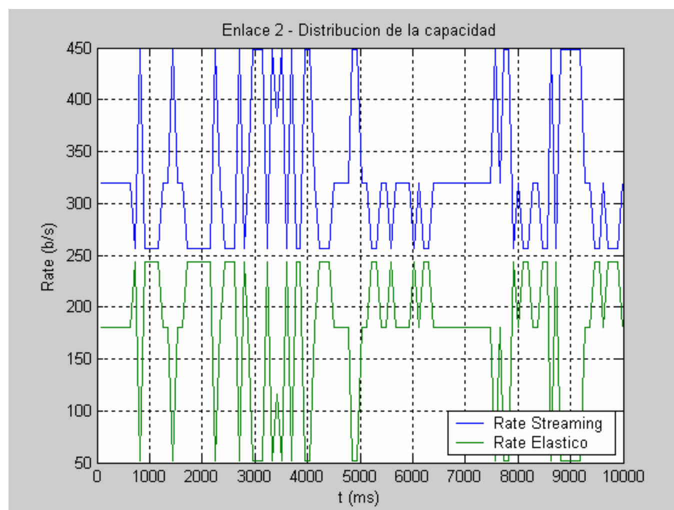

Figura 10.2: Distribución de la capacidad del enlace 2

10.2. Simulación 2

Para probar el simulador en topologías más grandes. Se realizó la simulación correspondiente a la arquitectura que se hace referencia en la figura 10,3.

Con la simulación se intentó simular una topología compleja, formada por 20 enlaces y 18 nodos. Se implementaron dos servidores, uno encargado de generar flujos setreaming y el otro flujos elásticos.
Algunos de los clientes no comienzan las peticiones de flujos al inicio de la simulación, sino que lo hacen después de avanzada la misma.

Los datos obtenidos al realizar esta simulación junto con los datos del archivo de entrada se encuentran en `Simulaciones\Los dos tipos de flujos\20 Enlaces\`.

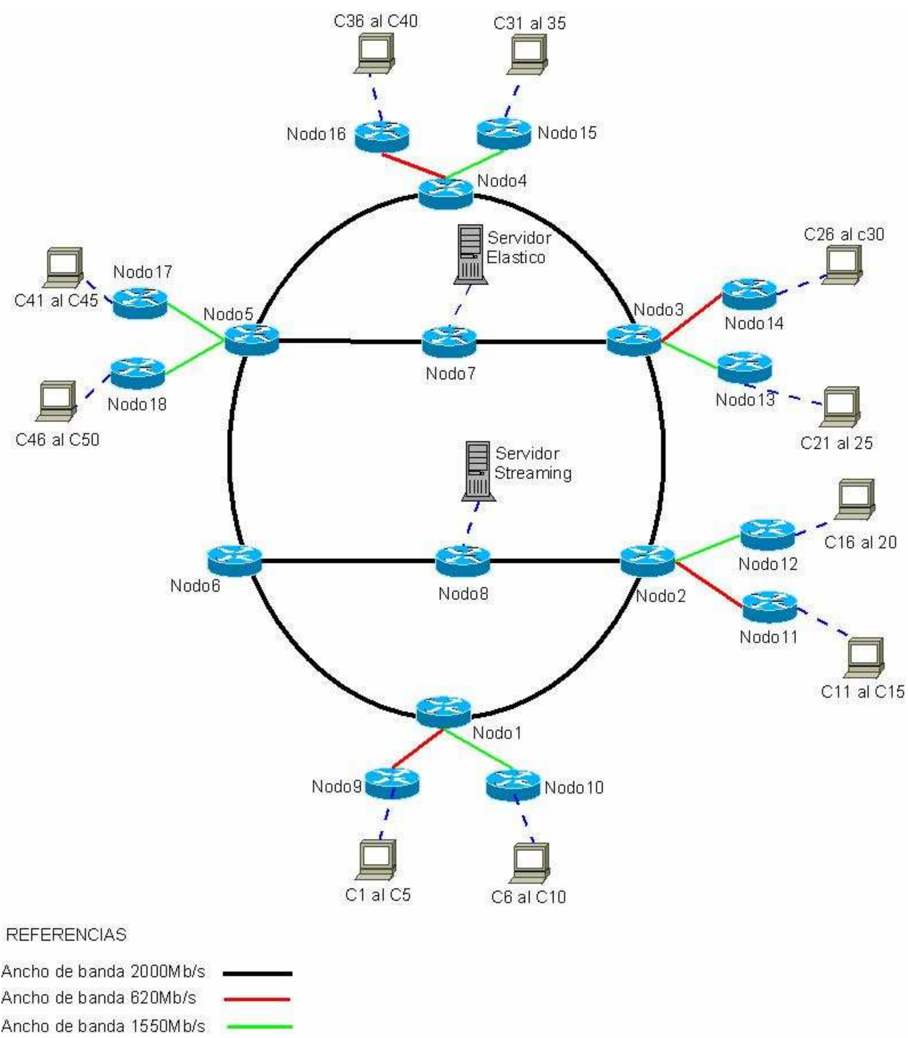

Figura 10.3: Representación de la topología simulada

El tiempo de ejecución de la misma fue de $21063 mseg$, se puede concluir que es aceptable dado lo complejo de la topología.

Capítulo 11

Conclusiones

Se puede concluir que los objetivos planteados al inicio del proyecto fueron cumplidos.

Se logró implementar una herramienta de simulación de redes multiservicio a escala de flujos.

En dicha herramienta, se puede simular la topología que se desea.

Se consiguió representar en la topología deseada, los servicios de Redes Triple Play (datos, voz y video). Dichos servicios se representan en el simulador como instancias de las clases **FlujoElastico** y **FlujoStreaming**.

Otros logros importantes fueron: la implementación de un algoritmo de asignación de recursos **Max - Min Fairness** (usado para asignar las tasas de envío a los flujos tipo datos), y una política de tarifación aplicada para el caso de flujos streaming.

Se implementó una clase, **Gestor**, que permiten extender el software para implementar nuevas políticas que se diseñen en el futuro.

La herramienta permite simular la topología deseada alcanzando tiempos de ejecución razonablemente buenos.

Es importante destacar que se implementó una interfaz gráfica para que el uso del simulador y la obtención de los resultados de cada simulación sean más amigables al usuario. Dicho objetivo no estaba previsto al inicio del proyecto.

También, a lo largo de la ejecución del mismo, se profundizaron los conocimientos en distintas áreas. Tal es el caso de la elección del lenguaje Java, hasta ahora solo se contaba con el nivel cognitivo adquirido durante la carrera lo cual resultaba insuficiente para el desarrollo de un software de este porte.

Para los integrantes del proyecto, este trabajo resultó en un gran aprendizaje para afrontar futuras experiencias en la vida profesional. La forma en que se evolucionó en el año de duración del proyecto y la manera en que se trabajó en el mismo, dejaron grandes enseñanzas para la vida.

Bibliografía

[1] Blanks, Jerry; Carson, Jhon S. y Nelson, Barry L. - *Discrete - Event System Simulation.* Segunda Edición. New Jersey 07458. ISBN: 0-13-217449-9

[2] Facultad de Ingeniería - UdelaR - *Apuntes del curso de Ruteo IP y Tecnologías de transporte.*

[3] ALCATEL - Disponible en internet en: *http:// www.alcatel-lucent.com - Documento de Redes Triple Play*

[4] Domínguez U. de Oregón, José - *Implementando IP Multicast.* Disponible en internet en: *eslared.org.ve/walc2004/apc-aa/archivos-aa/1e60354f4717edb9fb793dbc5219499d/IP_Multicast.pdf*

[5] Halabi, Sam - *Metro Ethernet.* Cisco 2003- ISBN: 158705096X

[6] Tutorial técnico de VPLS - *Revista de Telecomunicaciones de Alcatel - 4º trimestre de 2004*

[7] Ciciora, Walter - *Modern Cable Television Technology* Segunda Edición 2004 - ISBN: 1558608281

[9] Kumar, Anurag; Manjunath y Kuri, Joy - *Communication Networks* Segunda Edición. ISBN: 0-12-428751-4

[10] Ing. Fernando Paganini - Disponible en internet en: *telcom2006.fing.edu.uy/conferencias/pricing.pdf*

[11] Algoritmo de Water Filling - Disponible en internet en: *http://webscripts.softpedia.com/script/Communication-Tools/Water-Filling-Algorithm-31876.html*

[12] Buschiazzo, Dario; Ferragut, Andrés y Vázquez, Alejandro - *Ingeniería de Tráfico y Calidad de Servicio en Redes MPLS.* Proyecto de Fin de Carrera de Ing. Eléctrica - Año 2004

[13] G. de Veciana, T. Lee, T. Konstantopoulos - *Stability and Performance Analysis of Networks Supporting Services with Rate Control - Could the Internet Be Unstable?* INFOCOM 1999: 802-810

[14] J. Roberts and L. Massoulié - *Bandwidth sharing and admission control for elastic traffic.* - ITC Specialist Seminar, Yokohama, October 1998.

Apéndices

Apéndice A

Triple Play

A.1. Introducción

En la actualidad los operadores enfrentan la necesidad de crear nuevos servicios para mantener los clientes y captar nuevos, así como asegurar el crecimiento y las ganancias de la empresa.

Como solución a este problema, los operadores optan por ofrecer paquetes de servicios que sean atractivos para los usuarios desde una perspectiva económica. Es decir, que la compra de este paquete sea económicamente más rentable que si se optara por la compra de estos servicios de forma separada. Principalmente, si se tiene en cuenta que estos servicios de forma separada son ofrecidos por proveedores diferentes.

Si se estudian las tendencias a nivel mundial, parece ser que existen servicios que están ganando terreno en el mundo de las telco. Estos servicios son HSI[1], Voz IP y Video IP.

Bajo este marco surge Triple Play, cuyo objetivo es ofrecer los servicios de voz, datos y video en una única red IP.

En el siguiente capítulo se describe de forma breve la propuesta que Alcatel propone para el acceso a una red que implemente servicios Triple Play mostrando las limitaciones de las redes actuales y las posibles soluciones por las que los proveedores pueden optar. Se hace referencia también, a las diferentes tecnologías que forman parte de la solución de Triple Play para la mejor comprensión de la solución en su totalidad.

Además de la propuesta de Alcatel, es interesante conocer la forma en que implementan estos servicios las empresas de televisión para abonados, y en base a esto se consultó material brindado por la empresa "Nuevo Siglo". En el documento se explica un resumen de los aspectos más importantes.

[1] High Speed Internet

A.2. Propuesta de ALCATEL

A.2.1. Limitaciones de las redes actuales

Dar un servicio de Triple Play para un proveedor implica realizar modificaciones a la red actual, ya que la misma esta diseñada para la optimización de aplicaciones básicas como "Web browsing"y FTP.

En las redes actuales se tomaron como bases de diseño que, primero, para soportar servicios y aplicaciones como Web y FTP es suficiente con garantizar un servicio best effort[2] , y segundo, que la eficiencia de la red aumenta al agregar tráfico de muchos usuarios.

Para considerar que la eficiencia de la red aumenta al aumentar la cantidad de usuarios, se trabaja bajo los supuestos de que no todos los usuarios están en línea al mismo tiempo y que la demanda media es menor que la demanda pico. Estas consideraciones son aceptables para aplicaciones que no se realizan en tiempo real, pero para aplicaciones como por ejemplo transmisión de voz, la pérdida de paquetes puede hacer inaceptable la comunicación bajando la calidad y llegando incluso a caerse la conexión.

Otro punto a resolver en la red actual, es que el ancho de banda requerido por usuario para video es 50 veces el disponible para acceso a Internet. Esto causaría pérdidas de paquetes lo que inmediatamente repercutiría en la imagen. A esto se la agrega, el hecho de que es erróneo considerar que la demanda media es menor que la demanda pico, pues hay horarios "primetime"en los se debe tener en cuenta que una gran cantidad de usuarios van a estar mirando TV.

Modificaciones en la red de acceso a Internet deben realizarse si se quiere aumentar el ancho de banda y la calidad. Las arquitecturas de acceso a HSI están basadas actualmente en conexiones punto a punto a un único equipo (*BRAS*). Este equipo concentra todos los datos de los usuarios y las funciones de control de la red. Al aumentar la demanda de ancho de banda y la QoS, este equipo crece en requerimientos y costos.

A.2.2. Arquitectura y funcionalidades de la red Triple Play

Triple play propone para solucionar los problemas mencionados que funciones diferentes sean realizadas por equipos distintos.

La arquitectura Triple Play se basa en la división de la red de acceso en dos regiones para hacer un gerenciamiento de tráfico, primero a nivel de usuario y segundo a nivel de tipo de servicio, esto permite hacer un control de tráfico en diferentes escalas.

En vista de que Alcatel en uno de los mayores proveedores de servicios de este tipo a nivel mundial, la explicación estará basada en la propuesta de dicha empresa.

[2]Best Effort implica que el usuario envía paquetes a la red y esta hace su mejor esfuerzo para hacerlos llegar al destinatario, no asegurando ningún tipo de calidad de servicio (pérdidas, retardos, etc)

Esto se puede observar en la siguiente figura.

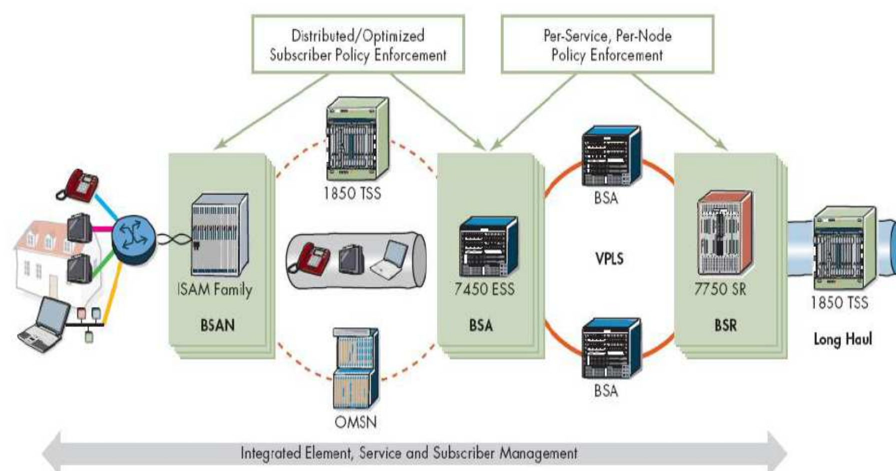

Figura A.1: Arquitectura Triple Play propuesta por Alcatel

La propuesta de Alcatel es la de utilizar diferentes equipos en diversas regiones de la red, cada uno implementando diferentes funciones para garantizar una buena QoS.

A continuación se detallan los nodos que se muestran en la Figura C1.

BSA [3]

Como se mencionó antes, para garantizar calidad de servicio, Alcatel propone la división del acceso, el BSA es el nodo encargado de agregar el tráfico por usuario en tráfico por servicio.

Este equipo es un switch Ethernet con soporte de QoS y policing, que se conecta via Ethernet/VPLS al BSR.

BSR [4]

Es el equipo que se encuentra entre la red Ethernet/VPLS e IP/MPLS. Asigna vía DHCP direcciones IP (en lugar de conexiones PPPoE) a los usuarios. Junto con el BSA y la red VPLS sustituyen a la vieja arquitectura BRAS/ATM de agregación de las redes de acceso.

BSAN [5]

Es el elemento de la red que se encarga de realizar el filtrado, autenticación de usuarios e IP multicast. [3]

[3]Broadband Service Aggregator
[4]Broadband Service Router
[5]Broadband Service Access Node

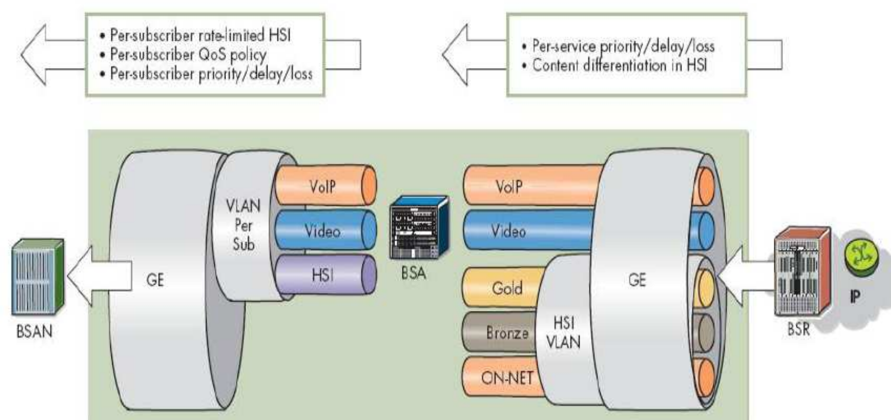

Figura A.2: Calidad de Servicio en los diferentes niveles de agregación

A.2.3. Tecnologías Asociadas

DiffServ

Calidad de Servicio en redes IP

La calidad de servicio depende de la percepción del usuario, es difícil de medir. Además depende del servicio, algunos tienen recomendaciones para garantizar cierta QoS, como ser: requerimientos en el ancho de banda, retardos, etc.

Existen diversos factores que dificultan poder garantizar cierta QoS en redes IP, ejemplo de esto: IP fue pensado con el paradigma ˵best effort˝. La buena escalabilidad de IP, se debe en gran parte a que los enrutadores son ˵tontos˝, la inteligencia está en las puntas. IP no es adecuada para brindar servicios de video y voz streaming, esto último debido a las pérdidas en ráfaga y el jitter.

Hasta el momento, la solución es sobredimensionar la red, pero esto no es suficiente. El que no sea suficiente se debe en gran parte a que se genera una pérdida de recursos importantes debido al sobredimensionamiento en los picos (el tráfico se da típicamente en ráfagas, produciendo congestiones temporales), además los puntos de mayor demanda y congestión son variables, otro factor importante es que el protocolo TCP tiene como lógica congestionar la red. [2]

Arquitecturas para ofrecer QoS en IP

IntServ

Basado en la utilización de algún protocolo de reserva (RSVP [6]) que permite la reserva de recursos a lo largo de los routers implicados en la comunicación. El principal problema de este modelo es la necesidad de mantener información sobre cada flujo en todos los routers de la red, lo cual lleva a problemas de escalabilidad.

[6] ReSerVation Protocol

DiffServ

En esta arquitectura los paquetes son clasificados y marcados para recibir un trato particular en cuanto al envío en cada salto. Esta arquitectura logra escalabilidad al implementar las funciones de clasificación y condicionamiento solo en los nodos de los bordes.

Esta arquitectura sólo provee servicio diferenciado en una dirección del flujo de tráfico y es por ende asimétrica.

En el caso del paquete IPv4, en el encabezado existe 1byte denominado tipo de servicio (TOS), este campo es poco utilizado en los paquetes "IP puros". Existe un campo equivalente en IPv6 llamado octeto de clase de servicio. En DiffServ se utilizan 6 de estos bits para marcarlos con un patrón que permite que los routers sepan que deben hacer con el paquete. El marcado puede ocurrir en dos lugares: en la fuente original del tráfico o en el primer router que en tráfico encuentra. (Los 2 bits restantes del octeto no se usan con este fin).

A la etiqueta se le conoce con el nombre de DSCP (Differentiated Service Code Point).

Cuando un paquete entra a un router, la lógica de salida selecciona su puerto de salida y el valor del DSCP es usado para conducir al paquete a una cola específica de tratamiento específico en ese puerto.

Como conclusión, tenemos que DiffServ soluciona el problema de la escalabilidad de IntServ, controla flujos agregados, pero presenta problemas como posibles bloqueos de flujos de prioridades bajas y si hay congestión puede degradarse la QoS aun en clases "altas".

Cabe mencionar también que DiffServ es el de mayor uso y es el que ha prosperado a nivel mundial. Fue originalmente pensado para IP, pero tecnologías como MPLS y Ethernet, que son "más de transporte"permiten mapear los bits usados en DiffServ para hacer QoS. [2]

Ejemplo de estos son: 802.1p y 802.1q

IP Multicast

IP Multicast es una herramienta que permite maximizar la eficiencia de una red, disminuyendo la cantidad de copias de un canal y el procesamiento en los enrutadores.

La comunicación IP tradicional, conocida como unicast, pasa información de una fuente a un único receptor. IP Multicast por el contrario comunica una fuente con miembros de un grupo que han expresado interés.

Cada paquete transmitido lleva una dirección origen y una dirección destino que corresponde a un grupo. Cada grupo tiene asociada una dirección IP Clase D y sus miembros pueden estar en cualquier lugar de la red. Los miembros de los grupos pueden subscribirse y retirarse indicando estas acciones a los enrutadores

mediante el protocolo IGMP.

Los enrutadores utilizan protocolos de enrutamiento de multicast para administrar los grupos, enviar mensajes, determinar el árbol de distribución multicast y las tablas de ruteo.

Entre estos protocolos estan DVMRP (Distance Vector Multicast Routing Protocol) y PIM (Protocol Independent Multicast). De este último existen variantes como ser PIM-DM (Dense Mode), utilizado para redes en las que existen varios transmisores y pocos receptores y donde las distancias de unos con otros son chicas y PIM-SM (Sparse Mode), utilizado en redes donde transmisores y receptores se encuentran separados por redes WAN. PIM-SSM (Source Specific Multicast) es una variante de PIM-SM. PIM-SSM esta optimizado para trabajar con una única fuente y miembros de un grupo multicast. De esta forma los miembros del grupo solo pueden recibir información de una fuente ya definida, eliminando lo que proviene de otras fuentes.

Cuando los paquetes llegan al router, este debe decidir hacia dónde replicarlos. Para ello utiliza el protocolo RPF. La verificación en RPF se hace comparando la tabla de enrutamiento para multicast con la dirección de origen del paquete. Si el paquete arribó en la interfaz especificada en la tabla de enrutamiento para la dirección de origen entonces la verificación es satisfactoria y el paquete es replicado. Sino el paquete es descartado en silencio.

Como puede verse el enrutamiento en multicast esta basado en la dirección origen del paquete y no en la de destino como en unicast. [4]

Metro Ethernet

La red metro ethernet es una arquitectura tecnológica destinada a suministrar servicios de conectividad MAN /WAN de nivel 2, a través de UNIs Ethernet. Estas redes se basan en sistemas multiservicio, es decir, soportan aplicaciones en tiempo real, streaming, flujos de datos continuos como audio y video, etc.

Los servicios metro ethernet no necesariamente necesitan Ethernet como tecnología de transporte, sino que pueden utilizar Ethernet sobre SONET/SDH (EOS), RPR (Resilient Packet Ring) o MPLS.

Operadores que ya han montado su red SONET/SDH prefieren utilizar esta red para dar servicios ethernet. Existen dos formas estandarizadas de llevar ethernet sobre SONET/SDH, LAPS y GFP. La trama ethernet es encapsulada y mapeada hacia la carga útil de SONET/SDH (SPE), se transporta por el anillo y se extrae al final. Estas acciones son realizadas por la función EOS. La función EOS puede ser implementada en diferentes lugares, dentro del equipo SONET/SDH (ADM) o en el switch. RPR es un protocolo de capa MAC diseñado para realizar el control del ancho de banda y facilitar el desarrollo de servicios orientados a datos en redes con arquitectura de anillo.

La implementación de RPR es más frecuente en proveedores de cable, que realizan la agregación de nodos llamados CMTSs a routers RPR. Estos routers se conectan por OC48 RPR y luego el tráfico se agrega a través de un hub a Internet.

Comparando RPT y SONET/SDH, RPT corre con la ventaja de que desde sus orígenes fue desarrollado para el uso de datos.

Desde el 2000 han aparecido varias propuestas para extender ethernet a la capa de transporte, muchas funcionaron, otras no. Las razones para implementar ethernet son varias, entre ellas el bajo costo y el ancho de banda. Cuando se usa ethernet en el transporte, el acceso se puede hacer con dos topologías diferentes, como anillo o "hub and spoke". La elección de una u otra depende en la experiencia anterior del proveedor, para el que esté acostumbrado a trabajar con topologías de anillos, seguramente ésta sea la más rentable, pero para el que no lo está, puede resultar costoso.

Los anillos Gigabit Ethernet son conexiones punto a punto entre switches. En esta topología se deben tener en cuenta limitaciones como el ancho de banda (Solo 1GB) y las protecciones que se deben realizar, parte de ese ancho de banda se reserva para la prevención de loops.

En una configuración Gigabit Ethernet Hub-and-Spoke, los switches de los edificios se unen a través de dos links a un switch borde. En este modelo el ancho de banda de cada usuario puede variar. Si bien este tipo de acceso es más caro, muchos proveedores lo prefieren, comparando con la implementación del anillo, debido a la escalabilidad y viabilidad. [5]

MPLS (Multiprotocol Label Switching)

Es un mecanismo de transporte de datos estándar creado por la IETF y definido en el RFC3031. Opera entre la capa de red y la capa de enlace del modelo de capas OSI. Es una tecnología que maneja distintos tipos de tráfico de manera eficiente.

Surgió por la necesidad de solucionar el problema del gran crecimiento de las tablas de ruteo.[7]. Otro problema es que se quería unificar el servicio de transporte de datos para las redes basadas en circuitos y las basadas en paquetes, ya que por ejemplo usando IP/ATM se tienen problemas de escalabilidad, de gestión (al existir dos tecnologías existen dos planos de gestión).

La idea de MPLS es realizar una partición del conjunto de todos los posibles paquetes en clases de equivalencias (FECs: Forwarding Equivalence Classes - etiquetas). Los paquetes de una misma clase de equivalencia van a ser enviados de la misma manera. Para cada FEC se construye un túnel: LSP (Label Switched Path), es como un circuito virtual. (Usa la idea de ATM).

Los enrutadores en MPLS se conocen como LSRs. Cada vez que un paquete ingresa a un dominio MPLS a través de un router de borde (LER [8]) se le asigna una etiqueta (FEC), la cual va a determinar el tratamiento y la ruta que transitará dicho paquete. En cada uno de los saltos cada LSR lee la etiqueta y decide por medio de ésta el tratamiento que debe darle al paquete. De esta forma cada nodo de una red MPLS decide que hacer con el paquete solo mirando la etiqueta, sin necesidad de examinar el encabezado de capa de red correspondiente.

[7]Esto ya no es un problema debido a que actualmente se hace mucho por hardware
[8]Label Edge Router

Las etiquetas son un conjunto de 4 Bytes que indican que hacer con el paquete. En general cada paquete tiene un conjunto de estas etiquetas, un stack. La gran ventaja del stack de etiquetas es que permite la agregación (simplificación de las tablas en el interior de la red) y permite distintos niveles de jerarquías. La utilización de etiquetas permite un encaminamiento más simple y rápido.

La gran ventaja de MPLS es poder realizar Ingeniería de Tráfico. Se puede realizar balance de carga, para que no se saturen algunos enlaces y otros no. Permite la reserva de recursos en la base de la demanda (CBR), se permiten definir rutas explícitas, etc. [2]

VPLS (Virtual Private LAN Service)

VPLS es también conocido como TLS (Servicio de LAN Transparente), es una VPN multipunto de Capa 2 que permite conectar múltiples sitios en un único dominio puenteado sobre una red MPLS/IP gestionada por el proveedor. Todos los sitios del cliente en un caso de VPLS parecen estar en la misma LAN [9], sin tener en cuenta sus localizaciones. VPLS utiliza una interfaz Ethernet con el cliente, simplificando la frontera LAN/WAN [10] y permitiendo un aprovisionamiento rápido y flexible del servicio.

Una red con VPLS consta de CEs (bordes de cliente), PEs (bordes de proveedor) y de una red central MPLS: El dispositivo CE es un router o conmutador situado en las instalaciones del cliente y se conecta al PE mediante un AC (circuito de conexión). En VPLS se asume que Ethernet es la interfaz entre el CE y PE. El dispositivo PE es donde reside toda la inteligencia de VPN, donde el VPLS comienza y termina, y donde se establecen todos los túneles necesarios para conectar con todos los otros PEs. Como VPLS es un servicio Ethernet de Capa 2, el PE debe ser capaz de conocer, puentear y replicar la MAC [11] en base a VPLSs. La red central MPLS/IP que interconecta los PEs no participa realmente en la funcionalidad de VPN. El tráfico se conmuta simplemente basándose en etiquetas MPLS. La base de cualquier servicio VPN multipunto es una malla completa de túneles MPLS que se establecen entre todos los PEs que participan en el servicio VPN. Se utiliza LDP (protocolo de distribución de etiqueta) para establecer estos túneles. Las VPNs multipunto pueden crearse encima de esta malla completa, ocultando la complejidad de la VPN desde los routers centrales. Para cada instancia VPLS se crea una malla completa de túneles internos (llamados pseudowires o pseudos cables) entre todos los PEs que participan en la instancia VPLS. Un mecanismo de auto-detección localiza todos los PEs que participan en la instancia VPLS. Un PW consta de un par de LSPs unidireccionales punto-a-punto de un solo salto en direcciones opuestas, cada uno identificado por una etiqueta PW, también llamada VC [12]. Las etiquetas PW se intercambian entre un par de PEs usando el mencionado protocolo de señalización LDP. El identificador VPLS se intercambia con las etiquetas, así ambos PWs pueden enlazarse y asociarse a una instancia VPLS particular. Este intercambio de etiquetas PW tiene que darse entre cada pareja de PEs participantes en una instancia VPLS concreta, y las etiquetas PW tienen solamente un significado local entre cada una de esas parejas.

[9] Red de Área Local
[10] Red de Área Extensa
[11] control de acceso a los medios
[12] Conexión Virtual

La creación de PWs con una pareja de LSPs permite a un PE participar en el aprendizaje del MAC: cuando PE recibe una trama Ethernet con una dirección de fuente MAC desconocida, el PE sabe en qué VC se envió. Los routers PE deben soportar todas las prestaciones "clásicas"Ethernet, como aprendizaje del MAC, replicación y envío de paquetes. Conocen las direcciones MAC de la fuente MAC del tráfico que llega a sus puertos de acceso y de red. Desde un punto de vista funcional, esto significa que los PEs deben implementar un puente por cada instancia VPLS, al que se le suele llamar VB (puente virtual). La funcionalidad VB se lleva a cabo en el PE mediante una FIB (retransmisión de base de información) para cada supuesto de VPLS; esta FIB se retransmite a con todas las direcciones MAC aprendidas. Todo el tráfico se conmuta en base a las direcciones MAC y se reenvía entre todos los routers PE participantes, usando túneles LSP. Los paquetes desconocidos (es decir, las direcciones de destino MAC que no han sido aprendidas) se replican y reenvían en todos los LSPs a todos los routers PE que participan en ese servicio hasta que responde la estación de destino y la dirección MAC es aprendida por los routers PE asociados con dicho servicio. Para evitar bucles de reenvío se usa la regla llamada "Split Horizon"[13]. En el contexto VPLS, esta regla implica básicamente que un PE nunca debe enviar un paquete a un PW si ese paquete se ha recibido de un PW. Esto asegura que el tráfico no pueda formar un bucle sobre la red de backbone usando PWs. El hecho de que haya siempre una malla completa de PWs entre los dispositivos PE asegura que cada paquete emitido alcanzará su destino dentro del VPLS.

VPLS Jerárquico o H-VPLS

La arquitectura H-VPLS se construye sobre la base de la solución VPLS, ampliándola para proporcionar distintas ventajas operacionales y de escala. Es especialmente útil en despliegues a gran escala con un gran número de PEs y/o MTU (unidades multiusuario).

Los proveedores de servicio instalan MTUs en edificios compartidos para dar servicio a distintas empresas radicadas en ellos; cada empresa puede, potencialmente, pertenecer a diferentes VPN VPLS. H-VPLS permite que los servicios VPLS se extiendan por múltiples redes metropolitanas. Se utiliza una conexión spoke para conectar cada servicio VPLS entre dos áreas metropolitanas. En su forma más simple, podría ser un LSP de túnel. Un conjunto de etiquetas PW de ingreso y egreso se intercambian entre los dispositivos PE de borde para crear un PW por cada instancia de servicio VPLS a transportar sobre este LSP. Los routers PE en cada extremo tratan este PW inter-metropolitano como una conexión spoke virtual para el servicio VPLS, de la misma manera que tratan las conexiones PE-MTU. Esta arquitectura reduce al mínimo la taza de señalización y evita una malla total de VCs y LSPs entre las dos redes metropolitanas. [6]

A.3. Propuesta de empresas cableras - Nuevo Siglo

A.3.1. Cable Data Transport

No solo las empresas de telefonía han adoptado la nueva tendencia de proveer a sus clientes de servicios de datos y televisión agregados a los ya tradicionales

[13]horizonte dividido

como la voz, sino que empresas de televisión por cable se han sumado a este gran desafío. Los factores que hacen al cable un medio atractivo son su precio, velocidad y que no es orientado a conexión.

La tecnología asociada a la transmisión de datos en una red de cable es el CABLE MODEM y el protocolo utilizado es el DOCSIS.

A continuación se explicarán conceptos que están relacionados al transporte de datos en una red HFC (Hibrid Fibre Coaxial).

Métodos de Modulación

Una de las razones por la que es interesante estudiar los diferentes métodos de modulación es la eficiencia espectral, es decir el ancho de banda necesario para la transmisión de cierta cantidad de datos por segundo. Un tipo de modulación con una alta eficiencia espectral necesitará menos ancho de banda que otro con menos eficiencia para transmitir la misma cantidad de datos. Por otro lado, es de esperar que sea más costoso y susceptible a la introducción de ruido y distorsión. Considerando *BPSK* vs *256-QAM*, BPSK necesita un ancho de banda igual al bit rate (para una tasa de 1 Mb/s se necesita 1Mhz de ancho de banda), por otro lado para transmitir lo mismo en 256-QAM se necesita solamente 0.125 MHz. Como contraposición para 256-QAM se necesita una C/N 13dB mejor y el costo del hardware es considerablemente más alto.

FSK (Frequency Shift Keying)

Los sistemas FSK son frecuentemente utilizados en aplicaciones de cable que requieren baja velocidad de transmisión de datos.

La eficiencia espectral es baja pero esto no tiene importancia para este tipo de aplicaciones. La transmisión no es fácilmente dañada por ruidos o distorsión, y el hardware no es costoso.

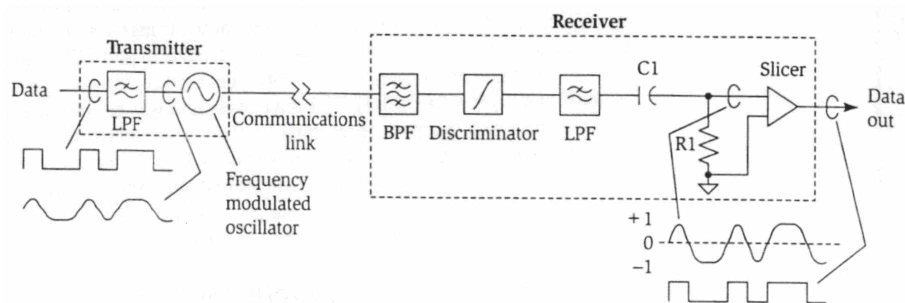

Figura A.3: Sistema FSK

En la transmisión de datos, no solo es necesario reconstruir la señal sino también se debe obtener la señal de reloj para identificar cuando una muestra debe ser tomada. Esto puede implementarse de varias formas pero la más utilizada en sistemas de bajo costo es la codificación Manchester. Esta codificación es aplicada

en los datos antes de ser modulada. La codificación Manchester da información de la señal de reloj, esto lo hace cambiando el nivel de la señal en la mitad de la celda correspondiente al bit.

BPSK (Biphase Shift Keying)

En BPSK la fase es corrida 180º entre la transmisión de un 1 o un 0. En la siguiente figura se puede ver un sistema BPSK con un flujo de datos NRZ.

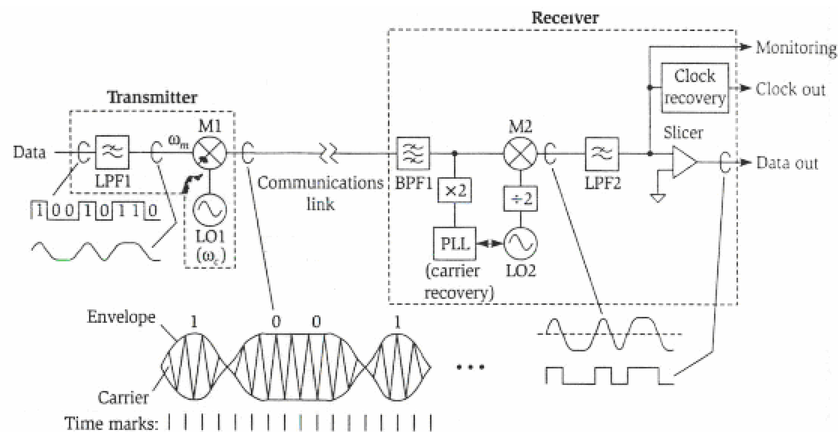

Figura A.4: Sistema de transmisión BPSK

Si bien se podría utilizar la codificación Manchester para la transmisión de la señal de reloj en BPSK, esta no se usa debido al ancho de banda que se requiere. Por lo cual lo que se transmite es un flujo de datos NRZ, y la reconstrucción del reloj es sincronizando las transiciones en los datos recibidos.

QPSK (Quadrature Phase Shift Keying)

BPSK tiene inmunidad al ruido, es simple y su costo es bajo, pero es ineficiente en el uso del espectro. Para realizar un mejor uso del espectro en QPSK hay diferentes niveles representados por diferentes valores analógicos y/o fases a los cuales es aproximada la señal, que luego son codificados y transmitidos. Esto se logra transmitiendo más de un bit al mismo tiempo.

Las desventajas de este sistema son una mayor complejidad y menos inmunidad al ruido.

Higher (Order QAM Modulation)

Es similar a QPSK, solo que se agregan niveles, mayor cantidad de divisiones en el eje.

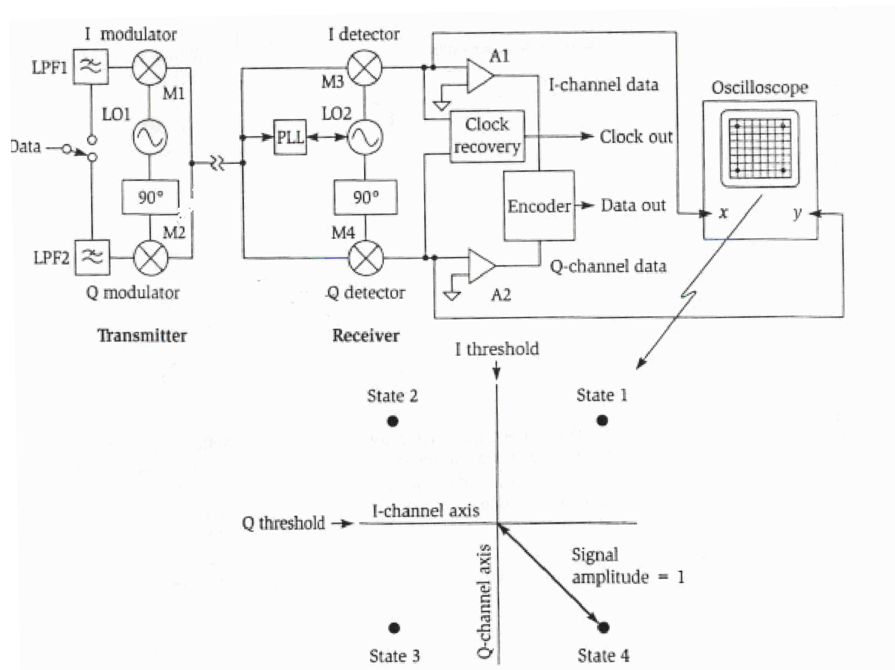

Figura A.5: Sistema básico QPSK

Dependiendo de la cantidad de divisiones que se hacen, se pueden tener las siguientes formas de modular: 16-QAM, 32-QAM ,64-QAM y 256-QAM. Las mismas son propuestas para servicios que requieren de una mejor eficiencia espectral que la ofrecida en QPSK.

Formas de compartir el espectro

A continuación se describen algunas formas de compartir el espectro de forma tal que su uso sea más eficiente.

TDMA (Time Division Multiple Access)

En un sistema TDMA cada transmisor transmite en la misma frecuencia pero en diferentes momentos. Se debe esperar cierto tiempo entre cada transmisión para evitar errores. Como cada transmisor no esta transmitiendo todo el tiempo, la tasa de transmisión debe ser lo suficientemente rápida para que se puedan enviar todos los bits.

Una señal de reloj debe ser transmitida para definir los time slots. No es necesario que cada time slot tenga la misma longitud, varios protocolos, incluido Docsis, permiten que el largo sea variado dependiendo en la cantidad de datos a transmitir.

FDM (Frequency Division Multiplexing)

Los sistemas FDM dividen el ancho de banda en subcanales que utilizan para cada transmisión, dedicándole un subcanal para cada transmisor. Se requiere de varios receptores para el procesamiento de las señales y se necesita una guarda de banda para que se pueda realizar el filtrado de las señales.

CDMA (Code Divison Multiple Access)

En un sistema CDMA los datos a modular son combinados con una secuencia específica de datos (PRBS) conocido como spreading code. Luego son modulados por un m-ary QAM, donde se utiliza la misma frecuencia portadora por todos los moduladores.

El resultado de esto es, la presencia de varios canales ocupando el mismo ancho de banda al mismo tiempo pero con diferentes códigos.

Todas las señales son recibidas simultáneamente y cada una tiene su despreading code de forma que se puedan separar unas de otras. Todos los códigos a utilizar deben ser ortogonales de forma de no interferir unos con otros.

Una limitante de la cantidad de señales que se pueden transmitir a la vez, es la cantidad de códigos ortogonales disponibles. La cantidad de códigos depende de la cantidad de bits de la secuencia.

Protocolo Docsis en Cable MODEM

Docsis es el protocolo, conjunto de especificaciones, utilizado para el transporte de datos en cable modem. El estándar le permite al consumidor pensar que si cambia de sistema de cable, podrá seguir utilizando el mismo modem.

La IEEE esta trabajando en otro estándar (802.14) que compartirá la misma capa física pero serán diferentes en capa MAC. De todas formas los dos estándares existirán en los mismos canales de bajada.

Capa Física

La capa física se encarga de la modulación de los datos y de la correcta sincronización entre el transmisor y receptor.

Docsis define los siguientes parámetros para el downstream y upstream.

Parámetros	Valores
Frecuencia	$91 - 857MHz$
Modulación	64 ó $256QAM$

Cuadro A.1: Características del downstream RF

Tasa de símbolo	Tasa de bit	Filtro
5,056941$Msim/s$	30,342Mb/s	18 % root raised cosine

Cuadro A.2: Parámetros para 64 QAM

Tasa de símbolo	Tasa de bit	Filtro
5,360537$Msim/s$	42,884Mb/s	12 % root raised cosine

Cuadro A.3: Parámetros para 256 QAM

Parámetros	Valores
Frecuencia	$5 - 42MHz$
Modulación	$QPSK$ ó 16QAM

Cuadro A.4: Características del upstream RF

Tasa de símbolo	Tasa de bit	Filtro
$160, 320, 640, 1{,}280, 2{,}560ksim/s$	$320, 640, 1{,}280, 2{,}560, 5{,}120Kb/s$	25 % root raised cosine

Cuadro A.5: Parámetros para QPSK

Tasa de símbolo	Tasa de bit	Filtro
$160, 320, 640, 1{,}280, 2{,}560ksim/s$	$640, 1{,}280, 2{,}560, 5{,}120, 10{,}240Kb/s$	25 % root raised cosine

Cuadro A.6: Parámetros para 16 QAM

Capa MAC

La capa MAC se encarga de la asignación de ancho de banda, es decir, determinación de la frecuencia y tasa de datos a utilizar. Esto es controlado por el CMTS (Cable modem termination system), de acuerdo a parámetros programados y condiciones existentes.

Debido a la existencia de "upstream messages"que son bastante cortos el protocolo permite, para mejorar la eficiencia, subdividir los time slots permitiendo el acceso a una mayor cantidad de módems.

Docsis agrega una subcapa de seguridad, adicional a la de capa Mac, SSI Security System Interface. Cable Modems tienen la libertad de implementarla o no.

A.3.2. Telefonía por Cable

El servicio de telefonía en una red de cable no se encuentra aún estandarizado por lo que se detallará una de las implementaciones mas usadas a nivel mundial.

En un principio los sistemas de telefonía estaban basados en tecnologías de switching circuits, pero ahora cada vez más la tendencia es la orientación a servicios sobre IP. En esta parte del documento se verá un poco de ambas.

Para poder entender la solución de telefonía para una red de cable se explicará primero la arquitectura de una red telefónica.

Conceptos básicos de telefonía

La arquitectura de una red de telefonía fija esta constituida por varios elementos. Uno de ellos es el **switch**, cuya funcionalidad es interconectar los teléfonos de los usuarios.

El switch tiene básicamente dos tipos de interfaces diferentes, una para la conexión con otros switches y otra para la conexión con los teléfonos.

Cada teléfono en la casa del usuario se conecta a un switch llamado **End Office o Central Office**. Las End Office se conectan entre sí(intraLATA) y con otras end office que comunican con el exterior (interLATA). El termino LATA significa Local Access and Transport.

En Estados Unidos, por ejemplo, el protocolo con el cual se hablan los switches es el SS7, el cual provee información básica de ruteo.

El switch concentra las siguientes funcionalidades:

- Detección de una llamada entrante
- Generación de tonos en la línea
- Almacenamiento de los dígitos discados
- Validación de dígitos
- Enrutamiento de llamadas

- Mantener el canal de la conversación
- Terminar una llamada

En la actualidad la conexión de los teléfonos no se hace directamente al switch, ya que los costos serían elevados debido a la gran cantidad de cobre que se necesitaría.

Una forma más eficiente de conectar los teléfonos al switch es a través de un DLC (Digital Loop Carrier).

Un sistema DLC agrega un concentrador para multiplexar las señales provenientes de los teléfonos de los clientes en una única línea al switch. El DLC es también el encargado de realizar la conversión de analógico a digital.

La interfaz estandarizada del DLC al switch es la TR-303, que es capaz de soportar hasta alrededor de 2048 líneas.

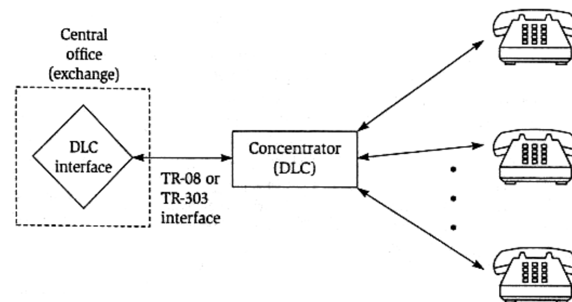

Figura A.6: Integración del DLC

Modificación del concepto de DLC para dar Telefonía por Cable

Como se dijo anteriormente, el DLC colecta llamadas de un número de líneas y las ubica en una misma línea hacia el switch. En un sistema de cable esta función es implementada por el HDT (Host Digital Terminal). El HDT mira hacia el switch como si fuera un concentrador de la misma forma que lo hace el DLC, pero las interfaces que van hacia los teléfonos miran hacia la red HFC[14].

Se necesita realizar una transformación entre la señal que viene de una red HFC y las que se necesitan para que el teléfono analógico funcione. El dispositivo encargado de realizar esta adecuación se ubica en la casa del cliente y es llamado NID[15].

La señal necesita ser dividida en el NID para poder proveer otros servicios. Para ello se coloca un splitter dentro del NID o fuera del mismo. Es posible y recomendable colocar un filtro pasa altos para los otros servicios de cable en orden de no agregar más ruido a la señal.

[14]Red Híbrida
[15]Nework interface device

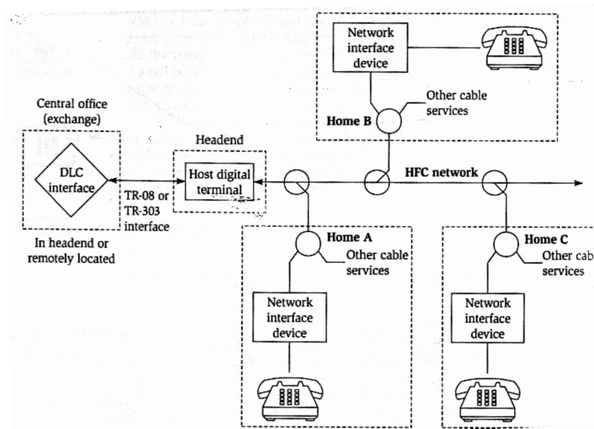

Figura A.7: Modificación de la arquitectura de DLC para dar Telefonía por Cable

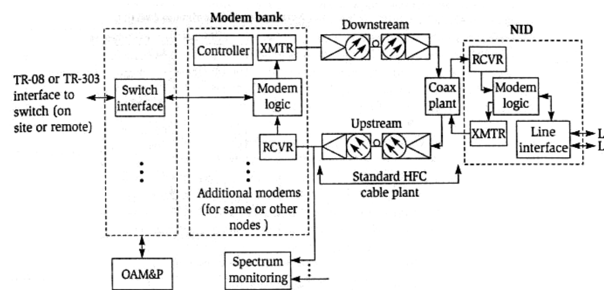

Figura A.8: Elementos del sistema telefónico

Telefonía IP

La tecnología IP esta generando un gran interés en las empresas de cable como alternativa al servicio convencional de telefonía.

El mismo es visto como un servicio de bajo costo para grandes distancias con una interfaz local. Donde se utiliza Internet para evitar los costos de larga distancia.

Las señales de voz son digitalizadas, comprimidas y enviadas como paquetes de datos por la red IP. En la práctica se presentan algunas limitaciones, como por ejemplo, que solo puede hablar uno de los lados por ves.

El sistema de utilizado por las empresas de cable para dar servicio de VoIP es llamado Cilente-Servidor.

Los elementos del sistema son:

- Softswitch (servidor) que es el encargado de establecer y terminar la llamada. El mismo no interviene durante el transcurso de la llamada en sí.
- Router, es el mismo que maneja otros paquetes de datos desde y hacia los clientes.

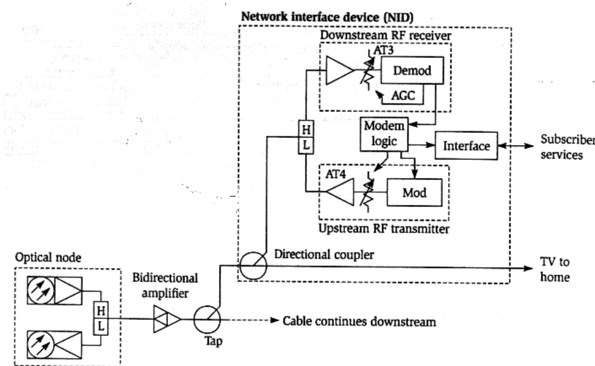

Figura A.9: Interfaz telefónica a nivel del usuario

- Gateway del cliente, que es un cable módem con circuitos especiales para conectar los teléfonos.

La señal de voz es multiplexada con otros datos, frecuentemente pero no siempre usando TDM/TDMA. Esta es la primer simplificación de VoIP, comparada con la actual tecnología de telefonía por cable, ya que no se necesitan circuitos separados para conectar a los clientes. De hecho los datos de voz recorren el mismo circuito que los otros datos. [7]

A.4. Conclusión

Los proveedores a nivel mundial esta optando por la venta de paquetes de servicios para asegurar la fidelidad de los clientes. Los servicios más atractivos para los clientes parecen ser video, voz y datos. Los operadores para poder ofrecerlos necesitan realizar cambios a sus redes actuales, que no se encuentran diseñadas y optimizadas para este propósito.

Alcatel, suministrador de este tipo de servicios, propone una solución para la red de acceso que se apoya en tecnologías como IP Multicast, MPLS, VPLS, etc para la correcta implantación de una red Triple Play que garantice buenos niveles de QoS.

El interés por los servicios de convergencia es una constante en los últimos encuentros de telecomunicaciones de Latinoamérica, como parte de una tendencia mundial que parece irreversible. Telcos, "cableras", proveedores de servicios de Internet y hasta las compañías eléctricas se ven ahora como posibles rivales en un mercado se suma complejidad y dispares protagonistas.

Apéndice B

Descripción de las clases implementadas

Las clases implementadas están organizadas en 4 paquetes. A continuación se realiza una descripción de las clases junto a sus métodos principales. Obs: Los métodos y los atributos son precedidos por un + si son públicos o − en caso de ser privados.

B.1. Package Topología

El paquete **Topología** está conformado por aquellas clases que describen la topología de la red, éstas son **Nodo** y **Enlace**. Además éste package, contiene la clase cuyas instancias representa los diferentes caminos definidos en la red, la clase **Ruta**. También consideramos adecuado incluir en éste paquete, las clases que representan los distintos agentes (quienes manejan los eventos) en la red.

B.1.1. Clase Agente

Es una clase abstracta de la que heredan todas las clases que manejen flujos. Éstas clases son: ServidorStreaming, ServidorElastico, Enlace, Nodo, ClienteStreaming, y ClienteElastico.

Atributos

- - *contadorDeAgentes:* Entero que se incrementa al crear un nuevo elemento de la clase Agente.
- - *nroDeAgente:* Entero que identifica una instancia de la clase **Agente** en la red.

Tiene dos métodos: + *equals(o Object)*[1] y + *getNroDeAgente()*.
Además cuenta con dos procedimientos, éstos del tipo abstracto, que deben ser implementados por las clases herederas. Los métodos mencionados son: *chequeo()* y *errorEnChequeo()*.

[1]Compara dos Agentes según su número identificatorio

B.1.2. Clase ClienteElastico

Clase que hereda de la clase **Agente**. Una instancia de esta clase, representa un conjunto de usuarios de la red que solicitan a la misma únicamente flujos del tipo elásticos. Los flujos solicitados por un mismo cliente elástico varían en su tamaño, ya que éste es sorteado al realizar cada petición.

Atributos

- - *contadorDeClientes:* Variable de clase, es usada para asignar el **nroCliente** una vez creada la instancia. Se actualiza con cada instancia de la clase.
- - *nroCliente:* Entero que identifica a cada cliente en la red
- - *nodoAsociado:* Cada cliente es asociado a un elemento de la clase **Nodo** con el fin de ubicarlo geográficamente en la red.
- + *nombre:* String que identifica al cliente en la red, éste atributo es seteado por el usuario al momento de la inicialización.
- - *mediaDePeticiones:* Parámetro para sortear el tiempo de envío de una solicitud de un flujo, también es determinado por el usuario.
- - *mediaDeTamanio:* Parámetro para sortear el tamaño de un flujo a solicitar (definido al crear la instancia correspondiente).
- - *elasticosRecibidos:* Entero que almacena la cantidad de flujos elásticos recibidos.
- - *trazaHabilitada:* Atributo boolean que toma el valor TRUE si el usuario desea que la instancia correspondiente expulse trazas.
- - *traza:* variable usada para trazar.

Operaciones

Para simplificar se omiten las operaciones de retorno de parámetro (*get()*). También se omite el método *toString()*.

- + *asociarNodo(Nodo nodo):* Método usado para asociar al cliente un nodo de la red y así ubicarlo en la misma.
- + *realizarAccion(Evento evento*:) Este método se invoca al terminar el envío de un flujo del tipo **FlujoElastico**[2]. El cliente obtiene el flujo, lo elimina de la lista de flujos que se están enviando e incrementa la variable **elasticosRecibidos**.
- + *comenzarAOfertar(double tiempoEvento, ServidorElastico agenteDestino):* Indica al cliente que debe realizar una petición de un flujo. Llama al método **generarPeticion(double tiempoEvento, ServidorElastico agenteDestino)**.

[2]Se chequea previamente que el método recibido como parámetro por referencia sea de la clase **TerminaElastico**

- - *generarPeticion(double tiempoEvento, ServidorElastico agenteDestino):* El cliente sortea un tiempo aleatorio con distribución exponencial. También sortea aleatoriamente el tamaño del flujo a solicitar, estos dos sorteos los realiza según sus variables: **mediaDePeticion** y **mediaDeTamanio**. Con estos parámetros construye un evento de la clase **SolicitarElastico** cuyo tiempo de evento es el sorteado más el tiempo actual (que es pasado como parámetro por referencia).

- *+ chequeo():* Método que debe ser implementado en ésta clase ya que está definido como abstracto en la clase **Agente**. El procedimiento es utilizado para chequear si la instancia creada tiene asociada un elemento de la clase **Nodo**. Él método es invocado desde la clase **Topología**, antes de correr la simulación. En caso de que alguno de los clientes no tenga un nodo asociado, la simulación es abortada.

- *+ errorEnChequeo():* Método encargado de imprimir en pantalla un mensaje de error en caso de que algún cliente no tenga un nodo asociado. Trabaja en conjunto con el método anterior.

- *+ habilitarTraza():* Procedimiento que se instancia al inicio de la simulación en el caso de que se configure que la instancia correspondiente debe expulsar datos hacia un archivo de texto con su nombre. En este método es donde se setea la variable **trazaHabilitada**.

- *+ printTraza():* Si el atributo **habilitarTraza** = TRUE, éste método es el encargado de escribir el parámetro **duracion** de cada flujo momento de recibirlo.

Constructores

- *+ ClienteElasico():* Constructor por defecto.

- *+ ClienteElastico(String nombre, double mediaDePeticiones,double mediaDeTamanio)*

B.1.3. Clase ClienteStreaming

Clase que hereda de la clase Agente. Las instancias de la clase **ClienteStreaming** representan usuarios que solicitan flujos Streaming, como es el caso de flujos de voz y video.

Atributos

- - *contadorDeClientes:* Variable de clase, es usada para asignar el **nroCliente** una vez creada la instancia. Se actualiza con cada instancia de la clase.

- - *nroCliente:* Entero que identifica a cada cliente en la red.

- - *nodoAsociado:* Usado para ubicar geográficamente al cliente en la topología.

- *+ nombre:* String que identifica al cliente en la red. Es definido por el usuario del simulador en el momento de la inicialización.

- \- *mediaDePeticiones:* Parámetro para sortear el tiempo de envío de una solicitud de un flujo.
- \- *mediaOfertas:* Parámetro para sortear el valor de la oferta.
- \- *mediaDuracion:* Parámetro para sortear el la duración del flujo tipo Streaming a solicitar.
- \- *rate:* Cada Cliente Streaming tiene asociada una tasa, que es seteada al crearlo. Los flujos solicitados deben ser enviados a dicho rate.
- \- *streamingRecibidos:* Entero que almacena la cantidad de flujos streaming recibidos.

Obs: Los campos **mediaDePeticiones**, **mediaOfertas**, **mediaDuracion** y **rate** son seteadas por el usuario.

Operaciones

Para simplificar se omiten las operaciones de retorno de parámetro (*get()*). También se omite el método *toString()*.

- *+ asociarNodo(Nodo nodo):* Método usado para asociarle al cliente un nodo de la red y así ubicarlo en la misma.
- *+ realizarAccion(Evento evento*:) Este método se invoca al terminar el envío de un flujo del tipo **FlujoStreaming**. El cliente obtiene el flujo, lo elimina de la lista de flujos que se están enviando e incrementa la variable **streamingRecibidos**. Además realiza una nueva petición de otro flujo invocando al método **generarPeticion(double tiempoEvento, ServidorStreaming agenteDestino)**.
- *+ comenzarAOfertar(double tiempoEvento, ServidorStreaming agenteDestino):* Indica al cliente que debe realizar una petición de un flujo. Llama al método **generarPeticion(double tiempoEvento, ServidorStreaming agenteDestino)**.
- *- generarPeticion(double tiempoEvento, ServidorElastico agenteDestino):* El cliente sortea un tiempo aleatorio con distribución exponencial. También sortea aleatoriamente la duración del flujo a solicitar (también con distribución exponencial), estos dos sorteos los realiza según sus variables: **mediaDePeticion** y **mediaDuracion**. Además sortea la oferta de acuerdo a una distribución uniforme. Con estos parámetros construye un evento de la clase **SolicitarStreaming** cuyo tiempo de evento es el sorteado más el tiempo actual (que es pasado como parámetro por referencia).
- *+ chequeo():* Método que debe ser implementado en ésta clase ya que está definido como abstracto en la clase **Agente**. Él método es invocado desde la clase **Topología**, antes de correr la simulación. En caso de que alguno de los clientes no tenga un nodo asociado, la simulación es abortada.
- *+ errorEnChequeo():* Método encargado de imprimir en pantalla un mensaje de error en caso de que algún cliente no tenga un nodo asociado. Ésta operación es invocada si el método anterior retorna FALSE.

Constructores

- + *ClienteStreaming():* Constructor por defecto.
- + *ClienteStreaming(String nombre, double mediaDePeticiones,double mediaOfertas, double rate, double mediaDuracion)*

B.1.4. Clase Nodo

Clase que hereda de la clase **Agente**. Las instancias de ésta clase representan elementos de interconexión de la red, interconectan enlaces entre si formando la topología deseada por el usuario. Cada cliente, cada servidor y cada gestor están asociado a un objeto de la clase **Nodo**.

Atributos

- - *contadorDeNodos:* Variable de clase que se incrementa al crear una instancia de la clase **Nodo**. Se utiliza para asignar el campo **nroDeNodo**.
- - *nroDeNodo:* Entero que identifica a la instancia de **Nodo** en la red.
- + *nombre:* String que representa al nodo, es seteado por el usuario al momento de definirlo.

Operaciones

Contiene un método que permite obtener el número identificatorio. Además contiene un método *toString()*.

Constructores

- + *Nodo()*
- + *Nodo(String nombre)*

B.1.5. Clase Enlace

Las instancias de la clase Enlace representan los elementos de la red que hacen posible la transmisión de los flujos en la misma. Cada enlace (instancia de la clase Enlace) estará asociado a dos instancias de la clase **Nodo** que le darán el sentido al mismo. En esta implementación consideraremos enlaces unidireccionales. A continuación se describen los campos principales así como también sus métodos más importantes. Se excluyen los métodos de acceso y definición de parámetros (*get()* y *set()*).

Atributos

- - *capacidad:* Cada instancia de la clase enlace conoce su capacidad de transmisión.
- + *capacidadStreaming:* Es un porcentaje de la capacidad total del enlace destinado para transportar flujos streaming.

- - *porcentageStreaming:* Variable seteada por el usuario del simulador que determina el valor de **capacidadStreaming**.
- - *nodoInicio:* Instancia de la clase **Nodo** que junto con el nodo fin le definen el sentido a la instancia de la clase **Enlace**.
- - *nodoFin:* Idem que anterior.
- - *contadorDeEnlaces:* Variable de clase usada para asignar el **nroDeEnlace**.
- - *nroDeEnlace:* Entero que identifica unívocamente al enlace en la red.
- + *nombre:* String que identifica al enlace, es fijado por el usuario del sistema al crear un enlace.
- + *listaAuxiliar:* LinkedList que contiene todos los flujos pertenecientes a la clase **FlujoElastico** que se están enviando. Es un atributo usado en el algoritmo de **Max - Min Fairness**.
- + *capacidadLibreMaxMin:* Variable usada en el algoritmo **Max - Min Fairness** que representa la capacidad destinada a repartir entre los flujos elásticos que se están enviando. Al crear una instancia de la clase **Enlace**, éste parámetro tiene el mismo valor que la capacidad total de la misma, se va actualizando en cada comienzo (o fin de envío) de un flujo perteneciente a la clase **FlujoStreaming**.
- + *capacidadMaxMin:* Variable usada únicamente en el algoritmo de asignación de tasas de transmisión.
- + *estaSaturado:* Variable boolean usada internamente en cada iteración del **Max- Min Fairness** que toma el valor TRUE cuando la capacidad del enlace ya se repartió completamente. En cada comienzo del algoritmo mencionado se la inicializa en FALSE.
- + *rateSolicitado:* Atributo usado para la implementación de la política de tarifación implementada. Es el rate equivalente a la cantidad de ofertas aceptadas en cada paso del algoritmo que implementa los remates.
- + *precioActual:* Al igual que la variable anterior, es usada para determinar que ofertas aceptar para obtener la máxima ganancia. A partir de **rateSolicitado**, el **precioAnterior** y la capacidad remanente, el enlace determina el precio en cada iteración.
- + *precioAnterior:* Idem que el anterior.
- *capacidadRestanteParaRemates:* Diferencia entre **rateSolicitado** y la capacidad remanente.
- - *trazaHabilitada:* Variable boolean que determina si la instancia de la clase **Enlace** desplegará diferentes trazas a lo largo de la simulación.
- - *traza:* Usada para escribir el archivo con las trazas correspondientes.

Operaciones

Se omiten los métodos de asignación y obtención de parámetros.

- + *compararNombre(String nombre):* Método que compara dos elementos de la clase String (usado para comparar dos enlaces según el parámetro **nombre**, en caso de ser iguales devuelve TRUE.

- + *calcularPrecio(double pasoDeIteracion):* Método utilizado en la implementación de la "subasta". Es invocado desde la clase **ControladorDeGestores** en uno de los pasos del algoritmo correspondiente.

- + *habilitarTraza():* Método encargado de setear la variable **trazaHabilitada** en TRUE, crea un archivo con el nombre del enlace, y nombra las columas que se van a escribir a lo largo de la simulación.

- + *printTraza():* Método encargado de escribir en el archivo de texto mencionado anteriormente, algunas variables de interés de la instancia de la clase **Enlace** en cuestión. El método es invocado al final de cada "remate".

- + *cheque():* Método aplicado a todas las instancias de la clase **Enlace**, que devuelve FALSE en caso de constatar que alguno de los nodos extremos no haya sido asociado.

- + *errorEnChequeo():* Al igual que el procedimiento anterior, deben ser implementados por ser métodos abstractos en **Agente**. Éste es el encargado de desplegar un mensaje de error en caso de existir.

Constructores

- + *Enlace()*

- + *Enlace(double capacidad, double porcentageParaStreaming, String nombre)*

B.1.6. Clase Ruta

Clase que representa todos los caminos definidos en la red. Consta de una lista de nodos y una lista de enlaces. Para crear una ruta, se debe indicar la lista de enlaces que la conforman, se supone además que dichos enlaces se indican ordenados dándole un sentido a la misma.

En el constructor, se chequea la continuidad de los enlaces. En caso de no ser continua se tira una excepción[3].

Atributos

- + *nombre:* String asignado por el usuario al momento de crear la ruta. Es identificatorio de la misma.

- - *enlaces:* LinkedList que contiene todos los enlaces que conforman la ruta ordenados según el sentido de la misma, se debe pasar como parámetro al crear cada ruta.

- - *nodos:* LinkedList que contiene todos los nodos que conforman la ruta ordenados según el sentido de la misma.

[3]Por más información ver Package Exceptions

- *- nodoOrigen:* Instancia de la clase **Nodo**, que representa el comienzo de cada instancia de la clase **Ruta**. Como se observa en el constructor, es necesario definirlo al crear cada instancia.
- *- precioDeRuta:* Atributo usado en el algoritmo de la política de tarifación aplicada.

Operaciones

Se omiten los métodos de seteo y obtención de parámetros. También ésta clase cuenta con el método *toString()*.

- *+ existeRuta(Nodo nodoOrigen, Nodo nodoDestino):* Método que devuelve TRUE en el caso que exista una ruta creada con el **nodoOrigen** como comienzo y **nodoFin** como nodo fin de la misma.
- *+ calcularPrecioDeRuta():* Método usado para el algoritmo de los remates. Consiste en hallar el precio de la ruta sumando los precios de todos los enlaces que la componen.

Constructores

- *+ Ruta()*
- *+ Ruta(LinkedList¡Enlace¿enlaces,Nodo nodoOrigen, String nombre)throws RutaNoContinua*

B.1.7. Clase ServidorElastico

Clase heredera de la clase Agente. Las instancias de ésta clase son las encargadas de crear los flujos del tipo **FlujoElastico** cuando son solicitados por algún cliente.

Atributos

- *- nroDeServidorElastico:* Entero representativo de cada instancia creada.
- *- contadorDeServidores:* Variable de clase que se incrementa con cada objeto nuevo de la clase **ServidorElastico**
- *- contadorFlujosEnviados:* Atributo usado para almacenar la cantidad de flujos que fueron enviados por cada instancia.
- *- nodoAsociado:* Objeto de la clase **Nodo** asignado por el usuario con el fin de ubicarlo geográficamente en la red.
- *+ nombre:* String definido por el usuario al momento de la inicialización.

Operaciones

Existen operaciones de obtención de parámetros que no se detallarán. Además está implementado el método **toString()** para la clase.

- *+ asociarNodo(Nodo nodoAsociado):* Método usado para asociar la instancia de la clase **Nodo** tal como se comentó a anteriormente.

- *+ generarFlujo(SolicitarElastico evento):* Método invocado en el procedimiento **realizarAcción()** de la clase **SolicitarElastico**. Como ya se explicó, las instancias de la clase **SolicitarElastico** son creadas por objetos de la clase **ClienteElastico** al momento de solicitar un flujo. La política implementada para flujos elásticos, es siempre aceptar toda solicitud. La tasa de los mismos se adecua a la cantidad de otros flujos que comparta los recursos de la red. El método en cuestión, a partir del tamaño solicitado por el cliente, crea una instancia de la clase **FlujoElastico**, chequea si existe una ruta definida desde él hasta el cliente y en caso de encontrarla se la asocia al flujo creado (En caso contrario despliega un mensaje en pantalla dado por la excepción denominada **DestinoNoAlcanzable**). La búsqueda se realiza en el conjunto de todas las rutas definidas[4]. Si la ruta deseada no existe se tira una excepción, de lo contrario, si todo está bien, genera el evento de comienzo de envío del correspondiente flujo. El tiempo asociado a dicho evento es el tiempo global en ese momento.
- *+ equals(Object o):* Procedimiento usado para comparar dos instancias de la clase según el **nroDeServidorElastico**.

Al heredar de la clase **Agente**, también tiene implementados los métodos *chequeo()* y *errorEnChequeo()* invocados antes de comenzar con la simulación.

Constructores

- *+ ServidorElastico(String nombre)*
- *+ ServidorElastico()*

B.1.8. Clase ServidorStreaming

Clase heredera de la clase Agente. Las instancias de ésta clase son las encargadas de crear los flujos del tipo **FlujoStreaming** cuando son solicitados por algún cliente y cuyas ofertas son aceptadas después de aplicar la política de tarifación implementada.

Atributos

- *- nroDeServidorStreaming:* Entero representativo de cada instancia creada.
- *- contadorDeServidores:* Variable de clase que se incrementa con cada objeto nuevo de la clase **ServidorStreaming**
- *- contadorFlujosEnviados:* Atributo usado para almacenar la cantidad de flujos que fueron enviados por cada instancia.
- *- nodoAsociado:* Objeto de la clase **Nodo** asignado por el usuario con el fin de ubicarlo geográficamente en la red.
- *- cacheDeRutas:* LinkedList en la que se almacenan las rutas usadas. Con el fin de optimizar la búsqueda en posteriores envíos de flujos dirigidos a destinos repetidos.
- *+ nombre:* String definido por el usuario al momento de la inicialización.

[4]Rutas conocidas por la clase Topología

Operaciones

Existen operaciones de obtención de parámetros que no se detallarán. Además está implementado el método **toString()** para la clase.

- *+ asociarNodo(Nodo nodoAsociado):* Método usado para asociar la instancia de la clase **Nodo** tal como se comentó a anteriormente.
- *+ comenzarEnvioStreaming(double tiempoInicio,ClienteStreaming agenteDestino, double duracion):* El método en cuestión, a partir de la duración solicitada por el cliente, crea una instancia de la clase **FlujoStreaming**, chequea si existe una ruta definida desde él hasta el cliente y en caso de encontrarla se la asocia al flujo creado (En caso contrario despliega un mensaje en pantalla dado por la excepción denominada **DestinoNoAlcanzable**). Si todo está bien, genera a su vez el evento de comienzo de envío del correspondiente flujo[5]. Al tratarse de un flujo streaming, éste debe ser transmitido a una tasa prefijada por el cliente. El tiempo asociado a dicho evento es el tiempo global en ese momento.
- *- devolverRuta(Nodo nodoOrigen, Nodo nodoDestino):* Método que realiza la búsqueda de una ruta en el caché del servidor que tenga como nodo inicio a la instancia**nodoOrigen** y como nodo fin a **nodoDestino**. Es un método invocado a la hora de asignarle una ruta al flujo solicitado.

Análogamente a la clase anterior, se implementaron los método de verificación previos a dar comienzo a la simulación.

Constructores

- *+ ServidorStreaming(String nombre)*
- *+ ServidorStreaming()*

B.1.9. Clase ControladorDeGestores

Al crear un objeto de la clase **Gestor** se debe crear inmediatamente un elemento de la clase **ControladorDeGestores**. El simulador está diseñado para que exista un único Controlador de Gestores en la red. Este elemento, tal como el nombre de la clase lo indica, es el encargado de darle órdenes a los distintos gestores presentes en la red. Es además el encargado de crear eventos de la clase **LlamadaAControladorDeGestores**.

Atributos

- *- tiempoEntreSorteos:* Tiempo configurable por el usuario que define el tiempo en que se realizan los remates.
- *+ listaDeGestores:* LinkedList que contiene todos los gestores definidos en la red.
- *- pasoDeIteracion:* Define el paso de iteración del algoritmo aplicado, también es definido por el usuario al comienzo de la simulación.

[5]instancia de la clase ComienzaStreaming

Operaciones

- *+ agregarGestor(Gestor gestor):* Cada vez que un elemento de la clase **Gestor** es creado, inmediatamente se agrega a la lista de gestores.
- *+ comenzarAlgoritmo():* Es el método encargado de indicarle a los gestores que apliquen la política de tarifación. La política implementada consiste en remates que tienen como objetivo quedarse con las ofertas que den al operador la mayor ganancia. Además al ejecutarse este método, se crea otra instancia de la clase **LlamadaAControladorDeGestores**. La política implementada consta de una etapa de inicialización donde se setean, entre otras cosas, los precios a cero[6], se le pide a los gestores que calculen la derivada de su función utilidad, se setean a cero el número de ofertas aceptadas por todos los gestores, etc. En el primer paso del algoritmo, cada gestor, según su curva de derivada de utilidad y el precio de la ruta que gestiona, decide cuantas ofertas acepta[7]. Se le pide a los enlaces que calculen su precio para la cantidad de ofertas aceptadas, se calcula el precio de la ruta [8] y se vuelve al primer paso. Esto ocurre hasta llegar a la solución más óptima posible que es cuando todos los enlaces han bajado su precio.

Constructor

- *+ ControladorDeGestores(double tiempoEntreSorteos, double pasoDeIteracion)*

B.1.10. Clase Gestor

Las instancias de esta clase son las encargadas de aplicar la política de tarifación implementada[9]. Cada elemento perteneciente a esta clase gestiona una ruta determinada y un rate único[10]. Los gestores no son elementos de la topología pero, igual, hay que asociarlos a un nodo de la red.

Atributos

- *- listaDeOfertas:* LinkedList que contiene todas las ofertas solicitadas por los clientes, instancias de la clase **ClienteStreaming**, ordenadas de mayor a menor oferta.
- *- listaDeOfertasOrden:* LinkedList auxiliar usada para ordenar la lista de ofertas.
- *- listaDeOfertasAux:* Idem que la anterior
- *nodoAsociado:* Instancia de la clase **Nodo** asociada a cada elemento de la clase gestor.
- *+ nombre:* Variable seteada por el usuario para referirse a cada elemento de la clase **Gesto**.

[6]Precios de Ruta y Precios de Enlaces

[7]En el primer paso del algoritmo, al estar inicializado el precio de las rutas en cero, las ofertas aceptadas son todas las solicitadas.

[8]Suma de los precios de los enlaces que la componen

[9]Remates

[10]El usuario debe crear tantos gestores como rutas tenga, siempre que en éstas circulen flujos streaming. Además, no solo un gestor por ruta, sino que se deberá crear un gestor distinto por cada rate que pueda ser solicitado.

- - *nroGestor:* Entero identificatorio de cada gestor en la red.
- + *contadorDeGestor:* Variable de clase usada para asignar el **nroGestor**. Se incrementa con la creación de cada instancia.
- - *rutaAsociada:* Ruta que gestiona.
- - *rateAsociado:* Tasa que gestiona. nombre: String que identifica al gestor en la red, es ingresado por el usuario al momento de crearlo.
- + *utilidadDerivada:* LinkedList que representa los valores de la derivada de la curva de utilidad. Ésta última está determinada por la lista ordenada de las ofertas en función del rate.
- + *cantidadDeOfertasAceptadas:* Entero que almacena la cantidad de ofertas aceptadas en cada remate.
- + *gananciaTotal:* Variable que almacena la ganancia neta del gestor a lo largo de la simulación.
- + *ganancia:* Es la ganancia de cada instancia de la clase, resultante de aplicar el algoritmo de Remates.
- + *trazaHabilitada:* Variable boolean que toma el valor TRUE si se desea que el objeto imprima las trazas programadas.

Operaciones

Se omiten los métodos de seteo y obtención de parámetros.

- + *llegaOferta(SolicitarStreaming oferta):* Método invocado cuando se da un evento perteneciente a la clase **SolicitarStreaming**. Ésta operación cumple la función de llamar al método **agregarOferta(SolicitarStreaming evento)**
- + *agregarOferta(SolicitarStreaming evento):* Ingresa el evento pasado por referencia en la **listaDeOfertas**. Al ingresar el mismo, la lista queda ordenada según la oferta de cada solicitud (de mayor a menor).
- + *asociarNodo(Nodo nodo):* Le asocia al gestor un elemento de la clase **Nodo**, el nodo asociado debe pertenecer a la ruta gestionada por la instancia en cuestión.
- + *calcularRate():* Método usado para recorrer la curva de la derivada de la utilidad y dependiendo del precio de la ruta gestionada, seleccionar cuantas y cuales ofertas aceptar.
- + *derivarUtilidad():* A partir de la lista de solicitudes, ordenadas de mayor a menor oferta, se obtiene la curva de utilidad de cada instancia de la clase gestor. Esta curva es distinta para cada remate. Con éste método se obtiene la curva que representa la derivada de la utilidad[11].

[11] Por más detalles dirigirse a la *Parte IV* del algoritmo donde se explica la implementación del algoritmo

- *+ comenzarEnvioStreaming():* Luego de realizado el remate, cada gestor sabe cuantas ofertas debe dejar pasar. Éste método es el encargado de solicitar a las instancias de la clase **ServidorStreaming** que comiencen el envío de los flujos que ganaron la subasta. Además, para aquellos clientes que ofertaron y sus ofertas no fueron suficientes, se les manda de regreso un evento de la clase **OfertaInsuficienteParaStreaming**.

Ésta clase cuenta al igual que clases anteriores con la implementación de un método que verifica que las condiciones de arranque de la simulación estén correctas, éste es: *chequeo()*.
Además cuenta con los método que hacen posible la impresión de trazas. Las instancias que tengan el campo **habilitarTraza** en TRUE, crearán un archivo de texto con el nombre de la misma en el que se grabarán los valores: *tiempo de remate, ganancia del remate, ganancia neta y cantidad de ofertas aceptadas en el remate*. **Constructores**

- *+ Gestor()*
- *+ Gestor(String nombre)*

B.1.11. Clase Topologia

Clase que conoce todos los elementos de la topología de la red. Contiene todos los enlaces, todas las rutas, todos los nodos, los servidores y clientes.

Atributos

- *+ enlacesDeLaRed:* LinkedList constituida por todos los enlaces de la red. Cada enlace, al momento de crearlo, se lo asocia a esta lista.
- *+ nodosDeLaRed:* LinkedList constituida con todos los nodos de la red. Cada nodo, al momento de crearlo (al igual que los enlaces), se lo asocia a esta lista.
- *+ servidoresStreaming:* idem pero para servidores, instancias de la clase **ServidorElastico**.
- *+ servidoresElasticos:* Idem pero para servidores generadores de flujos del tipo streaming.
- *+ clientesElastico:* idem pero para clientes destinados a solicitar y recibir flujos que simulen el tráfico de Internet.
- *+ clientesStreaming:* idem pero para clientes destinados a solicitar instancias de la clase **FlujoStreaming**.
- *+ rutasDefinidas:* LinkedList que contiene todas las rutas definidas por el usuario en el momento de la inicialización.

Operaciones

- *+ agregarElemento(Objecto o):* Procedimiento que toma el objeto pasado por referencia y según de que tipo sea lo agrega a la LinkedList correspondiente.

- *+ devolverRuta(Nodo nodoOrigen, Nodo nodoDestino):* Devuelve una ruta que tenga dichos nodos como principio y fin.
- *+ chequeo():* Después de interpretar el archivo de texto donde se definen los datos a simular, se realiza el chequeo de posibles errores de configuración realizados por el usuario. Éste método invoca a los métodos *chequeo()* de cada una de las clases anteriores y en caso de ser necesario se invoca al método *errorEnChequeo()* de la clase que corresponda. Si alguna de las condiciones de simulación no se cumple, la misma es abortada.

Constructor

- *Topologia()*

B.2. Package Eventos

El paquete **Eventos** está conformado por las clases cuyas instancias representan los posibles eventos que dan paso a la simulación. Además, otra clase incluida en éste package, es la clase **ListaDeEventos**.

Todos los constructores de las siguientes clases tienen implementado agregar cada instancia creada a la lista de eventos de la simulación.

B.2.1. Clase Evento

Es una clase abstracta de la que heredan todos los eventos. Tiene un método abstracto que debe ser implementado por todas sus clases herederas(**+ realizarAccion()**). Además tiene definidos los métodos *+ equals()* y *+ compareTo()*.

B.2.2. Clase ComienzaElastico

Clase que hereda de la clase Evento. Ésta clase representa el comienzo del envío de un flujo del tipo Flujo Elástico. Las instancias de ésta clase son creadas por objetos de la clase **ServidorElastico**.

Atributos

- *- agenteOrigen:* Es el Agente que lo crea, en este caso es un objeto de la clase**ServidorElastico**.
- *- agenteDestino:* Es el cliente que solicitó el flujo elástico implícito en dicho evento.
- *+ flujo:* Objeto de la clase **FlujoElastico** que hace referencia al flujo que comienza a enviarse al ocurrir este evento.

Operaciones

- *+ realizarAccion():* Método encargado de colocar el flujo en la lista de flujos que se están enviando y crear el evento fin correspondiente a dicho flujo llamando al método **generarEventoFin()**. Llama a que se ejecute el

algoritmo de asignación de recursos Max-Min Fairness y luego solicita la actualización de los eventos de la clase **TerminaElastico** presentes en la lista de eventos, pues al cambiar la tasa de envío, los flujos elásticos cambian el tiempo en que terminarán de ser enviados. Además pide al agente destino, un cliente elástico, que vuelva a solicitar otro flujo.

- - *generarEventoFin():* A partir del tamaño del flujo y el rate asignado en ese momento, determina el evento **TerminaStreaming**.
  correspondiente.

Constructor

- *+ ComienzaElastico(double tiempoInicio, FlujoElastico flujo, ServidorElastico agenteOrigen, ClienteElastico agenteDestino)*

B.2.3. Clase ComienzaStreaming

Clase que hereda de la clase Evento. Ésta clase representa el comienzo del envío de un flujo del tipo Flujo Streaming. Las instancias de ésta clase son creadas por elementos de la clase **ServidorStreaming**.

Atributos

- - *agenteOrigen:* Es el Agente que lo crea, en este caso es un objeto de la clase**ServidorStreaming**.

- - *agenteDestino:* Es el cliente que solicitó el flujo streaming implícito en dicho evento.

- *+ flujo:* Objeto de la clase **FlujoStreaming** que hace referencia al flujo que comienza a enviarse al ocurrir este evento.

Operaciones

- *+ realizarAccion():* Método encargado de colocar el flujo en la lista de flujos que se están enviando. Además crea el evento fin correspondiente a dicho flujo, instancia perteneciente a la clase **TerminaStreaming**. Toma la ruta del flujo asociado al evento y actualiza la capacidad ocupada por los flujos streaming en todos los enlaces de dicha ruta. Llama a que se ejecute el algoritmo de asignación de recursos Max-Min Fairness y luego solicita la actualización de los eventos de la clase **TerminaElastico** presentes en la lista de eventos.

Constructor

- *+ ComienzaStreaming(double tiempoInicio, FlujoStreaming flujo, ServidorStreaming agenteOrigen, ClienteStreaming agenteDestino)*

B.2.4. LlamadaAControladorDeGestores

Clase que hereda de la clase **Evento**. Las instancias de ésta clase son creadas por los elementos de la clase **ControladorDeGestores**. Al recorrer la lista de eventos, al llegar el turno de un evento de éste tipo, trae como consecuencia realizar los remates.

Atributos

- - *controlador:* Elemento de la clase **ControladorDeGestores**, encargado de crear instancias de esta clase de evento.

Operaciones

- + *realizarAccion():* Método que debe ser implementado como consecuencia de heredar de la clase **Evento**. Éste método llama a la instancia de la clase **ControladorDeGestores**[12] para que ejecute el algoritmo de los remates.

Constructor

- + *LlamadaAControladorDeGestores(double tiempoInicio, ControladorDeGestores controlador).*

B.2.5. Clase SolicitarElastico

Clase que hereda de la clase **Evento**. Las instancias de ésta clase representan las peticiones que realizan los clientes elásticos solicitando flujos a la red.

Atributos

- - *agenteOrigen:* Instancia de la clase **ClienteElastico** que solicitó el flujo.
- - *agenteDestino:* Atributo que guarda el objeto de la clase **ServidorElastico** al que está dirigido el pedido.
- - *tamanio:* Variable que almacena el tamaño (sorteado por el cliente) del flujo solicitado.

Operaciones

Existen procedimientos que posibilitan acceder a los parámetros anteriormente nombrados.

- + *realizarAccion():* Método que ordena al servidor almacenado en el campo **agenteDestino**, que proceda a generar el flujo pedido por el cliente. Par ello llama al método **generarFlujo(SolicitarElastico evento)** de la clase **ServidorElastico**.

Constructor

- + *SolicitarElastico(double tiempoInicio, ClienteElastico agenteOrigen, ServidorElastico agenteDestino,double tamanio)*

[12] Instancia única en la red

B.2.6. Clase SolicitarStreaming

Clase que hereda de la clase **Evento**. Las instancias de ésta clase representan las ofertas que realizan los clientes streaming solicitando flujos a la red.

Atributos

- - *agenteOrigen:* Instancia de la clase **ClienteStreaming** que realizó la oferta.
- - *agenteDestino:* Atributo que guarda el objeto de la clase **ServidorStreaming** al que está dirigido el pedido.
- - *duración:* Variable que almacena la duración (sorteada por el cliente) del flujo solicitado.
- - *oferta:* Atributo que tiene asociada la oferta apostada por el cliente.
- - *nroDeOferta:* Entero que identificada cada instancia de la clase **SolicitarStreaming**
- - *contadorDeOfertas:* Variable de la clase que se incrementa con cada instancia de la misma.

Operaciones

Existen procedimientos que posibilitan acceder a los parámetros anteriormente nombrados.

- *+ realizarAccion():* Método encargado de buscar el gestor destinado a atender el tipo de solicitud, lo busca según la ruta y el rate gestionado. La búsqueda del elemento de la clase **Gestor** se realiza en la ruta correspondiente al flujo solicitado. Una vez encontrado el gestor, se ingresa la solicitud realizada por el cliente a la lista de ofertas. La petición estará en dicha lista hasta que ocurra el próximo remate.
- *+ equals(Object o):* Devuelve TRUE si dos eventos tienen la misma **oferta**.
- *+ compareTo(Object o):* Compara dos eventos pertenecientes a la clase descripta según su **oferta**.
- *+ getRutaAsociada():* Procedimiento encargado de buscar en todas las rutas definidas, aquella que vaya desde el nodo asociado al **agenteDestino** hasta el nodo asociado al **agenteOrigen**.

Constructor

- *+ SolicitarStreaming(double tiempoInicio,ClienteStreaming agenteOrigen,ServidorStreaming agenteDestino,double oferta, double duracion)*

B.2.7. Clase TerminaElastico

Clase que hereda de la clase **Evento**. Las instancias de ésta clase representan las terminaciones de transmisión de los flujos elásticos.

Atributos

- - *agenteOrigen:* Instancia de la clase **ServidorElastico** que genero el flujo asociado al evento.

- - *agenteDestino:* Atributo que guarda el objeto de la clase **ClienteElastico** al que está dirigido el pedido.

- - *flujo:* Variable que guarda la instancia de la clase **FlujoElastico** asociada.

Operaciones

Como se observa, todos los atributos son privados, es por eso que se implementaron métodos para acceder a los mismos, dichos método no se detallarán. Otro método implementado es el *toString()*.

- *+ realizarAccion():* Método encargado de pedirle al **agenteDestino** que saque el flujo que terminó de enviarse de la lista de flujos de flujos de la red. Además el cliente actualiza el contador de flujos recibidos. Después de realizado lo anterior, se invoca al método de **Max - Min Fairness** y luego se llama a que se actualicen las instancias de la clase **TerminaElastico** aún presentes en la red.

Constructor

- *+ TerminaElastico(double tiempoInicio,FlujoElastico flujo, ServidorElastico agenteOrigen, ClienteElastico agenteDestino)*

B.2.8. Clase TerminaStreaming

Clase que hereda de la clase **Evento**. Las instancias de ésta clase representan las terminaciones de transmisión de los flujos streaming, éstas son determinadas al comienzo del envío de los respectivos flujos y no son modificadas a lo largo de la simulación. Contrariamente a lo que pasa con las instancias de la clase **TerminaElastico**.

Atributos

- - *agenteOrigen:* Instancia de la clase **ServidorStreaming** que genero el flujo asociado al evento.

- - *agenteDestino:* Atributo que guarda el objeto de la clase **ClienteStreaming** al que está dirigido el pedido, es uno de los clientes ganadores de uno de los remates.

- - *flujo:* Variable que guarda la instancia de la clase **FlujoStreaming** asociada.

Operaciones

Como se observa, todos los atributos son privados, es por eso que se implementaron métodos para acceder a los mismos, dichos método no se detallarán. También se implementó el método *toString()*.

- *+ realizarAccion():* Método encargado de pedirle al **agenteDestino** que saque el flujo que terminó de enviarse de la lista de flujos de flujos de la red. Además el cliente actualiza el contador de flujos recibidos y realiza una nueva solicitud de otro flujo. Después de realizado lo anterior, se actualizan las capacidades**Se actualiza la capacidad a repartir por los flujos elásticos** de los enlaces pertenecientes a la ruta del flujo enviado, se invoca al método de **Max - Min Fairness** y luego se llama a que se actualicen las instancias de la clase **TerminaElastico** aún presentes en la red.

Constructor

- *+ TerminaStreaming(double tiempoInicio,FlujoStreaming flujo,ServidorStreaming agenteOrigen, ClienteStreaming agenteDestino)*

B.2.9. Clase OfertaInsuficienteParaStreaming

Clase que representa un tipo de evento de la simulación, por lo tanto hereda de la clase abstracta **Evento**. Las instancias de ésta clase son originadas por los gestores existentes en la red. Se crean para avisarle a aquellos clientes que su oferta fue rechazada.

Atributos

- *- agenteOrigen:* Elemento de la clase **ClienteStreaming** que realizó la oferta que fue rechazada.

- *- agenteDestino:* Elemento de la clase **ServidorStreaming**, que representa el servidor destinado a originar el flujo solicitado, en caso de que el pedido hubiese sido aceptado.

Operaciones

- *+ realizarAccion():* Método encargado de pedirle al cliente que vuelva a solicitar otro flujo.

Constructor

- *+ OfertaInsuficienteParaStreaming(double tiempoInicio, ClienteStreaming agenteOrigen, ServidorStreaming agenteDestino)*

B.2.10. EventoDeFin

Clase que hereda de la clase **Evento**. Se crea una única instancia de esta clase a lo largo de la simulación. Dicho objeto es creado al comienzo de la misma por la clase intérprete del archivo de entrada, la clase **Reader**. El tiempo que se le asocia al evento que determina el fin de la simulación es ingresado por el usuario.

Atributos

- *- agenteFinalizador:* Es una instancia de la clase **Reader** encargada de crear el evento que dará por terminada la simulación.

Operaciones

- *+ realizarAccion():* Como toda clase heredera de la clase **Evento**, debe implementar éste método. En este caso, la acción a realizar es llamar a la instancia de la clase **Reader** que lo generó para que de por finalizada la simulación.

Constructor

- *+ EventoDeFin(double tiempoFinSimulacion, Reader agenteFinalizador)*

B.2.11. Clase ListaDeEventos

Se crea una única instancia de esta clase. El atributo principal de la misma es una **LinkedList** formada por las instancias de la clase **Evento** que desencadenan la simulación.
Atributos
Existen atributos auxiliares que permiten ordenar la lista de eventos de menor a mayor tiempo asociado. Éstos atributos no se detallarán.

- *+ listaDeEventosPrincipal:* LinkedList constituída por elementos de la clase **Eventos**.

Operaciones

- *+ agregarEvento(Evento evento):* Este método agrega el elemento evento (pasado por referencia) a la **listaDeEventosPrincipal** de forma que quede ordenada según el atributo de la clase evento: **tiempoDeInicio**.

- *+ sustituirEvento(Evento eventoOriginal, Evento eventoNuevo):* El método busca el **eventoOriginal** en la lista de eventos, en caso de no encontrarlo se muestra un mensaje de error. Por el contrario, lo elimina y llama al método anterior pasándole el **eventoNuevo** como parámetro.

- *+ removeEvento(Evento evento):* El elemento pasado como parámetro se busca en la lista de eventos y en caso de encontrarlo es eliminado. Par ello se utilizan los métodos **indexOf(¡Object¿)** y **remove(¡int¿)** de la clase LinkedList.

Obs: Existen implementados otros métodos que permiten escribir la lista de eventos.

Constructor

- *+ ListaDeEventos()*

B.3. Package Flujos

Éste paquete contiene las clases que representan los distintos flujos en la red. Como se ha explicado a lo largo del documento, se pretende implementar flujos que simulen el comportamiento de una red con tecnología Triple Play. Es por eso que existen dos grandes clases de flujos: Flujos Elásticos y Flujos Streaming.

B.3.1. Clase Flujo

Clase abstracta de la que heredan dos clases que representan los dos tipos de flujos posibles en la red. Ésta clase contiene en dos listas, una por cada clase de flujo, los flujos que se están enviando en cada momento. Además, para el caso de las instancias de la clase **FlujoElastico**, contiene una lista con los eventos del tipo **TerminaElastico** correspondientes. Es la encargada de actualizar los tamaños que restan por enviar, y por lo tanto los tiempos de fin de los flujos elásticos cada vez que se invoca al algoritmo de Max - Min Fairness.

Atributos

- + *flujosElasticos:* LinkedList que contiene todos las instancias de la clase **FlujoElastico** que se están enviando en cada momento.
- + *listaDeEventosElasticos:* LinkedList formada por los eventos correspondientes a la clase **TerminaElastico** asociados a los flujos presentes en la lista anterior.
- + *flujosStreaming:* LinkedList que almacena los flujos del tipo streaming que se están enviando en la red.
- - *trazaHabilitada:* Variable booleana que toma el valor TRUE cuando se desean imprimir trazas.
- - *trazasElasticos:* Atributo para imprimir las trazas correspondientes a los flujos elásticos enviándose en la red. Se imprime el rate que se lleva cada flujo.
- - *trazasStreaming:* Atributo para imprimir las trazas correspondientes a los flujos streaming enviándose en la red. Se imprime el rate que se lleva cada flujo.

Operaciones

Existen métodos que permiten agregar flujos y eventos en las respectivas LinkedLists, y por lo tanto, también existen métodos que permiten eliminarlos. Éstos no se detallarán.

- + *actualizarListaAuxiliar():* Antes de aplicar el algoritmo de Max - Min Fairness, ésta clase, al ser la única que conoce todos los flujos que están siendo enviados, es la encargada de crear y completar la lista con los flujos elásticos que pasan por cada enlace de la red. Este método recorre la lista de flujos presentes en **flujosElasticos**. Como cada flujo conoce su ruta alcanza con recorrer todos los enlaces de la ruta asociada al flujo y agregarlo a la variable del enlace denominada **listaAuxiliar**.
- + *actualizar():* Es el método invocado enseguida de la ejecución del algoritmo de asignación de tasas a los flujos elásticos. Es el encargado de determinar

el tamaño que resta por enviar de cada flujo elástico, determinar el nuevo tiempo de finalización de envío de cada uno y por lo tanto también actualiza las instancias de la clase **TerminaElastico** presentes en la lista de eventos del simulador.

- *+ habilitarTraza():* Al igual que la clase enlace tiene la posibilidad de habilitar o no la impresión de datos hacia afuera. Éste método es el encargado de habilitar dicha funcionalidad.
- *+ printTrazaElasticos():* Método invocado al ejecutarse **actualizar()** que escribe los datos de rate de cada flujo presente en la red (siempre que la variable trazaHabilitada = TRUE).
- *+ printTrazaStreaming():* Procedimiento que escribe los datos de rate de cada flujo streaming presente en la red (siempre que la variable trazaHabilitada = TRUE).

B.3.2. Clase FlujoElastico

Clase que hereda de la clase **Flujo**. Las instancias de esta clase representan los flujos que no necesitan una tasa fija para ser enviados como lo es el tráfico de Internet.

Atributos

- *+ saturado:* Variable booleana usada en la implementación de Max - Min Fairness para identificar aquellos flujos que en cada paso de la iteración, pasan por algún enlace que ya no juega más en el algoritmo.
- *- rateActual:* Valor que toma la tasa de transmisión luego de aplicado el Max - Min Fairness.
- *- rateAnterior:* Variable usada para almacenar el rate próximo anterior. Es útil a la hora de calcular el tamaño restante por enviar.
- *- tamanio:* Número real que representa el tamaño del flujo en cuestión. Éste valor es sorteado por el cliente al momento de solicitarlo.
- *- tamanioRestante:* Atributo usado para actualizar los eventos del tipo **TerminaElastico** luego de algún cambio en la red[13].
- *- duracion:* Variable determinada a partir del tamaño restante por enviar y el rate asignado.
- *- ruta:* Ruta asociada a la instancia de la clase FlujoElastico.
- *- contadorDeFlujos:* Variable de clase usada para asignarle el **nroDeFlujo** una vez creado un objeto perteneciente a la clase.
- *- nroDeFlujo:* Entero representativo de cada flujo en la red.
- *- tiempoDeInicio:* Variable que almacena el tiempo en que el flujo comenzó a ser enviado.
- *- tiempoDeFin:* Variable que almacena el tiempo en que el flujo terminó de ser enviado.

[13]Cambio en la red: Comienzo y finalización del envío de algún flujo

- - *tiempoUltimaActualizacion:* Variable que almacena el último tiempo en que se actualizó el tamaño restante.
- - *agenteOrigen:* Es una instancia de la clase **ServidorElastico**, es quien genera el flujo.
- - *agenteDestino:* Es una instancia de la clase **ClienteElastico**, es quien solicita el flujo.

Operaciones

Existen métodos que permiten setear y obtener los parámetros anteriores. Para simplificar no se detallarán.

- *+ equals(Object o):* Método que permite comparar dos instancias de la clase según el número de flujo.

Constructor

- *+ FlujoElastico(double tiempoInicio, double tamanio, ServidorElastico agenteOrigen, ClienteElastico agenteDestino)*

B.3.3. Clase FlujoStreaming

Clase que hereda de la clase **Flujo**. Las instancias de esta clase representan los flujos que necesitan una tasa fija para ser enviados. Ejemplos de éstos flujos son los flujos de voz y video.

Atributos

- - *rate:* Variable usada para almacenar el rate que debe ser asignado al flujo en cuestión. Es determinado por el cliente en el momento de solicitarlo.
- - *duracion:* Atributo usado para almacenar la duración del flujo solicitado.
- - *tamanio:* Representa el tamaño de la instancia de la clase, está determinado por la duración y por el rate.
- *ruta:* Ruta asociada a la instancia de la clase FlujoStreaming.
- - *contadorDeFlujos:* Variable de clase usada para asignarle el **nroDeFlujo** una vez creado un objeto perteneciente a la clase.
- - *nroDeFlujo:* Entero representativo de cada flujo en la red.
- - *tiempoDeInicio:* Variable que almacena el tiempo en que el flujo comenzó a ser enviado.
- - *tiempoDeFin:* Variable que almacena el tiempo en que el flujo terminó de ser enviado.

Operaciones

Existen métodos que permiten setear y obtener los parámetros anteriores. Para simplificar no se detallarán.

- *+ equals(Object o):* Método que permite comparar dos instancias de la clase según el número de flujo.

Constructor

- *+ FlujoStreaming(double duracion, double rate)*

B.4. Package Main

Es el paquete conformado por la clase que ejecuta el algoritmo de **Max - Min Fairness**. También están las clases **GeneradorAleatorio**, usada para sortear números con distribución exponencial, **Reader**, y **Principal**. Ésta última contiene el método *main()*.

B.4.1. Clase GeneradorAleatorio

Clase que permite generar números aleatorios con distribución exponencial. Es usada por las instancias de las clases **ClienteElastico** y **ClienteStreaming** a la hora de sortear algunos de los parámetros de los flujos a solicitar.

B.4.2. Clase MaxMinFairness

Se crea una única instancia de ésta clase en la clase **Principal**. Contiene métodos que tiene como objetivo la implementación del algoritmo de **Max - Min Fairness**.

Atributos

- - *delta:* Es el rate que será asignado en cada paso de la iteración a cada instancia de la clase **FlujoElastico** que se está enviando.
- - *enlaceSaturado:* Variable que guarda en cada iteración la instancia de la clase **Enlace** que determinó el **delta**. Dicha variable se almacena con el fin de setearle a todos los flujos, cuya ruta pasa por el Enlace en cuestión, el campo**saturado** en TRUE[14]. Además se le setea a dicho enlace el atributo **saturado** en TRUE.
- - *minimo:* variable interna usada en el método privado **hallarMinimo()**
- - *minimoViejo:* variable usada en el método **hallarMinimo()**, es lo que retorna dicho método. Se usa para incrementar **delta** en cada paso.
- - *termino:* variable boolean usa para dar fin a la iteración, una vez que todos los enlaces están saturados.

Operaciones

La clase contiene un único método público (**asignarRate()**). Éste método invoca a los demás procedimientos para implementar el algoritmo de Max - Min Fairness.

[14]Por más detalles dirigirse a la clase **FlujoElastico**

- *+ asignarRate():* es el método principal. Primeramente inicializa todos los rates de los flujos elásticos de la red en ”0”, y la variable **saturado** de los mismos en FALSE. Además inicializa la **capacidadMaxMin** igual a la **capacidadLibreMaxMin** en ese momento en cada enlace. Como **capacidadLibreMaxMin** se entiende a la porción de la capacidad total del enlace que no está ocupada por las instancias de la clase **FlujoStreaming**. Este método realiza todas las iteraciones necesarias, hasta completar el algoritmo, llamando a los siguientes métodos.

- *- asignarRateIntermedio(double deltaM):* En cada iteración se le asigna el **rate** a cada flujo no perteneciente a un enlace saturado.

- *setearFlujosSaturados():* En cada paso de la iteración, se setea la variable **saturado** en TRUE de los flujos que pertenecen al enlace que saturó en dicho paso.

- *- actualizarCapacidades():* En cada iteración se actualiza la **capacidadMaxMin** de todos los enlaces de la red.

- *- hallarMinimo():* Es el método que compara entre todos los enlaces de la red cuál es el más restrictivo. Devuelve el mínimo de rate a sumar en dicha iteración, además guarda el enlace que determinó dicho mínimo.

Constructor

- *+ MaxMinFairness()*

B.4.3. Clase Principal

Clase que contiene el método **main(String[] args)**. Ésta se encarga de manejar la **listaDeEventos** la cual es visible por todas las clases.

Atributos

- *+ lista:* Variable static y public que es una instancia de la clase ListaDeEventos. Todas las clases pueden accederla.

- *+ tiempoGlobal:* Variable static y public, que se actualiza cada vez que un evento perteneciente a lista llama a su método realizarAccion().

- *+ tiempoFin:* Variable static y public que se setea con el mismo tiempo del evento que finaliza la simulación.

- *+ MaxMin:* Variable static y public se utiliza para instanciar al algoritmo correspondiente.

Operaciones

- *+ ejecutar():* Método encargado de recorrer la lista de eventos, hace que cada evento realice la acción correspondiente.

B.4.4. Clase Reader

Se la puede ver como la clase intérprete del archivo de texto que tiene los datos que definen la simulación. Datos seteados por el usuario.

Es la clase encargada de construir los elementos de la topología así como también los servidores y clientes (usuarios de la red). Además dicha clase es la que define cuando comienza la simulación y cuando termina, definiendo los eventos de fin y dándole la orden a los clientes para que comiencen a solicitar flujos.

Además de crear todo lo que se pide en el .txt, se chequean posibles excepciones.

B.5. Package Exceptions

Dicho paquete contiene dos clases que implementan dos tipos de excepciones a considerar.

B.5.1. RutaNoContinua:

Esta excepción ocurre cuando al crear una ruta se indica un conjunto de enlaces que no forma un camino continuo, es verificado en el constructor de la clase **Ruta** según nodos asociados a cada instancia de **Enlace**.

B.5.2. DestinoNoAlcanzable

Esta excepción ocurre al momento de asignarle la ruta a un flujo, si no existe ninguna ruta que vaya del inicio al fin deseado.

Apéndice C

Plan de Trabajo - Entregado en Mayo de 2007

A continuación se incluye parte del documento entregado en Mayo de 2007 al terminar la primer etapa de planificación.

C.1. Objetivo general del Proyecto

Implementar un entorno de simulación donde se puedan aplicar diferentes políticas de tarifación para una topología o arquitectura de red con tecnología "Triple Play". A partir de él se estudiará el comportamiento de la red debido a diferentes perfiles de usuarios aplicando las diferentes políticas de tarifación que surjan del proyecto PDT y así poder evaluar el uso de los recursos del operador.

Se considerarán como criterios de éxito:

- Llegar al fin del proyecto en el tiempo estipulado cumpliendo con los requisitos, entre ellos, el más importante, que el software realizado permita simular el tráfico de una red Triple Play considerando:

- Tipos de usuarios
- Tipos de tráfico
- Políticas de tarifación e Ingeniería de Tráfico (Según los modelos que surjan del análisis a realizar en el proyecto PDT)

- Que el simulador cuente con una interfaz de usuario con un manual detallado incluyendo ejemplos de uso típico

- Realizar un resumen con lo aprendido sobre la tecnología Triple Play.

C.2. Actores, supuestos y restricciones

Actores:

- El grupo de integrantes del proyecto PDT
- Grupo de proyecto

Supuestos:

- Contar con la bibliografía y fuentes de información adecuadas para el estudio de la tecnología Triple play
- Que el grupo del PDT provea en un tiempo acordado de modelos para implementar, por ejemplo los perfiles de usuario; así como también que tenga prontas en las fechas previstas distintas políticas de tarifación para poder probar el simulador
- Que el software desarrollado se pueda ejecutar en tiempos aceptables

Restricciones:

- Recursos informáticos en los que debe correr el simulador
- Horas hombre disponible por los integrantes del equipo

C.3. Objetivos específicos

1. Estudiar la tecnología Triple Play y las posibles formas de implementarla en el simulador. Presentar un documento conteniendo un resumen de lo concluido
2. Estudiar los diferentes tipos de arquitectura que podemos simular y elegir alguno
3. Realizar un software que permita simular flujos en una red con una arquitectura Triple Play y permita aplicar políticas de tarifación y asignación de recursos
4. Verificar el funcionamiento del simulador, y realizar ajustes en caso de ser necesario
5. Implementar una Interfaz de Usuario
6. Aplicar las políticas de tarifación que surjan del proyecto PDT
7. Realizar documentación total del proyecto y presentación del simulador
8. Preparar de presentaciones de los entregables (realizar transparencias, resumen, documentos, etc)

C.4. WBS

1. *a*) Recolectar material de Triple Play
   b) Estudiar el material
   c) Realizar un documento con un resumen de lo aprendido de esa tecnología
2. *a*) Estudiar las diferentes topologías y protocolos que se utilizan en la tecnología Triple Play
   b) Elegir la arquitectura a implementar
3. *a*) Definir casos de uso que definan una arquitectura sencilla e implementarla. Se acuerda empezar por el estudio de un sistema sencillo formado por un único enlace y varias fuentes de flujo
   b) Definir flujos que simulen un comportamiento específico, tratando de acercarse al comportamiento de Triple Play
   c) Llegar a un acuerdo con el grupo del proyecto PDT para modelar los diferentes perfiles de usuario
   d) Implementar las clases que modelan a los distintos usuarios
   e) Definir casos de uso más complejos, teniendo un mayor conocimiento de los requerimientos por parte de los integrantes del proyecto PDT
   f) Implementar el simulador
4. *a*) Generar diferentes escenarios para realizar simulaciones y "debuging"
   b) Corregir errores
5. *a*) Definir la interfaz de usuario y como se relaciona
   b) Realizar la interfaz incluyendo un manual detallado de la misma
6. *a*) Aplicar políticas de tarifación suministradas por los miembros del PDT y analizar resultados
7. *a*) Realizar documentación final del proyecto incluyendo la ingeniería de software
   b) Presentación el proyecto
8. *a*) Presentación del entregable de setiembre de 2007
   b) Presentación del entregable de febrero de 2008

C.5. Objetivos planteados para los entregables intermedios

1er. Entregable (septiembre 2007)

- primer prototipo que simule una arquitectura y flujos sencillos. Se implementará un software que simule una arquitectura formada por un enlace y varias de fuentes de flujo. (la finalidad de este simulador es comenzar con un problema sencillo para luego adaptarlo a lo que se desea simular). Se entregarán ejemplos de prueba, y se evaluarán tiempos de ejecución y con esto la performance del simulador.

- documento describiendo las principales componentes de una arquitectura Triple Play y cómo se planea integrar estas al simulador.
- documento con plan de diseño del software final, se informará que se implementó hasta el momento.

2do. Entregable (febrero 2008)

- versión de pruebas del software final incluyendo implementación de diferentes perfiles de usuarios (se estarán haciendo pruebas y corrección de errores).
- primera versión de la documentación del software.
- informar sobre el diseño de la interfaz de usuario para la herramienta

C.6. Análisis de riesgos

En el siguiente cuadro se muestran los riesgos, y se definen para cada uno: la probabilidad de ocurrencia y el nivel de impacto en el cronograma.

	Riesgos	Probabilidad	Impacto
1	Nunca ningún integrante programó un software de dicha magnitud	muy probable	alto
2	Dependendemos de recibir información de terceros	moderada	alto
3	La performance no es la esperada (tiempo de ejecución, etc)	moderada	bajo
4	No se dispone del tiempo que se esperaba cuando se planificó	moderada	alto
5	Un integrante del equipo debe viajar o se enferma por mucho tiempo	Poco probable	medio

Figura C.1: Análisis de Riesgos

Observando, se tiene que hacer una planificación de respuestas a cada riesgo:

Nunca ningún integrante programó un software de dicha magnitud: para evitar que la falta de experiencia en programación influya en el resultado total del proyecto, se tomará como medida comenzar con dicha tarea al comienzo de cronograma. En particular se comenzará con el estudio e implementación de un software que permita simular un enlace y varias fuentes de flujo.

Dependemos de recibir información de terceros: se planificó suponiendo que el grupo PDT proporcionará distintas políticas de tarifación para probarlas en el simulador. También se definirán junto a los miembros del grupo PDT los diferentes perfiles de usuario. Si esto último no es así, se implementarán algunos perfiles a modo de ejemplo y se dejará documentado como se crearía un usuario que responda a un perfil (por ejemplo, que varíe la intensidad de flujos que envía)

No se dispone del tiempo que se esperaba cuando se planificó: si se constata dicho problema, se deberá re - planificar lo que reste del proyecto de modo que cada integrante del grupo pueda dedicarle lo que dispone de tiempo para el mismo. Esto se evaluará en los entregables intermedios.

C.7. Cronograma del proyecto - Listado de tareas

En el siguiente cuadro se indican las tareas a realizar para cumplir los objetivos específicos indicados, con sus respectivas fechas de inicio y fin (asignadas en la primer planificación), además de la carga horaria programada.

Tareas	Horas Asignadas	Fecha Inicio	Fecha Fin
Recolectar material de "Triple Play"	30	15/05/2007	28/07/2007
Estudiar el material	100	28/05/2007	11/08/2007
Realizar un documento con un resumen de lo aprendido de esa tecnología	30	13/08/2007	25/08/2007
Estudio de diferentes topologías y protocolos que se utilizan en la tecnología "Triple Play" y elegir la arquitectura a implementar.	50	11/06/2007	30/06/2007
Elegir la arquitectura a implementar	5	02/07/2007	06/07/2007
Definir casos de uso que definan una arquitectura sencilla e implementarla.	125	28/05/2007	30/06/2007
Definir flujos que simulen un comportamiento específico, tratando de acercarse al comportamiento de "Triple Play".	10	02/07/2007	07/07/2007
Llegar a un acuerdo con el grupo del proyecto PDT para modelar los diferentes perfiles de usuario.	30	24/09/2007	30/09/2007
Implementar las clases que modelan a los distintos usuarios.	200	01/10/2007	24/11/2007
Definir casos de uso más complejos, teniendo un mayor conocimiento de los requerimientos por parte de los integrantes del proyecto PDT.	80	09/07/2007	28/07/2007
Implementar el simulador.	800	09/07/2007	22/12/2007
Generar diferentes escenarios para realizar simulaciones y "debuging"	180	07/01/2008	31/03/2008
Corregir errores.	150	29/01/2008	31/03/2008
Definir la interfaz de usuario y como se relaciona.	50	28/01/2008	09/02/2008
Realizar la interfaz incluyendo un manual detallado de la misma.	150	13/02/2007	31/03/2008
Aplicar políticas de tarifación suministradas por los miembros del PDT y analizar resultados.	40	12/03/2007	20/03/2008
Realizar documentación final del proyecto incluyendo la ingeniería de software.	180	10/12/2007	17/02/2008
Preparar presentación del proyecto	60	01/04/2008	20/04/2008
Entregable de setiembre	20	03/09/2007	15/09/2007
Entregable de febrero	20	04/02/2008	17/02/2008

Figura C.2: Tareas Planificadas

Apéndice D

Evaluación de la planificación

En el capítulo anterior se muestra un cuadro con las fechas propuestas para el inicio y fin de cada hito a cumplir.

Dichas fechas (prácticamente en su mayoría) no fueron cumplidas. Una de las causas, fue la falta de experiencia a la hora de planificar por parte de los integrantes del grupo de proyecto. Pero, existieron otros motivos:

- En ocasiones no se le dedicó el tiempo necesario al proyecto, lo que llevó a que algunos integrantes del grupo se saturaran de tareas en otros momentos.
- La implementación del primer prototipo (red formada por un enlace) llevó más tiempo del planificado. Una causa de esto último, fue que hubo que dedicarle tiempo, por parte de los integrantes del proyecto, al repaso del lenguaje de programación.
- Por no tener ciertas precauciones a la hora de programar el prototipo mencionado, la adaptación del mismo a una red de varios enlaces costó más de lo imaginado.
- La definición de las políticas de tarifación fue más tarde de lo planificado por lo que se pudo programar la aplicación de sólo una política.

Apéndice E

Simulaciones

E.1. Verificación del primer prototipo - flujos streaming

Formula teórica de ErlangB

$$Pbloc = \frac{\frac{A^N}{N!}}{\sum \frac{A^i}{i!}} \tag{E.1}$$

Simulación Nº1

`DATOS:`
Capacidad Enlace = 400

Flujos:

- media duración = 100
- rate = 10
- media de tamaño debería estar cerca de 100/10 = 10.

Servidor: media de arribos = 2

Evaluando la fórmula de **Erlang B** en los siguientes valores,

$\lambda = 0{,}5$ (media con que tiran flujos los Servidores)
$\mu = 0{,}01$ (como la duración de cada flujo tiene media 100, consideramos que la tasa de atención es 0.01)
$A = \lambda/\mu = 50$(tráfico medio en Erlangs)
$N = 40$ (Número de "circuitos disponibles")

se obtiene que la probabilidad de bloqueo es de **Pblo= 0.2498**.

`Datos experimentales`

$FlujosRechazados/FlujosTotalesEnviados$, lo obtenemos del software para cada t.

Los resultados de la simulación son:

Cantidad de Flujos enviados por el servidor: 25231
Cantidad de Flujos rechazados por el enlace: 6380
Cantidad de Flujos cursados por el enlace: 18811
Cantidad de Flujos perdidos por el enlace: 40 *(perdidos porque la simulación terminó antes de que se finalicen de enviar)*
Cantidad de Flujos recibidos por el Cliente: 18811

Tiempo de ejecución del sistema : $4000mseg$

A continuación se muestra la probabilidad de bloqueo contra la hallada por ErlangB y el histograma correspondiente.

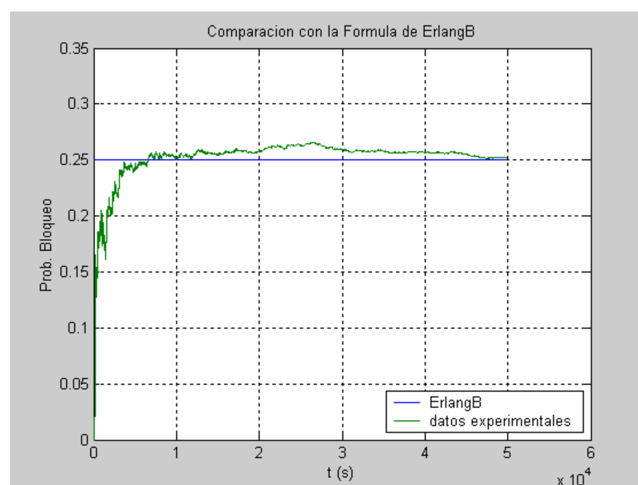

Figura E.1: Comparación de los datos experimentales con los teóricos dados por Erlang B

Histograma normalizado.

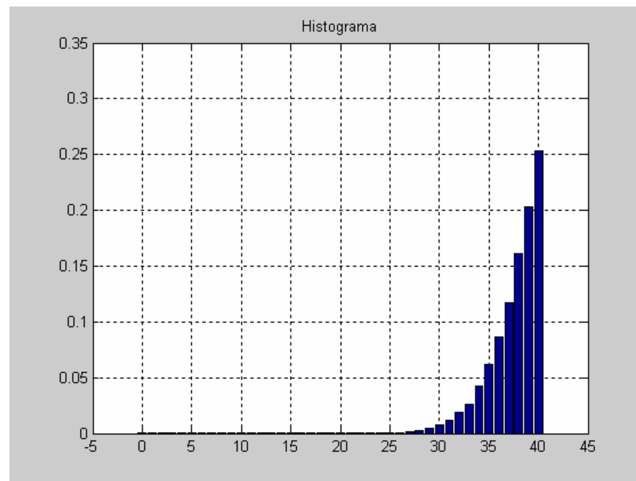

Figura E.2: Histograma

Simulación Nº2

`DATOS:`
Capacidad Enlace = 400

Flujos:

- media duración = 100
- rate = 10
- media de tamaño debería estar cerca de 100/10 = 10.

Servidor: media de arribos = 3,5

Evaluando la fórmula de Erlang en los siguientes valores,

$\lambda = 0{,}2857$ (media con que tiran flujos los Servidores)
$\mu = 0{,}01$ (como la duración de cada flujo tiene media 100, consideramos que la tasa de atención es 0.01)
$A = \lambda/\mu = 28{,}57$(tráfico medio en Erlangs)
$N = 40$ (Número de "circuitos disponibles")

se obtiene que la probabilidad de bloqueo es de **Pblo= 0.0084**.

`Datos experimentales`

$FlujosRechazados/FlujosTotalesEnviados$, lo obtenemos del software para cada t.

Los resultados de la simulación son:

Cantidad de Flujos enviados por el servidor: 14197

Cantidad de Flujos rechazados por el enlace: 131
Cantidad de Flujos cursados por el enlace: 14034
Cantidad de Flujos perdidos por el enlace: 32 *(perdidos porque la simulación terminó antes de que se finalicen de enviar)*
Cantidad de Flujos recibidos por el Cliente: 14034

Tiempo de ejecución del sistema : $2688mseg$

A continuación se muestra la probabilidad de bloqueo contra la hallada por ErlangB y el histograma correspondiente.

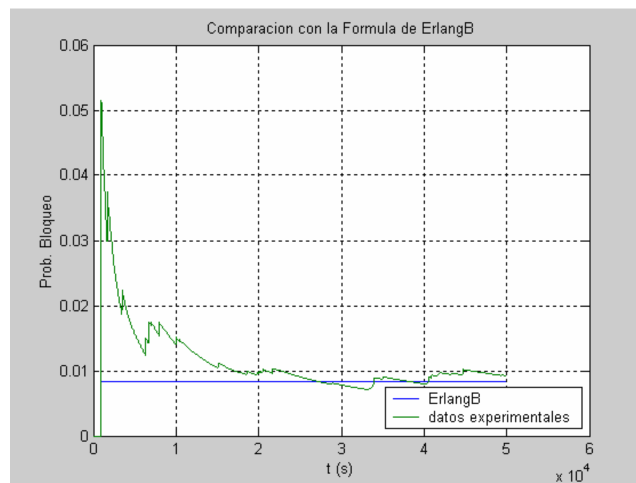

Figura E.3: Comparación de los datos experimentales con los teóricos dados por Erlang B

Histograma normalizado.

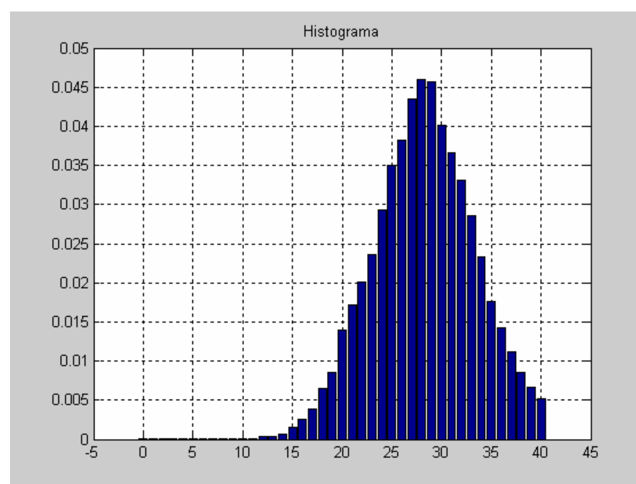

Figura E.4: Histograma

Se observa claramente que la probabilidad acumulada de bloqueo experimental se va acercando a la teórica hallada por ErlangB luego de ocurrido el transitorio. Esto pasa para ambos casos, por esto se puede concluir que el sistema se está comportando tal como se esperaba. Los histogramas también dan coherentes según los valores de tráfico en Erlangs de cada simulación, además se visualiza en las gráficas que el último valor del histograma es del orden de la **Pbloqueo** de ErlangB.

E.2. Verificación de la implementación del algoritmo de Max Min Fairness

Para el caso de un enlace, se muestran a continuación dos gráficas de dos simulaciones realizadas, con el objetivo de verificar el funcionamiento del algoritmo de asignación de recursos implementado. Las simulaciones fueron realizadas solamente con flujos elásticos transmitidos en la red.

Los resultados experimentales se comparan con los obtenidos en la evaluación de la fórmula teórica: $\rho^n(1-\rho)$.

Simulación Nº1

- Capacidad Enlace = 400
- Flujos: media tamaño = 100
- Servidor: media de arribos = 0,4

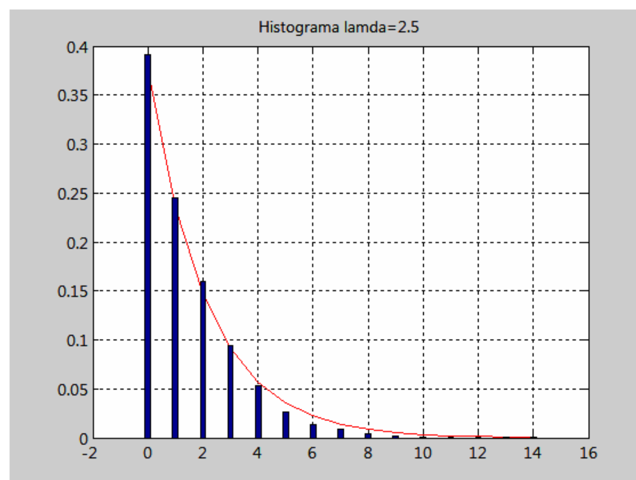

Figura E.5: Histograma

Simulación Nº2

- Capacidad Enlace = 400
- Flujos: media tamaño = 100
- Servidor: media de arribos = 0,5

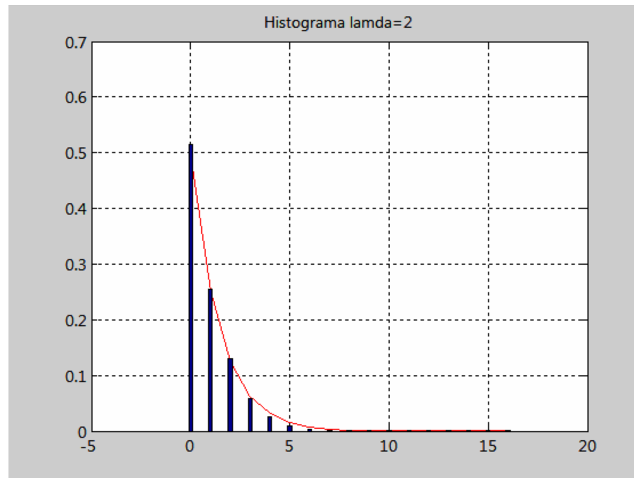

Figura E.6: Histograma

Como se observa en los gráficos anteriores, las respuestas teóricas (curva roja) y experimentales obtenidas para los flujos elásticos coinciden en gran parte. Dicha respuesta teórica es la correspondiente al modelo $\rho^n(1-\rho)$ para arribos con distribución Poisson.

Apéndice F

Manual de Usuario

La interfaz de usuario consiste en un archivo de texto con el nombre **"Datos de Entrada.txt"**. Al inicio de la simulación el programa realiza la lectura de dicho documento. Se debe crear una carpeta con el nombre **Trazas** en el disco C, y en esa carpeta se debe alojar el archivo antes de correr la simulación.

En **"Datos de Entrada.txt"** se define toda la topología de la red: enlaces, nodos, servidores[1] y clientes[2].

En el archivo de lectura, se debe asociar cada enlace a dos nodos [3], cada cliente y cada servidor a un nodo previamente definido. Además se deben definir las posibles rutas.

Otra cosa importante a realizar es, en el caso de existir clientes que soliciten flujos del tipo **FlujoStreaming**, se deberán crear los gestores correspondientes. Existirá un gestor por ruta y por rate solicitado. Esto no es necesario cuando existan clientes que soliciten únicamente flujos del tipo **FlujoElastico**.

No se deben crear elementos con el mismo nombre.

El usuario deberá crear el evento fin. Al llegar el turno de este evento se da por finalizada la simulación.

A continuación se muestra como crear cada elemento de la topología y como dar el comienzo y el fin de la simulación.

F.1. ¿Cómo definir un nodo?

Un nodo es un elemento de la topología que permite entre otras cosas la interconexión de los enlaces en la red.

[1]Los servidores pueden ser de dos tipos. Simulando los flujos que existen en Triple Play los servidores pueden pertenecer a la clase *ServidorElastico* o *ServidorStreaming*.

[2]Hay dos clases de clientes, *ClienteElastico* y *ClienteStreaming*

[3]Nodo inicio y nodo fin

Se define con la siguiente sintaxis:

```
Nodo(<nombre>)
```

El atributo **nombre** es un String y es representativo de cada nodo en la red.

Ejemplo:

Nodo(nodo1) : se declara un nodo al que se lo conocerá en la red con el nombre de nodo1.

F.2. ¿Cómo definir un enlace?

Para crear un enlace, se escribe lo siguiente:

```
Enlace(<capacidad>,<porcentageStreaming>,<nombre>)
```

Los campos **capacidad** y **porcentageStreaming** se deben completar con dos números reales. Estos representan la capacidad del enlace (b/s) y el % de la misma destinado para flujos del tipo streaming.

El campo **nombre** es un String y es representativo de cada enlace.

Luego de crear cada enlace es necesario asociarlo a dos nodos, para ello se escribe:

```
<nombre del enlace>.setNodoInicio(<nombre del nodo>)

<nombre del enlace>.setNodoFin(<nombre del nodo>)
```

Ejemplo:

```
Enlace(400,80,enlace1)

enlace1.setNodoInicio(nodo1)

enlace1.setNodoFin(nodo2)
```

En el ejemplo anterior se crea un enlace llamado **enlace1** con capacidad 400b/s y 80 % de su capacidad será destinada para flujos streaming. Se le asocian dos nodos, llamados **nodo1** y **nodo2**, como nodo inicio y nodo fin respectivamente (Se supone que los enlaces son unidireccionales).

Obs: nodo1 y nodo2 deben haber sido creados antes de ser asignados al enlace.

F.3. ¿Cómo definir un cliente que solicite flujos streaming?

Un cliente en la red representa a un conjunto de usuarios que se comporta de una forma en particular. Los clientes serán los encargados de solicitar flujos a los distintos servidores de la red.

La sintaxis para crear un cliente streaming es:

```
ClienteStreaming(<nombre>,<mediaDePeticiones>,<mediaOfertas>,
<rate>,<mediaDuracion>)
```

Al igual que los demás elementos de la topología de la red, el cliente tiene un nombre que lo identifica.

Cada cliente, en el caso de solicitar algún flujo, enviará peticiones a la red que serán analizadas por el correspondiente Gestor. El campo **mediaDePeticiones** es un real que indica la media de la distribución exponencial en que se realizan dichos pedidos.

Por otra parte, cada petición debe ir acompañada de una oferta[4]. El campo **mediaOfertas** es el utilizado para obtener una oferta.

El campo **rate**, indica la tasa que deberán tener los flujos solicitados por el cliente.
El atributo a llenar por el usuario mencionado con el nombre **mediaDuracion**, representa la media con que se sortearán aleatoriamente las duraciones de los flujos a pedir.

Además de definir el cliente, se lo debe ubicar geográficamente en la red. Esto se logra asociándolo a un nodo[5].

A continuación se muestra como crear un cliente, como asociarlo a un nodo y como indicar que comience a solicitar flujos. En el ejemplo se le indica al cliente que comience a solicitar flujos en el tiempo 0, o sea al inicio de la simulación.

```
ClienteStreaming(c1,2,.5,50,100)

c1.asociarNodo(nodo3)

c1.start(0,servidor)
```

obs: servidor es un servidor del tipo **ServidorStreaming** que debe ser creado antes de la última línea del ejemplo.

[4]dinero dispuesto a pagar por el cliente
[5]Previamente creado

F.4. ¿Cómo definir un cliente que solicite flujos elásticos?

La sintaxis para crear un cliente elástico es:

```
ClienteElastico(<nombre>,<mediaDePeticiones>,<mediaTamanio>)
```

Al igual que el caso anterior, tiene un nombre que lo identifica.

El campo **mediaDePeticiones** es un real que indica la media de la distribución exponencial en que se realizan dichos pedidos.

El atributo a llenar por el usuario mencionado con el nombre **mediaTamanio**, representa la media con que se sortearán aleatoriamente los tamanios de los flujos a solicitar.

Además de definir el cliente, se lo debe ubicar geográficamente en la red. Esto se logra asociándolo a un nodo[6].

A continuación se muestra como crear un cliente, como asociarlo a un nodo y como indicar que comience a solicitar flujos.

```
ClienteElastico(c1,2,100)

c1.asociarNodo(nodo1)

c1.start(0,servidor)
```

El campo **servidor** corresponde al nombre de una instancia de la clase **ServidorElastico** previamente definida.

F.5. ¿Cómo definir un ServidorElastico?

Una instancia de la clase **ServidorElastico** es un generador de flujos que simulan el tráfico de internet[7].

Para crear un elemento de esta clase se debe escribir en el archivo de texto lo siguiente:

```
ServidorElastico(<nombre>)
```

Además de definir el servidor, hay que asociarlo a un nodo de la red.

En el ejemplo siguiente, se muestra como definir un servidor de la clase ServidorElastico llamado **servidor1**. El servidor es asociado al **nodo1**, nodo previamente definido en el archivo de texto.

[6] Previamente creado
[7] Flujos que no necesitan un rate fijo para ser enviados

```
ServidorElastico(servidor1)

servidor1.asociarNodo(nodo1)
```

F.6. ¿Cómo definir un ServidorStreaming?

Un servidor del tipo ServidorStreaming es un generador de flujos que simulan el tráfico de voz y video[8].

En el siguiente ejemplo se muestra como crear el servidor y como asociarlo al nodo correspondiente.

```
ServidorStreaming(servidor1)

servidor1.asociarNodo(nodo1)
```

En las líneas anteriores se emula la creación de un servidor que cree flujos **Streaming**. Al servidor se le puso el nombre de **servidor1** y se lo ubicó geográficamente en la red asociado al **nodo1**.

F.7. ¿Cómo definir una ruta?

Para definir una ruta se le deben pasar tres parámetros, una lista de uno o más nombres de enlaces (separados por un espacio)[9]. Además se debe definir el nodo de inicio y un nombre que represente la ruta.

A continuación se muestran dos ejemplos, una ruta de un enlace y una ruta de dos enlaces (ambas coinciden en el nodo de inicio).

```
Ruta(enlace1,nodo1,ruta1)

Ruta(enlace1 enlace2,nodo1,ruta2)
```

Obs: las rutas deben ser creadas desde los servidores a los clientes, no viceversa.

F.8. ¿Cómo definir un Gestor?

El Gestor se debe crear en el caso de existir peticiones de flujos Streaming a la red, ya que es el encargado de decidir que ofertas aceptar o no.

[8] Flujos que necesitan un rate fijo para ser enviados

[9] Dichos nombres de enlaces deben hacer referencia a enlaces previamente definidos en el archivo de texto. Además debe existir continuidad en la secuencia de enlaces, en caso de no ser continua se tirará una excepción del tipo **RutaNoContinua**

Además de crearlo se debe asociar a un nodo de la red. Además se lo deberá asociar a una ruta y rate a gestionar.

Sintaxis:

```
Gestor(g1);

g1.asociarRuta(ruta1);

g1.asociarNodo(nodo3);

g1.asociarRate(100);
```

F.9. ¿Cómo definir un ControladorDeGestores?

El usuario debe tener presente, que se debe definir un elemento de la clase **ControladorDeGestores**, en caso de existir solicitudes de flujos streaming en la red.

La sintaxis para crear un elemento de esta clase es:

```
ControladorDeGestores(<tiempoEntreRemates>,<pasoDeIteracion>)
```

Un ejemplo de lo anterior es:
`ControladorDeGestores(90,1e-4)`

El campo **tiempoEntreRemates** define el tiempo en que se van a dar los Remates, al aplicar la política de tarifación implementada. En dicha implementación se utiliza el paso de iteración seteado por el usuario.

F.10. ¿Cómo definir el Evento Fin?

El evento fin debe ser definido por el usuario del simulador con la siguiente sintaxis:

```
Fin(<tiempoFin>)
```

En el atributo de tiempoFin es el que define el tiempo de fin de la simulación.

F.11. ¿Cómo habilitar las trazas a los distintos elementos de la red?

Para habilitar las trazas existen dos opciones.

Habilitar todas las trazas de una vez

Al final del archivo de texto, si se desea que todos los elementos que tienen la posibilidad de expulsar datos lo hagan, se debe escribir la siguiente línea:

```
TrazasON
```

Habilitar las trazas por separado

Los elementos que tienen la opción de habilitar la opción de trazar son las instancias de las clases:

- Gestor
- Enlace
- ClienteElastico
- Flujo

Para habilitar dicha opción se debe escribir en el archivo de texto la siguiente sintaxis:

```
<nombre>.trazar
```

El atributo **nombre** corresponde al nombre definido para la instancia (de alguna de las clases mencionadas) que se desea que escriba sus datos.