ESCUELA TÉCNICA Y SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

PROYECTO FIN DE CARRERA

Estudio del Algoritmo de Acondicionado PETER

Autor: Miguel Ángel Martínez Ramírez
Directora: María Dolores Cano Baños

Cartagena, Septiembre de 2017

Autor	Miguel Ángel Martínez Ramírez
E-mail del Autor	miguel.martinez@si.upct.es
Directora	María Dolores Cano Baños
E-mail de la Directora	mDolores.cano@upct.es
Título del PFC	Estudio del Algoritmo de Acondicionado PETER
Descritores	Network Simulator, Acondicionado de Tráfico, Fair Index, TSW, Token Bucket, CB, CBM, PETER

Resúmen

Con el aumento del volumen del tráfico ofrecido y cursado por Internet ha surgido la necesidad de proporcionar calidad de servicio (QoS). Esta necesidad ha afectado tanto a los proveedores, bien sea como hecho diferenciador ante la competencia, o para satisfacer las necesidades de servicio requeridas por determinados usuarios, dispuestos a pagar por ese servicio añadido a fin de poder usar determinadas aplicaciones que precisan una elevada calidad de servicio por parte de la red (por ejemplo servicios en tiempo real). Para este fin, el IETF ha propuesto diversas arquitecturas, entre las que se encuentra la denominada DiffServ. Esta arquitectura facilita la escalabilidad y el despliegue en las redes, ya que no precisa que todos los nodos tengan implementada esta arquitectura para que su uso mejore el rendimiento del sistema. Diffserv propone un tratamiento diferenciado en los nodos para un conjunto reducido de flujos o clases, de forma que todos los paquetes que pertenezcan a una misma clase recibirán un mismo tratamiento por parte de la red. Así, cuanto mayor sea la prioridad o el ancho de banda asignado a la clase mejor trato recibirá. La ventaja de DiffServ frente a otras arquitecturas consiste en la posibilidad de utilizar la actual infraestructura de red sin necesidad de introducir grandes cambios, lo que posibilita un despliegue gradual.

Un usuario que contrate un servicio asegurado espera que se le garantice el ancho de banda que ha contratado y además, tiene la convicción de que el tráfico que genere en exceso también se transmitirá si hay recursos disponibles. Este de reparto del ancho de banda sobrante se realizará del modo más justo posible. Algunos autores entienden por justicia, un reparto equitativo, mientras que otros autores, consideran un reparto justo, como un reparto proporcional a los recursos contratados por cada una de las fuentes de tráfico. El objetivo principal de este proyecto es el de implementar y analizar las prestaciones en NS-2 del algoritmo de acondicionado denominado PETER (Proportional Excess Traffic conditionER), que ofrece un reparto proporcional del ancho de banda no contratado dentro del servicio asegurado AF. El acondicionador de tráfico se sitúa junto a la fuente de tráfico (donde se establece el contrato) pero fuera del alcance del usuario final. Básicamente, este acondicionador marca paquetes IP con dos niveles de precedencia (in o out, lo que simplifica el esquema), utilizando el marcador CB, para posteriormente aplicar la nueva función policía PETER. La función PETER ajusta el throughput de las fuentes de tráfico, adaptándolas a las condiciones de la red mediante descarte de paquetes si fuera necesario.

Titulación	Ingeniero de Telecomunicación
Intensificación	Planificación y Gestión de Telecomunicaciones
Departamento	Tecnologías de la Información y las Comunicaciones
Fecha de Presentación	Septiembre de 2017

En memoria de mi padre

A mi madre y hermanos

Índice de Contenidos

1. Introducción **1**

1.1. Conceptos Básicos 1

1.1.1. Los Servicios Diferenciados 3

1.1.1.1. Dominio de Servicios Diferenciados 3

1.1.1.2. Región de Servicios Diferenciados 5

1.1.1.3. Clasificación y Acondicionado del Tráfico en un Dominio Diffserv 5

1.1.1.4. Comportamiento por Saltos 8

1.1.2. Mecanismos de Planificación Colas 11

1.1.3. Gestión Activa de Colas. Control de la Congestión 14

1.1.3.1. Random Early Detection (RED) 15

1.1.3.2. Generalizaciones del Algoritmo RED 19

1.2. Objetivos del Proyecto 22

1.3. Simulación de Acondicionadores de Tráfico 23

1.3.1. Escenarios de Simulación 24

1.3.2. Índice de Justicia 26

2. Análisis de Acondicionadores de Tráfico Clásicos en Diffserv **29**

2.1. Introducción 29

2.2. Acondicionador de Tráfico Basado en una Estimación de la Tasa. TSW (*Time Sliding Window*) 30

2.2.1. Configuración 32

2.2.2. Análisis de Prestaciones 35

2.3. Acondicionadores de Tráfico Basados en *Tokens* 40

2.3.1. Acondicionador de Tráfico *Token Bucket* 40

2.3.2. Acondicionador de Tráfico CB *(Counters Based)* 47

2.4. Conclusiones 53

3. Técnicas para un Reparto Equitativo del Ancho de Banda Excedente. **55**

3.1. Acondicionador de Tráfico Basado en las Condiciones de la Red. CBM (*Counter Based Modified*) 55

3.1.1. Funcionamiento del Acondicionador 56

3.1.2. Análisis de Prestaciones 61

3.2. Conclusiones 65

4. Técnicas para un Reparto Proporcional del Ancho de Banda Excedente. Algoritmo de Acondicionado PETER **69**
4.1. Descripción del Algoritmo 70
4.2. Análisis de prestaciones 72
4.2.1. Tasa de Velocidad de Paquetes *in* 73
4.2.2. Reparto del Ancho de Banda Excedente 77
4.3. PETER-DQ (Dual Queue) 81
4.3.1. Tasa de Velocidad de Paquetes *in* 82
4.3.2. Reparto del Ancho de Banda Excedente 85
4.4. Incremento del Número de Fuentes 89
4.5. Conclusiones 91

5. Conclusiones **93**

A. Herramienta de Simulación. Simulador de Redes *ns-2* **99**
A.1. Estructura Interna del Simulador 100
A.2. Módulo de Servicios Diferenciados 102
A.2.1. Detalles de Implementación 102
A.2.1.1. Cola RED en el Módulo Diffserv 103
A.2.1.2. Routers Interior y Frontera 104
A.2.1.3. Módulo de Políticas de Provisión de Recursos 105
A.2.2. Ejemplos de Configuración 106
A.3. Creación de nuevos módulos 108

B. Extracción de Resultados **111**
B.1. Estructura del Fichero de Trazas 111
B.2. Monitores 114
B.3. Herramientas Desarrolladas 116

Lista de Acrónimos **121**

Referencias Bibliográficas **125**

Índice de Figuras

1.1. Dominio de Servicios Diferenciados 4
1.2. Región de Servicios Diferenciados 5
1.3. Vista lógica del clasificador de paquetes y acondicionador de tráfico 7
1.4. Cabecera IPv4 9
1.5. Planificador PQ 13
1.6. Planificador FQ 13
1.7. Funcionamiento del algoritmo RED 16
1.8. Pseudo-código del algoritmo RED 17
1.9. Funcionamiento del algoritmo RIO 20
1.10. Pseudo-código del algoritmo RIO 21
1.11. Topología de la red simulada 25

2.1. Acondicionador de tráfico TSW 31
2.2. Pseudo-código estimador de tasa de TSW 31
2.3. Pseudo-código algoritmo marcado de TSW 32
2.4. Winlegth vs β para la fuente de tráfico con contrato de 1 Mbps 33
2.5. Configuración del parámetro β para la fuente de menor contrato 34
2.6. Guía de configuración del acondicionador de tráfico TSW 35
2.7. TSW: Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 20 ms para todas las conexiones) 36
2.8. TSW: Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 20 ms para todas las conexiones) 37
2.9. TSW: Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es de 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms) 38
2.10. TSW: Variación del índice de justicia para los escenarios de simulación A, B y C con una variación de la carga de la red entre el 10 % y 90 % 39
2.11. Pseudo-código del algoritmo Token Bucket 40
2.12. Acondicionador de tráfico Token Bucket 41
2.13. Throughput de paquetes *in* para los algoritmos Token Bucket y TSW en función del RTT. Fuente de tráfico TCP S_1 y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps) 42

2.14. Throughput de paquetes *in* para los algoritmos Token Bucket y TSW en función del RTT. Fuente de tráfico TCP S_{10} y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps) 43
2.15. Token Bucket: Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 20 ms para todas las conexiones) 44
2.16. Token Bucket: Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 20 ms para todas las conexiones) 45
2.17. Token Bucket: Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms) 46
2.18. Token Bucket: Variación del índice de justicia para los escenarios de simulación A, B y C con una variación de la carga de la red entre el 10 % y 90 % 47
2.19. Pseudo-código del algoritmo *Counters Based* 48
2.20. Throughput de paquetes *in* para el algoritmo CB en función del RTT. Fuente de tráfico TCP S_1 y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps) 48
2.21. Throughput de paquetes *in* para el algoritmo CB en función del RTT. Fuente de tráfico TCP S_{10} y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps) 49
2.22. CB: Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 20 ms para todas las conexiones) 50
2.23. CB: Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 20 ms para todas las conexiones) 51
2.24. CB: Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms) 52
2.25. CB: Variación del índice de justicia para los escenarios de simulación A, B y C con una variación de la carga de la red entre el 10 % y 90 % 53

3.1. Descarte anticipado de paquetes en el acondicionador CBM 56
3.2. Número de paquetes *out* generados entre paquetes *in* consecutivos por la fuente Src_1 57
3.3. Número de paquetes *out* generados entre paquetes *in* consecutivos por la fuente Src_{10} 57
3.4. Pseudo-código del algoritmo CBM 58
3.5. Número de paquetes *out* generados entre paquetes *in* consecutivos por la fuente Src_1 tras aplicar el descarte de parquetes 60
3.6. Número de paquetes *out* generados entre paquetes *in* consecutivos por la fuente Src_{10} tras aplicar el descarte de parquetes 60

3.7. CBM: Throughput de paquetes *in* en función del RTT para las fuentes de tráfico TCP S_1 y S_{10} y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps) 61
3.8. CBM: Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 20 ms para todas las conexiones) 62
3.9. CBM: Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 20 ms para todas las conexiones) 63
3.10. CBM: Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms) 64
3.11. CBM: Variación del índice de justicia para los escenarios de simulación A, B y C con una variación de la carga de la red entre el 10 % y 90 % 65
3.12. Comparativa del índice de justicia para el escenario de simulación C para los acondicionadores de tráfico TSW, CB y CBM y con una variación de la carga de la red entre el 10 % y 90 % 66
3.13. Acondicionador de tráfico CBM. Comparativa de los índices de justicia para una carga de la red del 60 % 67

4.1. Funcionamiento PETER 71
4.2. Pseudo-código del algoritmo PETER 72
4.3. PETER: Throughput de paquetes *in* en función del RTT para las fuentes de tráfico TCP S_1, S_3, S_5, S_7 y S_9 y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps) 74
4.4. PETER: Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 20 ms para todas las conexiones) 74
4.5. PETER: Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 20 ms para todas las conexiones) 75
4.6. PETER: Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms) 75
4.7. PETER: Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps.RTT es 80, 80, 60, 60, 40, 40, 20, 20, 10, 10 ms) 76
4.8. PETER: Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms) 76
4.9. Comparativa del índice de justicia para los escenarios de simulación A, B y C para los acondicionador de tráfico PETER con una variación de la carga de la red entre el 10 % y 90 % 80
4.10. Nodo DQ 81

4.11. PETER DQ: Throughput de paquetes *in* en función del RTT para las fuentes de tráfico TCP S_1, S_3, S_5, S_7 y S_9 y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps) 82
4.12. PETER DQ: Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 20 ms para todas las conexiones) 83
4.13. PETER DQ: Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 20 ms para todas las conexiones) 83
4.14. PETER DQ: Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms) 84
4.15. PETER DQ: Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps.RTT es 80, 80, 60, 60, 40, 40, 20, 20, 10, 10 ms) 84
4.16. PETER DQ: Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms) 85
4.17. Comparativa del índice de justicia para los escenarios de simulación A, B y C para los acondicionador de tráfico PETER con una variación de la carga de la red entre el 10 % y 90 % 88

A.1. Dualidad C++ y OTcl en *ns-2* 101
A.2. Estructura simplificada del simulador *ns-2* 102
A.3. Posición de la clase dsREDQueue en la jerarquía de clases 103
A.4. Parámetros PHB 104
A.5. Posición de las clases dsCore y dsEdge en la jerarquía de clases 105
A.6. Adición de la clase dsREDClass a la jerarquía de clases 109

B.1. Estructura del enlace con objetos de trazado 112
B.2. Estructura del fichero de Trazas 112
B.3. Ejemplo de fichero de trazas 113
B.4. Monitorización de una cola RED 114
B.5. Fichero monitor resultante 115
B.6. Evolución de la ocupación media e instántanea de una cola RED 115
B.7. Ejemplo de salida de awgStats.awk 117
B.8. Ejemplo de salida de instantThroughput.awk 117
B.9. Automatización de las simulaciones 118
B.10. Herramienta de visualización de resultados. Vista general 119
B.11. Herramienta de visualización de resultados. Análisis de datos 119

Índice de Tablas

1.1. Grupos de DSCP del campo DS de la cabecera IP 9
1.2. DSCP empleados en el servicio AF 10
1.3. Correspondencia del campo precedencia con los servicios Diffserv 11

2.1. TSW: Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 20 ms para todas las conexiones) 36
2.2. TSW: Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 20 ms para todas las conexiones) 37
2.3. TSW: Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms) 38
2.4. Token Bucket: Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 20 ms para todas las conexiones) 44
2.5. Token Bucket: Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 20 ms para todas las conexiones) 45
2.6. Token Bucket: Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms) 46
2.7. CB: Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 20 ms para todas las conexiones) 50
2.8. CB: Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 20 ms para todas las conexiones) 51
2.9. CB: Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms) 52

3.1. CBM: Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 20 ms para todas las conexiones) 62

3.2. CBM: Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 20 ms para todas las conexiones) 63
3.3. CBM: Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms) 64
3.4. Acondicionador de tráfico CBM. Comparativa de los índices de justicia para una carga de la red del 60 % 67

4.1. PETER: Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 20 ms para todas las conexiones) 77
4.2. PETER: Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 20 ms para todas las conexiones) 78
4.3. PETER: Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms) 78
4.4. PETER: Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms) 79
4.5. PETER: Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 80, 80, 60, 60, 40, 40, 20, 20, 10, 10 ms) 80
4.6. PETER DQ: Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 20 ms para todas las conexiones) 86
4.7. PETER DQ: Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 20 ms para todas las conexiones) 86
4.8. PETER DQ: Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms) 87
4.9. PETER DQ: Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms) 87
4.10. PETER DQ: Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 80, 80, 60, 60, 40, 40, 20, 20, 10, 10 ms) 88
4.11. PETER-RIO y PETER-DQ: Throughput total y de paquetes *in* para 20 fuentes de tráfico TCP y una carga de la red del 60 % (RTT es 20 ms para todas las conexiones) 89

4.12. PETER-RIO y PETER-DQ: Throughput total y de paquetes *in* para 20 fuentes de tráfico TCP y una carga de la red del 60 % (RTT es 10, 10, 10, 10, 20, 20, 20, 20, 40, 40, 40, 40, 60, 60, 60, 60, 80, 80, 80, 80 ms de S_1aS_{20}) . 90
4.13. PETER-RIO y PETER-DQ: Throughput total y de paquetes *in* para 20 fuentes de tráfico TCP y una carga de la red del 60 % (RTT es 10, 10, 10, 10, 20, 20, 20, 20, 40, 40, 40, 40, 60, 60, 60, 60, 80, 80, 80, 80 ms de S_1aS_{20}) . 91

5.1. Throughput total y de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 20 ms para todas las conexiones) 96
5.2. Throughput total y de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 20 ms para todas las conexiones) 97
5.3. Throughput total y de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms) 98

Capítulo 1

Introducción

1.1. Conceptos Básicos

Desde la aparición de Internet, su uso ha sido cada vez mayor. En los años noventa el público en general, más allá de las comunidades científicas, descubrió Internet como una potente herramienta de trabajo y de ocio. Esto ha provocado un crecimiento exponencial del tráfico de Internet, sin que nadie pueda prever cuándo terminará esa tendencia. Inicialmente, las aplicaciones que popularizaron Internet fueron fundamentalmente la navegación web, el correo electrónico, acceso remoto, servicios de descarga de ficheros (FTP). En este contexto, el modelo de convencional de servicios *best effort*, era lo suficientemente bueno para proporcionar al usuario un servicio final aceptable, dado que la mayoría de estas aplicaciones pueden considerarse de baja prioridad: aplicaciones de datos de pequeño ancho de banda con una elevada tolerancia al retardo y a las variaciones del mismo (*jitter*).

Sin embargo, esta situación está cambiando rápidamente, el enorme aumento del número de conexiones y la demanda cada vez mayor de nuevos servicios, ha propiciado que los servicios ofrecidos por Internet no sólo sean mejores y más especializados, lo que ha significado un proceso continuo de innovación, sino que además, ha significado un desafiante *test* de escalabilidad a los diseños originales.

Recientemente, la tecnología IP se está erigiendo como la base para conseguir la convergencia de servicios, siempre deseada por los proveedores de servicios (ISP). De esta forma el protocolo IP, aunque inicialmente no fue concebido para ello, está siendo utilizado para transportar información multimedia con requisitos de tiempo real, proporcionando servicios cada vez más extendidos como son los de telefonía, televisión, radio y videoconferencia. El buen funcionamiento de las aplicaciones se puede ver afectado al no poder garantizar el ancho de banda solicitado, su deterioro puede causar problemas significativos a los usuarios multimedia, lo que se materializa en una pobre calidad de servicio.

Los recursos disponibles (ancho de banda y capacidad de almacenamiento en routers) son siempre limitados, mientras que la demanda por lo general siempre ha sido creciente.

Para evitar la congestión de la red como consecuencia de este incremento de la demanda, existen dos tendencias bien distintas. Una implica sobredimensionar adecuadamente la red de transporte. Sin embargo, esta alternativa presenta algunas deficiencias, por una parte debe considerarse, que en la actualidad dado que las redes, cada vez más, se utilizan para ofertar servicios múltiples y heterogéneos, ya no se dimensiona el número de canales si no el ancho de banda, por lo que los modelos tradicionales de dimensionado ya no son válidos. En este sentido, un rápido incremento del volumen del tráfico en la red, podría desbordar en muy poco tiempo el sistema sobredimensionado. Por otra parte, se debe buscar un máximo aprovechamiento de los recursos y un óptimo grado de calidad en los servicios ofertados.

La otra tendencia, trata de gestionar de forma inteligente los recursos disponibles, compartiéndolos de manera desigual entre los diferentes flujos de tráfico. Sin embargo, en el esquema tradicional (*best effort*), la red es incapaz de distinguir ente las diferentes aplicaciones que transporta, por lo que necesitaremos que, de alguna manera, las funciones de Calidad de Servicio (*Quality of Service*, QoS) nos permitan catalogar las diferentes aplicaciones para reservarles unos determinados recursos de red. Esta necesidad ha afectado tanto a los proveedores, bien sea como hecho diferenciador ante la competencia, o para satisfacer las necesidades de servicio requeridas por determinados usuarios, dispuestos a pagar por ese servicio añadido a fin de poder usar determinadas aplicaciones que precisan una elevada calidad de servicio por parte de la red.

En realidad, existía cierta calidad de servicio en IPv4, pues en la cabecera IP se encontraba el campo denominado TOS (*Type Of Service*) de ocho bits, de los cuales los tres primeros representaban una prioridad (definida como precedencia), que permitía marcar los paquetes según su importancia, marcado que permitía establecer en principio políticas o prioridades en la transmisión de los paquetes por la red.

Aunque la prioridad representa una cierta calidad de servicio, por cuanto que permite clasificar los paquetes en categorías, no es capaz en general de ofrecer una garantía estricta, al estilo de la ofrecida por ATM, donde es posible reservar un caudal determinado para un circuito, aplicación o flujo determinado, asignándole un circuito virtual. Aunque un flujo concreto marque sus paquetes con máxima prioridad, no es posible garantizar un caudal mínimo o un retardo máximo, ya que podría sufrir congestión como consecuencia de una carga excesiva de paquetes de esa prioridad, producido por el tráfico agregado de otros flujos.

Ante las deficiencias presentadas, la IETF (*(Internet Engineering Task Force*) ha propuesto diversas tecnologías. La primera de estas propuestas de soporte de calidad de servicio fue la arquitectura de servicios integrados(*Integrated Services*, IntServ) [17]. Esta arquitectura define un modelo que se basa en garantizar QoS a las aplicaciones a través de la reserva de recursos extremo-a-extremo de la red para cada flujo antes de ser transmitido. Las aplicaciones solicitan el nivel de servicio necesario para poder trabajar apropiadamente y las reservas son mantenidas hasta que la aplicación termina o mientras cumpla con las demandas solicitadas. La reserva de ancho de banda puede ser hecha dinámicamente

utilizando el protocolo RSVP (*Resource Reservation Protocol*) [18]. Una vez realizada la reserva, la aplicación puede iniciar el envío de tráfico en la ruta sobre la cual se han solicitado los recursos. Cada fuente se ha comprometido con cierto patrón de generación el cual debe ser cumplido. Esta arquitectura si bien garantiza de manera estricta la QoS solicitada, no es lo suficientemente escalable debido a que el tratamiento por flujos puede dificultar su despliegue en escenarios donde conviven una gran multitud de flujos. Es muy costoso mantener en cada nodo una tabla de estados y reservas por cada uno de los flujo para el control de admisión, esto conduce a la existencia de un considerable tráfico de señalización (por el protocolo RSVP) a lo largo de la ruta, y ocupación de recursos en cada router.

Dado que la arquitectura IntServ seguía sin resolver el problema de la calidad de servicio en Internet, hacia 1997 la IETF presentó un modelo alternativo, denominado Arquitectura de Servicios Diferenciados (*Differentiated Services*, Diffserv) [21][28]. La idea básica de DiffServ, radica en que cada paquete incluye un código, denominado DSCP (*DiffServ Code Point*), que indica la clase a la que pertenece, y a partir del cual cada uno de los routers tratará de forma adecuada y diferenciada con respecto al resto de las clases, por lo que no han de mantener ninguna información sobre conexiones o flujos concretos. A ese comportamiento, en general, se le denomina PHB (per-hop Behavior) o comportamiento por salto. En el núcleo de la red, los paquetes son encaminados según el PHB asociado con su DSCP correspondiente. El número de clases posibles es limitado e independiente del número de hosts o de flujos transferidos a través de los routers, por lo que la arquitectura DiffServ es escalable. En realidad el modelo DiffServ "reinventó", hasta cierto punto, el olvidado campo TOS de IPv4, que incluía entre otros aspectos el subcampo precedencia que se comentó anteriormente. A nivel de red se desea establecer la diferenciación de servicios con el fin de poder acomodar los requerimientos de diferentes tipos de tráfico y necesidades de usuario. De esta manera los distribuidores de servicios, pueden aplicar diferentes baremos de precios según el servicio prestado al usuario final.

1.1.1. Los Servicios Diferenciados

El objetivo de esta sección es describir los componentes principales que conforman la arquitectura de servicios diferenciados [21], así como una serie de conceptos que serán de especial utilidad para el desarrollo del presente documento.

1.1.1.1. Dominio de Servicios Diferenciados

Un dominio de Servicios Diferenciados (*DS*) es un conjunto contiguo de nodos DS que operan bajo una política de provisión de servicio y un conjunto de grupos PHB común implementados en cada nodo. Un dominio tendrá sus límites bien diferenciados a través de un conjunto de nodos frontera que clasifican y posiblemente acondicionan el tráfico que ingresa en el dominio a fin de asegurar que los paquetes que transitan por él son apropiadamente marcados para seleccionar un PHB determinado de alguno de los grupos de PHB soportados en el dominio.

Los nodos dentro del dominio seleccionan la forma de encaminar los paquetes basándose en sus DSCP, haciendo corresponder ese valor a alguno de los PHB soportados usando bien las correspondencias DSCP-PHB recomendadas o bien algunas correspondencias definidas localmente. Tal y como se señala en [21], la inclusión de nodos que no implementen DS dentro de un dominio DS puede dar lugar a un comportamiento impredecible y puede impedir a la red satisfacer los acuerdos de nivel de servicio (*Service Level Agreements*, SLA), donde este acuerdo se define como un contrato de servicios entre un proveedor de servicios (*Internet Services Provider*, ISP) y su cliente, que define las responsabilidades del proveedor en términos del nivel de funcionamiento de la red (rendimiento, tasa de pérdidas, retardo, jitter) y la disponibilidad temporal, el método de medida, las consecuencias cuando los niveles de servicio no se consiguen o si los niveles de tráfico definidos son superados por el cliente, así como el precio de todos estos servicios.

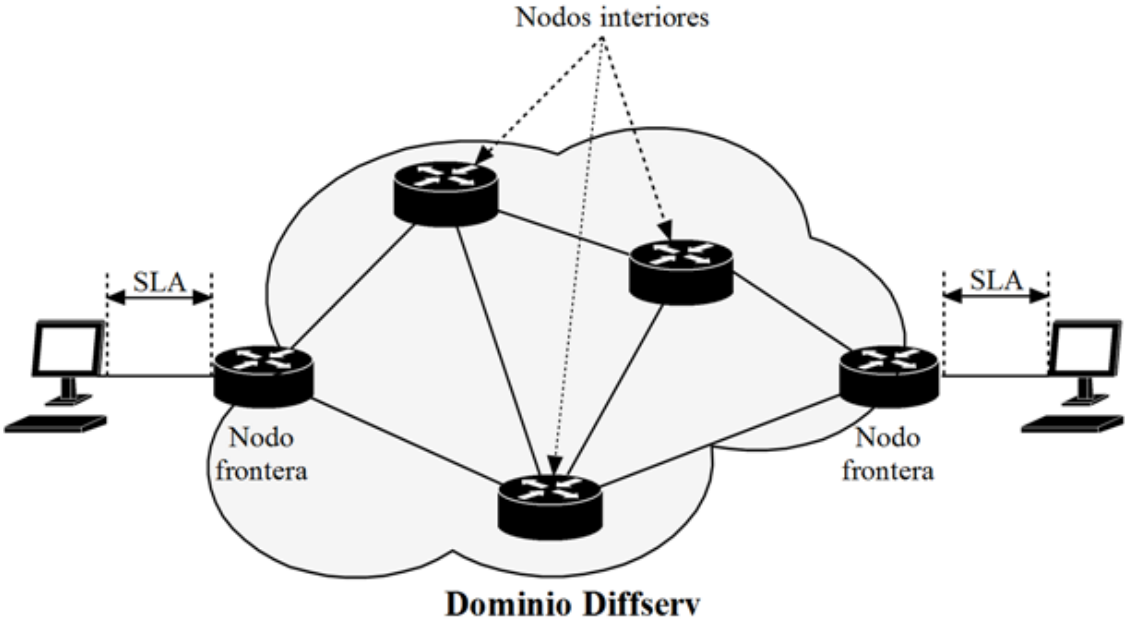

Figura 1.1: Dominio de Servicios Diferenciados

Un dominio normalmente consta de una o más redes bajo la misma administración, por ejemplo, la Intranet de organización o un ISP. El administrador de un dominio es responsable de asegurar que se provean los recursos adecuados y/o de reservarlos para dar soporte a los SLA ofrecidos por el dominio.

Como puede observarse en la Figura 1.1, un dominio de servicios diferenciados consta de nodos interiores y nodos frontera. Los nodos frontera interconectan el dominio con otros dominios que pueden o no soportar Diffserv, en cambio, los nodos interiores solo conectan con otros nodos interiores o frontera dentro del mismo dominio. Ambos tipos de nodos deben ser capaces de aplicar el PHB apropiado a los paquetes basándose en el código

DSCP, de lo contrario, el comportamiento resultante puede ser impredecible. Los nodos frontera, además, puede que sean requeridos para desempeñar funciones de acondicionamiento de tráfico, tal como se define en un Acuerdo de Acondicionamiento de Tráfico (*Traffic Conditioning Agreement*, TCA), entre su dominio y el dominio contiguo con el que conecta.

1.1.1.2. Región de Servicios Diferenciados

Una Región de Servicios Diferenciados es un conjunto de uno o más dominios Diffserv contiguos (ver Figura 1.2). Las Regiones de Servicios Diferenciados son capaces de soportar diferentes servicios para un flujo de datos que atraviese varios dominios dentro de una misma región. Nótese, por tanto, que un dominio particular en una región Diffserv puede soportar diferentes grupos de PHB internamente, sin embargo, para permitir servicios que se extiendan a través de diferentes dominios, los dominios contiguos deben cada uno establecer un SLA entre ellos que define (explicita ó implícitamente) cierta TCA, para especificar de que forma el tráfico de un dominio a otro es acondicionado en la frontera entre los dos dominios.

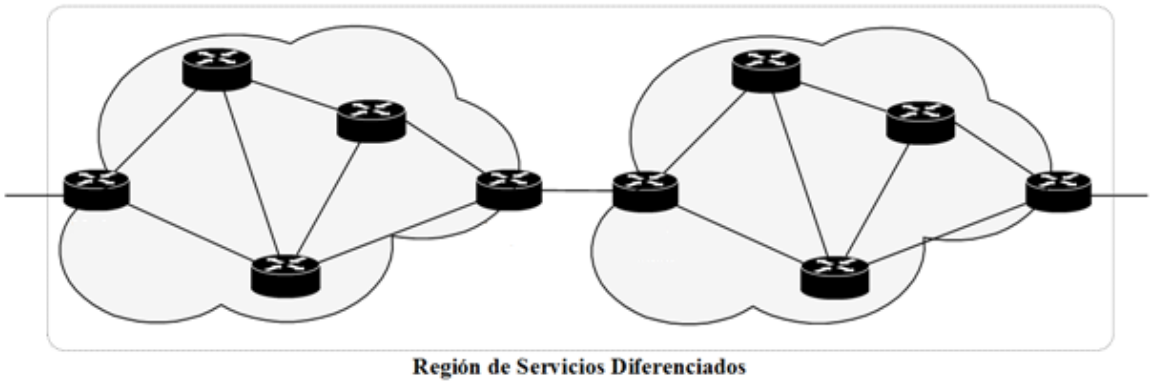

Figura 1.2: Región de Servicios Diferenciados

Es posible que algunos dominios dentro de una región puedan adoptar una política de provisión de servicios común y soportar un conjunto de grupos de PHB similares, eliminado de esta manera eliminar la necesidad de acondicionar el tráfico entre los dominios.

1.1.1.3. Clasificación y Acondicionado del Tráfico en un Dominio Diffserv

En la sección 1.1.1.1 se introdujo el concepto de TCA (*Traffic Conditioning Agreement*). A raíz de lo comentado, destacar que este acuerdo de acondicionado de tráfico puede especificar las reglas empleadas para llevar a cabo las tareas de clasificación y remarcado de paquetes, los perfiles del tráfico y las acciones a llevar a cabo en los flujos de tráfico cuando éstos se acomodan o no a los perfiles dados, por lo que se deduce, que el *TCA* entre

dominios deriva (explícita o implícitamente) del acuerdo de nivel de servicio (*SLA*).

La política de clasificación de paquetes identifica cuáles son los paquetes que pueden llegar a recibir un servicio diferenciado por parte de la red mediante el acondicionado y/o mapeado a uno o más comportamientos agregados (*Behaviour Agregates*, BA) dentro del dominio Diffserv. El objetivo de los clasificadores de paquetes es la de "dirigir" los paquetes que cumplen una norma especificada hacia un elemento acondicionador de tráfico para su posterior procesamiento (ver Figura 1.3). Los clasificadores deben configurarse mediante algún procedimiento que esté en concordancia con el *TCA* apropiado, además han de autenticar la información que se usa para clasificar los paquetes. En función de la información considerada para llevar a cabo la clasificación de paquetes, se definen dos tipos de clasificadores:

- **Clasificador de comportamiento agregado (*Behavior Agregate*, BA)**. Clasifica los paquetes considerando únicamente el DSCP del paquete.
- **Clasificador multicampo (*Multi-field Classifier*)**. Clasifica los paquetes basándose en el valor de uno o más campos de la cabecera del paquete, como puede ser la dirección origen, dirección destino, puertos origen y otra información como por ejemplo el interfaz de entrada.

Por otra parte, el acondicionado de tráfico desempeña las labores de medición, conformado, funciones policía y/o remarcado para asegurar que el tráfico que entra a un dominio Diffserv está conforme con las reglas especificadas en el TCA. La magnitud que puede alcanzar el acondicionado del tráfico depende de las especificaciones del servicio ofrecido, y puede ir desde un simple remarcado del DSCP a la elaboración de complejas funciones policía. Tal y como puede observarse en la Figura 1.3, un Acondicionador de Tráfico puede contener los siguientes elementos: un medidor, un marcador, un conformador y un descartador, siendo por tanto capaz de alterar las características temporales del flujo para adaptarlo a un perfil de tráfico. Las características más importantes de cada uno de estos elementos son las que se reseñan a continuación:

- **Medidor**: Mide las propiedades temporales de un flujo de paquetes comparándolo a un perfil de tráfico especificado en un determinado TCA. Un medidor pasa la información de estado a otras funciones de acondicionamiento para lanzar una acción particular para cada paquete. A raíz de lo anterior, se deduce, que es precisamente en este bloque funcional donde se dictamina si un determinado paquete esta o no dentro del del perfil (*in* o *out of profile*). Los medidores se pueden clasificar como orientados a paquete u orientados a flujo. Por una parte, un medidor orientado a paquete será aquel que mide el flujo de entrada cada vez que llega un paquete completo (Token Buket, CB, ..), en cambio, los medidores orientados a flujos miden el flujo en intervalos determinados de tiempo (medidor *EWMA*, medidor de tasa media, ..).
- **Marcador**: Establecen el valor del campo DS de la cabecera del paquete a un DSCP particular, añadiendo de esta forma, el paquete a un PHB especifico. El marcador se

puede configurar para marcar todos los paquetes dirigidos a él con un único DSCP, o bien para marcar un paquete con un valor de DSCP determinado dentro de un grupo de DSCPs, es decir, seleccionar un PHB determinado dentro de un grupo de PHBs, de acuerdo con el estado de un medidor. Cuando el marcador cambia el valor del DSCP de un paquete se dice que ha re-marcado un paquete.

- **Conformador**: Son los encargados de regular la salida de paquetes para que el flujo a la salida esté conforme con un determinado perfil de tráfico, para ello consta de un buffer finito para almacenar los paquetes que se desean retardar. Debido al carácter finito del buffer que almacena los paquetes, puede que, algunos paquetes se descarten si éste se encuentra lleno. Los conformadores más usados son los que implementan los algoritmos Leaky Bucket y Token Bucket.
- **Descartador**: Descartan algunos o todos los paquetes de un determinado flujo de tráfico con el fin de adaptarlo a un determinado perfil de tráfico. Nótese, que un descartador puede estar implementado cómo un caso especial de conformador (por ejemplo Leaky Bucket), en donde el tamaño del buffer que almacena los paquetes es cero.

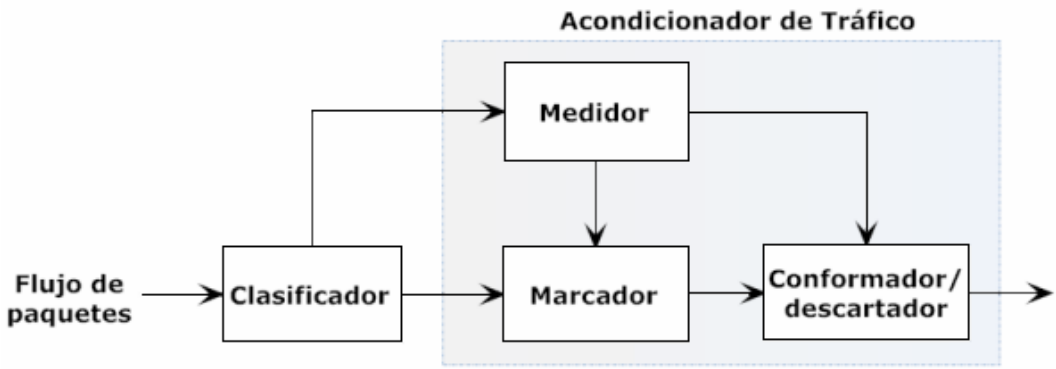

Figura 1.3: Vista lógica del clasificador de paquetes y acondicionador de tráfico

Aunque la arquitectura básica asume que la clasificación es una tarea compleja y las funciones de acondicionamiento de tráfico están ubicadas únicamente en los nodos frontera de un dominio, el desempeño de estas funciones en el interior del dominio no están prohibidas, sin embargo se está transfiriendo cierta complejidad en el diseño del sistema por lo que no resulta aconsejable esta alternativa.

Otra opción que se contempla es la de situarla en el nodo fuente, en este caso, tal y como se señala en [21], existen ciertas ventajas respecto al hecho de marcar paquetes cerca del origen de tráfico. Por una parte, la fuente tráfico puede considerar de manera más sencilla cuáles son las preferencias de las aplicaciones en el momento de decidir qué paquetes deben recibir mejor tratamiento de envío. Además, la clasificación de paquetes es más simple, antes que el tráfico haya sido agregado a otros paquetes provenientes de otras

fuentes, ya que el número de reglas de clasificación que deben ser aplicadas dentro de un nodo único es reducido. En este caso, el nodo frontera del dominio origen debe también monitorizar la concordancia con el TCA, así como vigilar el conformado o re-marcado de paquetes según sea necesario.

La última opción, sería la de localizarlas en dominios que no soportan Diffserv. El tráfico de origen o los nodos intermedios en un dominio que no soporta Diffserv, pueden emplear acondicionadores de tráfico para pre-marcar los paquetes antes de que estos alcancen el router frontera del siguiente dominio.

1.1.1.4. Comportamiento por Saltos

Formalmente, un PHB (*Per Hop Behavior*) se define como el tratamiento observable externamente que un nodo DS aplica de forma individual a un BA. PHB va a definir los requerimientos mínimos para encaminar un BA determinado, pero da libertad a los proveedores de servicio para usar los mecanismos que crean convenientes con el fin de conseguir estos requerimientos. Más concretamente, un PHB se refiere a las características de planificación y gestión de colas y de conformado que un nodo aplica a cualquier paquete de un determinado BA y que vienen dadas por el SLA que este BA tiene asociado.

Algunos PHB son definidos con tratamientos muy parecidos y se les denomina grupos PHB. Un ejemplo puede ser un conjunto de PHB que especifican el mismo tratamiento respecto a planificación y gestión de colas de la salida, pero indican diferentes probabilidades de descarte al gestor de colas según el tipo de tráfico. Dado que el DSCP de cada paquete indica el BA al que pertenece, también proporcionará el PHB que se le debe aplicar.

Como se ha comentado en varias ocasiones, la información sobre la calidad de servicio de cada paquete se especifica en un campo de un byte de la cabecera IP denominado DS, cuya estructura puede observase en la Figura 1.4. Este campo suplanta en Servicios Diferenciados a los campos *TOS* de la cabecera IP en IPv4 o Clase de Servicio en IPv6. En el campo DS, los seis bits más significativos forman el DSCP del paquete mientras que los dos bits restantes reciben el nombre de bits CU (*Current Unused*), cuyo uso se ha propuesto para la técnica de control de congestión ECN (*Explicit Congestion Notification*).

A raíz de todo lo anterior, se deduce que el conjunto de paquetes marcados con el mismo valor de DSCP que cruza la red DS en una determinada dirección recibe el nombre de BA (Behavior Aggregate). De esta forma, una gran cantidad de paquetes de múltiples aplicaciones y fuentes distintas pueden pertenecer al mismo BA. De esta forma, el DSCP puede emplearse para identificar hasta $2^6 = 64$ BA diferentes, aunque la IETF define un pequeño conjunto de DSCP estándar para asegurar la interoperatividad entre dominios diferentes. De cualquier forma, un dominio DS es libre de usar DSCP no estándar dentro del dominio siempre que los paquetes sean remarcados apropiadamente cuando abandonen dicho dominio. Los valores de DSCP se dividen en los tres grupos siguientes mostrados en

la Tabla 1.1.

DSCP	Posibles Valores	Uso
xxxxx0	32	Estándar
xxxx11	16	Local/Experimental
xxxx01	16	Reservado

Tabla 1.1: Grupos de DSCP del campo DS de la cabecera IP

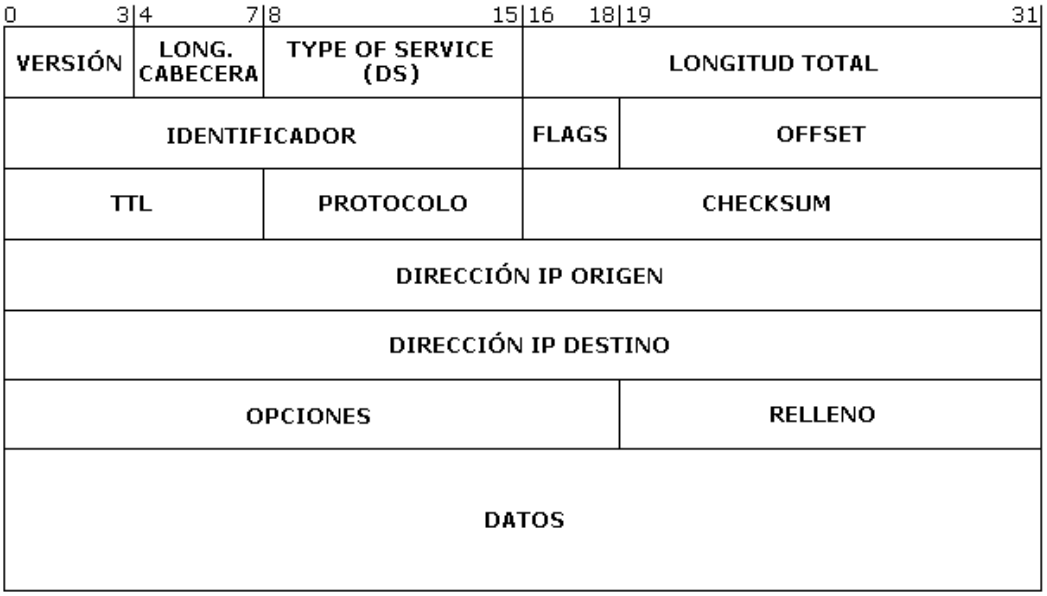

Figura 1.4: Cabecera IPv4

Así pues, de momento se contemplan 32 posibles categorías, correspondientes a los cinco primeros bits del campo DS. Bajo este contexto, la IETF ha definido los PHB EF (*Expedited Forwarding*) [23], AF (*Assured Forwarding*) [22] y el PHB por Defecto (*Best Effort*) [20]. Para cada uno de estos PHB se propone un servicio punto a punto que se conseguiría aplicando al flujo del cliente ese PHB en cada uno de los nodos por los que va pasando hasta llegar a su destino. Las características principales de cada uno de ellos son las siguientes:

- **Servicio EF**: Este servicio es el que proporciona una calidad mayor de todos. Se debe garantizar un caudal mínimo, una tasa máxima de pérdida de paquetes, un retardo medio máximo y un jitter máximo. El valor del DSCP relacionado con este servicio es "101110".

- **Servicio AF**: Este servicio asegura un trato preferente, pero no garantiza caudales, retardos, etc. Se definen cuatro clases posibles pudiéndose asignar a cada clase una cantidad de recursos en los routers (ancho de banda, espacio en buffers, etc..). La clase se indica en los tres primeros bits del DSCP. Para cada clase se definen tres categorías de descarte de paquetes (probabilidad alta, media y baja) que se especifican en los dos bits siguientes (cuarto y quinto). Existen por tanto 12 valores de DSCP diferentes asociados con este tipo de servicio, que son los mostrados en la Tabla 1.2.

Clases	Precedencia de descarte		
	Baja	**Media**	**Alta**
1	00101	00110	00111
2	01001	01010	01011
3	01101	01110	01111
4	10001	10010	10011

Tabla 1.2: DSCP empleados en el servicio AF

  En este tipo de servicio, el proveedor de servicios puede aplicar una determinada función policía al usuario, y si el éste excede lo pactado el proveedor puede descartar datagramas, o bien aumentar la precedencia de descarte. Algunos ISP ofrecen servicios denominados "olímpicos" con categorías denominadas oro, plata y bronce (o tiempo-real, negocios y normal). Generalmente estos servicios se basan en las diversas clases de este servicio.

- **Servicio**: Este servicio se caracteriza por tener a cero los tres primeros bits del DSCP. En este caso los dos bits restantes pueden utilizarse para marcar una prioridad, dentro del grupo *best effort*. En este servicio no se ofrece ningún tipo de garantías.

El campo DS es una incorporación reciente en la cabecera IP, tal y como se comentó, anteriormente existía en su lugar un campo denominado TOS cuya estructura era la siguiente: precedencia (3 bits), flags (4 bits) y reservado (1 bit). El subcampo "Precedencia" permitía especificar una prioridad entre 0 y 7 para el paquete (7 máxima prioridad). Este campo es en cierto modo el antecesor del campo DS. A continuación se encontraba un subcampo compuesto por cuatro bits que actuaban como indicadores o "flags" mediante los cuales el usuario podía indicar sus preferencias respecto a la ruta que seguiría el paquete. Los flags, denominados D, T, R y C, permitían indicar si se prefería una ruta con servicio de bajo retardo (D = Delay), elevado rendimiento (T = Throughput), elevada fiabilidad (R = Reliability) o bajo coste (C = Cost). El campo TOS ha sido muy impopular: el subcampo precedencia se ha implementado muy raramente en los routers. En cuanto a los flags D,T,R,C prácticamente no se han utilizado y su inclusión en la cabecera IP ha sido muy criticada. Estos problemas facilitaron evidentemente la transformación del campo TOS en el DS, aunque existen todavía routers en Internet que interpretan este campo con su antiguo

significado de campo TOS. Dado que DiffServ casi siempre utiliza solo los tres primeros bits del DSCP para marcar los paquetes, y que los servicios de más prioridad, como es el caso del EF, se asocian con los valores más altos de esos tres bits, en la práctica hay bastante compatibilidad entre el nuevo campo DSCP del byte DS y el antiguo campo precedencia del byte TOS, como puede observarse en la Tabla 1.3.

DSCP	Posibles Valores
0	Best Effort
1	AF clase 1
2	AF clase 2
3	AF clase 3
4	AF clase 3
5	EF
6	Reservado
7	Reservado

Tabla 1.3: Correspondencia del campo precedencia con los servicios Diffserv

Evidentemente esta compatibilidad no es accidental. Tradicionalmente el campo Precedencia no hacía uso de los dos niveles de prioridad más altos, que quedaban reservados para mensajes de gestión de la red. En DiffServ se han reservado también los dos valores más altos de los tres primeros bits con lo que se mantiene la compatibilidad con el campo precedencia.

1.1.2. Mecanismos de Planificación Colas

Los mecanismos de planificación de colas determinan las características temporales de los paquetes. Indican en qué momento los paquetes almacenados en las colas internas de un nodo deben ser enviados al siguiente nodo. En general, los algoritmos de planificación presentan un compromiso entre la complejidad de su implementación y el cumplimiento de las características temporales de QoS. En este sentido, para satisfacer las necesidades de QoS de las diferentes conexiones, los nodos necesitan aplicar mecanismos de prioridad y gestión. Por una parte, la prioridad hace referencia normalmente a la capacidad de proporcionar diferentes tratamientos al retardo, por ejemplo, los paquetes de prioridad superior son servidos siempre antes que los de menor prioridad, en el contexto de dar salida a los paquetes. Por otra parte, los nodos también necesitan emplear algún mecanismo de gestión para asegurarse que algunas conexiones obtengan los recursos contratados (en términos de retardo y ancho de banda efectivo). Este mecanismo además asegura que

cualquier capacidad excedente esté distribuida de la manera más justa posible.

Entre los principales algoritmos de planificación se pueden citar: FIFO (*First In First Out*), PQ (*Priority Queue*), FQ (*Fair Queueing*), WFQ (*Weighted Fair Queueing*), WRR (*Weighted Round Robin*), etc.. Las características más relevantes de algunos de estos esquemas son las siguientes:

- **FIFO**: En la planificación FIFO, todos los paquetes son tratados igualmente colocándolos en una cola simple y de esta forma va sirviéndolos en el mismo orden en el que fueron llegando. Debido a su sencillez, resulta obvio, que para conmutadores basado en software, FIFO proporciona en la cola una carga computacional extremadamente baja en comparación con otros mecanismos más elaborados, donde las operaciones de procesado, por lo general resultan algo más complejas. A raíz de lo anterior, cabe destacar, que este algoritmo es bastante predecible, y por ende, facilita la fase de diseño considerablemente, puesto que por ejemplo, el cálculo del retardo medio de la cola se obtiene a partir del tamaño de la misma.

  Las principales deficiencias que presenta este mecanismo, es que no establece ningún tipo de prioridad para los paquetes entrantes, pues tan sólo se considera el orden de llegada de los mismos. Precisamente, esta características es la que lo hace poco adecuado para flujos de tráfico con requisitos de tiempo real, pues este trato igualitario puede provocar un aumento en el retardo y el *jitter*. Por otra parte, un determinado flujo puede monopolizar el espacio en la cola, denegando de esta forma el servicio al resto de flujos.

- **PQ**: Este mecanismo de gestión de colas, es una alternativa sencilla que asegura que el tráfico importante se maneja más rápidamente que el resto de paquetes, conforme a una definición estricta de prioridades. Como muestra la Figura 1.5, esta técnica utiliza cuatro colas para cuatro tipos de prioridad: alta, media, normal y baja, donde la cola de prioridad normal se usa para los paquetes que no hayan sido clasificados. En este modelo, las colas de mayor prioridad son siempre atendidas primero, luego la siguiente de menor prioridad, y así sucesivamente. Si una cola de menor prioridad está siendo atendida, y un paquete ingresa a una cola de mayor prioridad, ésta es atendida inmediatamente.

  Los gestores de tráfico que envían flujos a las colas de alta prioridad deben controlar que el volumen de tráfico de alta prioridad no sea excesivo. Si el tráfico fuera excesivo, se sirve muy poco tráfico de baja prioridad, manteniéndose las colas de baja prioridad por lo general llenas o casi llenas. Esto hace que se incremente el número de paquetes descartados, el retardo y el número de retransmisiones, pudiendo incluso, denegar el servicio a los paquetes de baja prioridad.

  PQ podría usarse para mejorar la estabilidad de la red en periodos de congestión, asignando la mayor prioridad a la cola que contenga el tráfico de control de la red y de protocolos de encaminamiento. Además, PQ ofrece una clase de servicio de

bajas pérdidas, bajo *jitter* y alto throughput para la cola de alta prioridad. Estas características permiten dar el tratamiento apropiado al tráfico que generan las aplicaciones con requisitos de tiempo real.

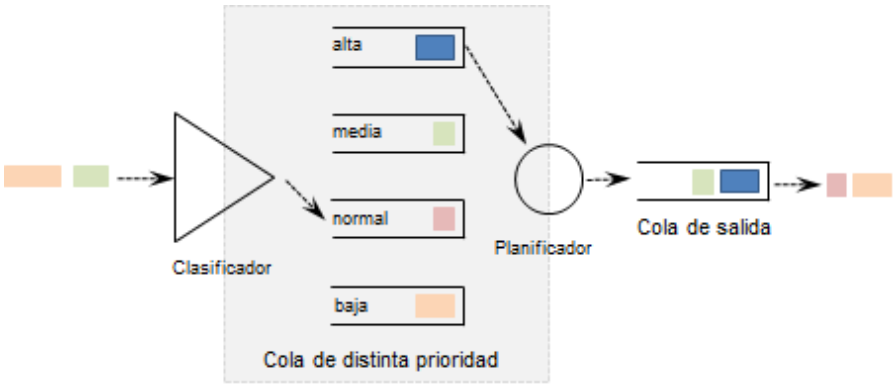

Figura 1.5: Planificador PQ

- **FQ**: Es un algoritmo de gestión de colas que opera generando una cola para cada uno de los flujos que atraviesa el router, es decir, siempre que un paquete llega al router, éste debe determinar a qué flujo pertenece, en base en sus direcciones IP y puertos TCP/UDP origen y destino, y colocarlo en la cola correspondiente. Cada cola es atendida en orden *Round Robin* (las colas se sirven en orden secuencial), asegurando, por tanto un distribución equitativa de los recursos para todos los flujos (ver Figura 1.6).

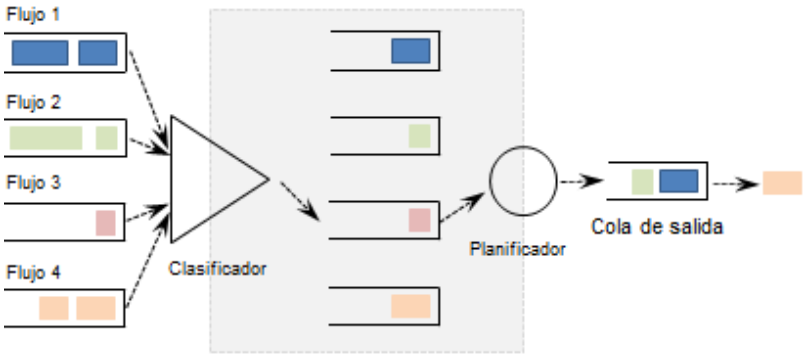

Figura 1.6: Planificador FQ

Aunque esta aproximación garantiza una distribución absolutamente justa, esta manera de proceder presenta una serie de inconvenientes que lo convierten en una solución poco viable para las redes actuales. Por una parte, el algoritmo requiere almacenar información sobre cada uno de los flujo para poder clasificarlos en la cola correspondiente, característica que lo hace poco escalable. Por otra parte, resulta obvio, que este mecanismo no proporciona un mecanismo que permita la gestión de

servicios en tiempo real, debido a que impide el reparto variable del ancho de banda para cada flujo.

Con el fin de mitigar la limitaciones expuestas, se propuso el algoritmo WFQ. La diferencia entre ambos radica, principalmente, que en este caso se asigna un peso a cada una de las colas para determinar la fracción del ancho de banda que le corresponde a cada uno de los flujos. Estableciendo de esta forma, cierto tratamiento diferenciado sobre aquellos flujos que son más sensibles al retardo.

1.1.3. Gestión Activa de Colas. Control de la Congestión

Si bien es cierto que una adecuada planificación determina el cumplimiento de los requisitos de QoS solicitados por las aplicaciones, esta será eficiente si se dispone de correctos mecanismos de gestión de la congestión. Estos mecanismos deben anticiparse a los problemas de congestión en cada una de las colas que contienen paquetes ya clasificados, y una mala elección afectará directamente a las pérdidas de paquetes.

Tradicionalmente, la técnica empleada en la gestión de las de colas de los routers era establecer una longitud máxima (en términos de paquetes) para cada cola. De esta forma, la cola acepta paquetes hasta alcanzar su longitud máxima, rechazando todos los paquetes entrantes hasta que uno de los paquetes que se encuentra en cola sea servido. Esta técnica es conocida como *Tail Drop*, ya que el último paquete en llegar es descartado si ésta se encuentra llena.

El método comentado ha sido empleado en internet durante muchos años, sin embargo presenta una serie de inconvenientes entre los que cabe destacar [19]:

1. En algunas ocasiones, el algoritmo *Tail Drop* permite a una sola conexión o a unas pocas monopolizar el espacio de la cola, evitando de esta forma que las demás conexiones puedan hacer uso de la cola. Este fenómeno es a menudo el resultado de la sincronización de las fuentes TCP o de otros efectos temporales.

2. El algoritmo *Tail Drop* tiende a mantener la cola al máximo nivel de ocupación durante largos periodos de tiempo, dado que el descarte de paquetes se inicia cuando la cola se encuentra llena. Uno de los problemas que presenta esta forma de proceder, es que la cola no mantiene el espacio suficiente para albergar rápidos crecimientos de tráfico o ráfagas. A pesar que los mecanismos implementados en TCP restringen la tasa de envío, no son capaces de recuperarse tan fácilmente de paquetes descartados cuando tiene lugar una ráfaga de datos, cosa que es menos costosa cuando se trata de recuperar un solo paquete descartado. El problema de congestión puede estar agravado aún más por la presencia en las redes de flujos que no sean lo suficientemente reactivos a la congestión. Es más, desde que se detecta la congestión hasta que se reacciona, puede transcurrir un intervalo no despreciable de tiempo que se traduce en pérdidas consecutivas de paquetes o ráfagas, tanto del flujo TCP involucrado como para los otros flujos no reactivos.

3. Otro efecto no deseado es el denominado sincronización global de TCP. Este fenómeno consiste en que cuando se presenta congestión, todos los flujos TCP experimentan descarte de paquetes en el router. Como consecuencia de ello disminuyen su ventana y atraviesan por el proceso de slow start, comportamiento consecuente con el control de flujo de TCP. Sin embargo, como todos los flujos experimentan pérdidas de manera más o menos simultánea, todos disminuirán su consumo de canal efectivo a la vez, conduciendo a situaciones de infrautilización de la capacidad del enlace.

Con el fin de mitigar estas limitaciones se desarrolla las disciplinas adecuadas de gestión de buffers para implementar conjuntamente con cada versión de TCP. Es lo que se conoce en la literatura como Active Queue Management (AQM). En AQM, cuando un router detecta una incipiente congestión, los paquetes son descartados (a pesar que la cola no se encuentre al cien por cien de su capacidad) con el fin de notificar a las fuentes que deben reducir la carga en la red y con ello poder controlar la velocidad de llegada de los paquetes. Si el paquete es descartado tempranamente, las colas se incrementarán de manera más gradual (conserva una ocupación media baja en los routers) evitando de esta forma la saturación de los routers, de tal forma que el AQM es capaz de soportar mayores ráfagas de tráfico.

Por otra parte, al operar con una baja ocupación en las colas, se reduce el retardo extremo a extremo. De esta forma se proporcionará un menor retardo en servicios interactivos, lo cual es especialmente relevante para el tráfico sensible al retardo como es el caso de VoIP. La idea esencial de la implementación de cualquier tipo de mecanismo de gestión de colas consiste en reemplazar la técnica tradicional de gestión de colas *Tail Drop* a favor de proporcionar un mejor rendimiento en términos de utilización de las líneas, probabilidad de pérdida de paquetes y del retardo extremo a extremo. En este sentido, se debe encontrar un equilibrio entre el número de paquetes descartados y el retardo de cola a la hora de elegir los parámetros de configuración. Colas de gran tamaño máximo implican menos paquetes descartados durante la congestión pero aumentan el retardo de cola. Y por otra parte, colas de pequeño tamaño reducido disminuyen el retardo durante la congestión pero aumentan el número de paquetes descartados.

1.1.3.1. Random Early Detection (RED)

El esquema de gestión de colas RED (Random Early Detection) fue introducido en 1993 por Floyd y Jacobson [29], y más tarde descrito en la RFC 2309 [19]. La idea básica de este algoritmo consiste en prevenir la congestión descartando paquetes de forma aleatoria cuando el tamaño medio de la cola alcanza cierto umbral.

El funcionamiento de RED queda especificado mediante tres parámetros: los valores umbrales de la cola (min_{th}, max_{th}) y la probabilidad máxima de descarte de paquetes (max_p). Las zonas de funcionamiento del algoritmo RED, en función de la ocupación media de la cola (avg) y los parámetros anteriores, son las que se reseñan a continuación (ver Figuras 1.8 y 1.7):

1. Zona de funcionamiento normal [0, min_{th}): no se produce el descarte de ningún paquete.

2. Zona de prevención de la congestión [min_{th}, max_{th}): El paquete entrante es eliminado con probabilidad p_a, que depende linealmente del valor medio de la cola y del número de paquetes que han sido recibidos desde la última eliminación, tal y como puede observarse en las expresiones (1.1) y (1.2).

$$p_b = max_p \cdot \frac{(avg - min_{th})}{(max_{th} - min_{th})} \tag{1.1}$$

$$p_a = \frac{p_b}{1 - count \cdot p_b} \tag{1.2}$$

3. Zona control de congestión [max_{th}, ∞): se considera una situación de congestión grave y se descartan todos los paquetes.

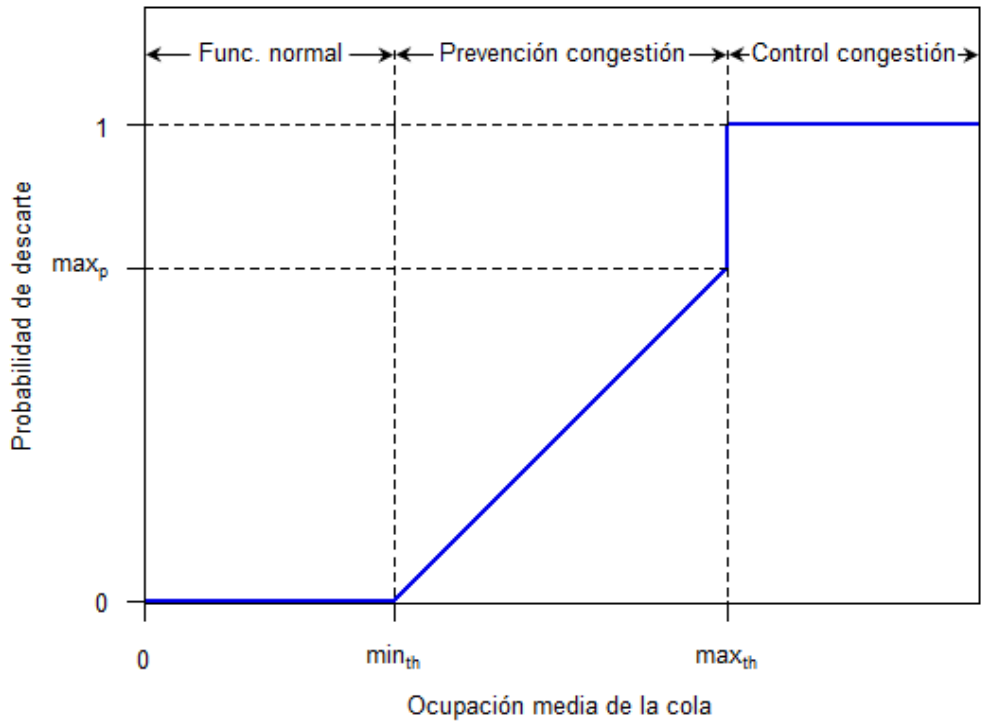

Figura 1.7: Funcionamiento del algoritmo RED

El valor medio de la ocupación de la cola avg se calcula cada vez que llega un paquete mediante el algoritmo *Exponentially Weighted Moving Average* (EWMA) definido por la expresión (1.3), donde q es la ocupación instantánea de la cola, avg es la ocupación media de la cola y w_q es el peso para calcular la ocupación media de la cola. El comportamiento del EWMA es el de un filtro paso bajo, para el que w_q define la frecuencia de corte.

$$avg = (1 - w_q) \cdot avg + w_q \cdot q \tag{1.3}$$

```
En cada llegada de un paquete:
   Actualizar avg (tamaño medio de la cola)

   if avg ≥ maxth
      descartar el paquete
   else if minth ≤ avg < maxth
      calcular probabilidad pa
      con probabilidad pa, descartar el paquete
   else
      almacenar en cola el paquete
```

Figura 1.8: Pseudo-código del algoritmo RED

En cuanto a la configuración de los parámetros de RED, a continuación se trata de justificar la elección de alguno de ellos:

- min_{th}: Su valor depende de la capacidad del enlace, del retardo de propagación, del tamaño máximo del buffer y del tipo de tráfico. Su valor se elige como un compromiso entre un retardo bajo y una utilización del enlace alta. Si se escoge un valor de min_{th} pequeño, el retardo medio de espera en la cola y la utilización del enlace serán también pequeños. Si se escoge un valor grande, se dará el comportamiento contrario.

- max_{th}: El valor óptimo para max_{th} depende del retardo máximo aceptable. No obstante, la diferencia entre min_{th} y max_{th} debería ser lo bastante grande como para evitar la sincronización entre todas las fuentes TCP. Normalmente, RED funciona adecuadamente cuando min_{th} es superior al incremento de la longitud media de la cola cada RTT, es decir, el número de paquetes que se puedan almacenar en un RTT debería ser inferior a esta diferencia para así evitar pérdidas. La recomendación que hace Floyd [14] es $max_{th} = 3 \cdot min_{th}$. Si una vez escogidos min_{th} y max_{th}, el retardo de espera es pequeño comparado con el retardo de transmisión y propagación del enlace, entonces es posible, incluso, aumentar estos valores para conseguir una mayor utilización del enlace y un aumento no significativo del retardo.

- max_p. En [29] no se discute en detalle sobre el valor adecuado para max_p. Su valor define la agresividad del algoritmo y depende mucho del número de fuentes multiplexadas sobre el enlace. Si max_p es grande, se descartarán paquetes de todas las fuentes y posiblemente la utilización del enlace sea baja. El valor óptimo de max_p es aquel que permita notificar la congestión al número adecuado de fuentes y permita conseguir maximizar la utilización del enlace. En [29] se propone un valor de 0.02 (2 %), aunque hoy en día se recomiendan valores mayores, por ejemplo 0.1 (10 %) es el empleado por defecto en NS-2 [2].

- Peso de la función promedio de la longitud de la cola (w_q). Si el valor de este parámetro es demasiado grande, entonces avg seguirá con precisión la ocupación instantánea de la cola y, posiblemente, se descarten paquetes durante picos instantáneos de tráfico innecesariamente. Como se indica en [29], dado min_{th} en paquetes, si se desea absorber una ráfaga de L paquetes sin descartes, se debería escoger un valor w_q que satisfaga la desigualdad de la expresión (1.4), para $avg < min_{th}$:

$$L + 1 + \frac{((1 - w_q)^{L+1} - 1)}{w_q} < min_{th} \tag{1.4}$$

Las ventajas que aporta la gestión activa de cola RED son entre otras:

- Identifica los estados tempranos de congestión y responde descartando aleatoriamente paquetes. Si la congestión continua creciendo, RED descarta paquetes de forma más agresiva para prevenir que la cola alcance el 100 por ciento de su capacidad, lo cual resultaría en una pérdida total de servicio. Esto permite a RED mantener un cierto nivel máximo de tamaño medio de cola incluso con los protocolos de transporte no cooperativos.

- Gracias a que RED no espera hasta que la cola esté completamente llena para comenzar a descartar paquetes, la cola puede aceptar ráfagas de tráfico y no descartar todos los paquetes de la ráfaga. Así, RED es apropiado para TCP porque no descarta grupos de paquetes de una única sesión TCP ayudando así a evitar la sincronización global de TCP.

- Permite mantener la cantidad de tráfico en la cola a nivel moderado. Ni demasiado bajo, lo que causaría que el ancho de banda estuviese infrautilizado, ni con valores cercanos a la capacidad máxima, donde el excesivo descarte de paquetes provocaría que una gran cantidad de sesiones TCP redujera sus tasas de transmisión, llevando a una pobre utilización del ancho de banda. Así, RED permite mantener el nivel de tráfico en cola de modo que se pueda obtener la mejor utilización del ancho de banda.

Las limitaciones de la gestión activa RED son:

- Puede ser muy difícil de configurar para obtener un funcionamiento predecible.

- RED no es apropiado para flujos no TCP tales como ICMP (Internet Control Message Protocol) ó UDP (User Datagram Protocol), ya que estos no detectan el descarte de paquetes y continuarían transmitiendo al mismo ritmo, perdiendo gran cantidad de paquetes debido a la congestión de la red.

- Existen algunos problemas en el uso e implementación de RED. Uno de ellos es que no tiene en cuenta las prioridades de los flujos a la hora de descartar, de modo que puede darse el caso de que se descarten paquetes de más prioridad mientras se estén sirviendo los de baja prioridad.

Este sistema podría utilizarse para gestionar una cola para tráfico *best effort*, ya que todos los flujos tendrán igual prioridad y el problema de RED de no distinguir prioridades no sería un inconveniente en este caso.

1.1.3.2. Generalizaciones del Algoritmo RED

RED tiende a tratar a todos los flujos que atraviesan el multiplexor de forma equitativa, ya que la probabilidad de descarte es mayor para los flujos con mayor tasa. Este tratamiento no es adecuado en un dominio DiffServ. Por ello se han propuesto diferentes variantes de este algoritmo como RIO (Random Early Drop with In/Out bit) [30].

RIO basa su funcionamiento en la existencia de dos colas virtuales, cada una con sus parámetros RED correspondientes (ver Figura 1.9). RIO emplea el mismo mecanismo de descarte de paquetes que RED, pero está configurado con dos conjuntos de parámetros, uno para calcular la probabilidad de descarte de paquetes IN (paquetes dentro del perfil), y otro la probabilidad de descarte de paquetes OUT (paquetes fuera del perfil), es decir, esta técnica emplea dos niveles de precedencia de descarte.

El funcionamiento de RIO es el siguiente (ver Figura 1.10): A la llegada de cada paquete, RIO examina si se trata de un paquete IN o un paquete OUT. Si se trata de un paquete paquete IN el router calculará el valor avg_{in} , la ocupación media de la cola virtual para los paquetes IN. En cambio, si se trata de un paquete etiquetado como OUT, el router calculará avg_{total}, la ocupación media de la cola física para todos los paquetes (tanto los marcados como IN como los marcados como OUT). Por tanto, considerando los tamaños medios de la cola calculado para cada tipo de perfil, la probabilidad de descarte en cada caso estará determinada por las expresiones (1.5) y (1.6):

$$p_{drop\_in} = max_{in} \cdot \frac{(avg_{in} - min_{in})}{(max_{in} - min_{in})} \tag{1.5}$$

$$p_{drop\_out} = max_{out} \cdot \frac{(avg_{total} - min_{out})}{(max_{out} - min_{out})} \tag{1.6}$$

Tal y como se señala en [31], en función de la ocupación media de la cola, RIO establece cinco zonas de funcionamiento . Las características más importantes de cada una de ellas son las que se reseñan a continuación:

1. Funcionamiento normal [0, min_{out}) : no se produce el descarte de ningún paquete.

2. Congestión incipiente [min_{out}, max_{out}): Con el fin de evitar que el tamaño medio de la cola del router crezca demasiado y por tanto se agrave el problema de congestión, se comienza a descartar paquetes OUT. Por otra parte, los paquetes IN entrantes observan un tamaño de la cola reducido por lo que no son descartados.

3. Congestión tolerada [max_{out}, min_{in}): En esta fase, el tamaño medio de la cola se calcula considerando únicamente paquetes IN, dado que todos los paquetes OUT entrantes son descartados automáticamente. Además debe considerarse que durante esta fase no se descarta ningún paquete IN. En la práctica puede darse el caso que el valor de los umbrales min_{th} y max_{th} coincidan, en cuyo caso esta fase no existiría.

4. Prevención de la congestión [min_{in}, max_{in}): En esta fase además de descartarse todos los paquetes OUT, se comienza a descartar paquetes IN con la finalidad de evitar que los buffers se desborden. Esta es una fase poco deseada para un ISP pues compromete el SLA que mantiene con el cliente.

5. Control de la congestión [max_{in}, ∞): El router descarta tanto paquetes IN como paquetes OUT con probabilidad 1. En esta fase, el objetivo principal del router ya no es la de establecer diferenciación alguna entre paquetes IN y OUT, si no que su función primordial es la de controlar la congestión, es decir, en esta fase el router RIO se degrada a un router *Tail-Drop*. Si el router se encuentra operando continuamente en esta fase, es un claro signo que el sistema no dispone de los recursos necesarios o bien la configuración de los acondicionadores de trafico RIO es errónea.

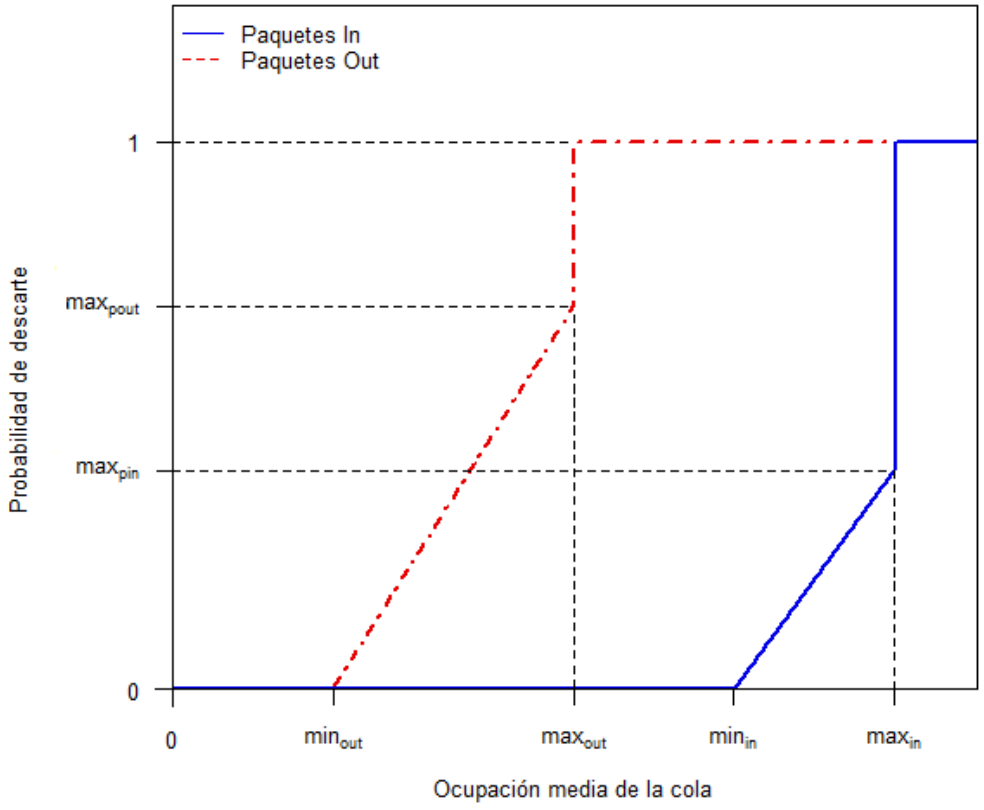

Figura 1.9: Funcionamiento del algoritmo RIO

A partir de la Figuras 1.9 y 1.10 se puede deducir que un router RIO es más agresivo en el descarte de paquetes OUT por tres razones: En primer lugar, el descarte de paquetes OUT se inicia mucho antes que el descarte de paquetes IN, esto se consigue tomando el valor de min_{out} mucho más pequeño que min_{in}. En segundo lugar, en la fase de congestión de la red, RIO descarta los paquetes OUT con una probabilidad más alta que los paquetes IN, dado que el valor de p_{out} es superior al valor de p_{in}. Y en tercer lugar, al establecer un valor

de max_{out} mucho más pequeño que max_{in}, el router RIO entra en la fase de control de congestión para paquetes OUT mucho antes que para los paquetes IN. En esencia, RIO descarta los paquetes OUT primero cuando se detecta una congestión incipiente, y descarta todos los paquetes OUT si la congestión de la red persiste, solamente como último recurso, cuando la red entra en fase de congestión, el router descarta paquetes IN con el propósito de controlar la congestión. En una red bien aprovisionada esto no debería suceder nunca, o al menos con una frecuencia muy baja.

```
En cada llegada de un paquete:

if Es un paquete marcado como IN
   Actualizar avgin(tamaño medio de la cola paquetes IN)

Actualizar avgtotal(tamaño medio de la cola paquetes IN y OUT)

if Es un paquete marcado como IN
   if minin ≤ avgin < maxin
      calcular probabilidad pin
      con probabilidad pin, descartar el paquete
   else if maxin ≤ avgin
      descartar el paquete
else
   almacenar en cola el paquete

if Es un paquete marcado como OUT
   if minout < avgtotal ≤ maxout
      calcular probabilidad pout
      con probabilidad pout, descartar el paquete
   else if maxout ≤ avgtotal
      descartar el paquete
else
   almacenar en cola el paquete
```

Figura 1.10: Pseudo-código del algoritmo RIO

Un ISP debe trata de configurar su sistema para evitar las fases 4 y 5 y operar principalmente en las fases 1, 2 y 3. Las fases 2 y 3 son para un router las zonas de funcionamiento ideal puesto que tanto la ocupación media de la cola y la instantánea son bajas, además de obtener una alta utilización del enlace. En este caso los paquetes OUT se descartan con el propósito de no comprometer los contratos de los ISPś con sus clientes. En cambio, si el router opera en la fase 1, la congestión es prácticamente nula, pero se obtiene una pobre utilización del enlace.

1.2. Objetivos del Proyecto

La arquitectura de Servicios Diferenciados aborda el problema de la provisión de calidad de servicio (QoS) en Internet mediante la definición de mecanismos de gestión de tráfico orientados a agregados de flujos, con la esperanza de que cada uno de los flujos que compone el agregado reciba una QoS determinada. De esta forma se mejora considerablemente la escalabilidad en la red troncal.

La arquitectura DiffServ define un conjunto limitado de servicios portadores y un conjunto limitado de Per-Hop Behaviors (PHBs) para soportar dichos servicios portadores. Dentro de los servicios ofrecidos, uno de los servicios más interesantes es el denominado servicio asegurado. Como ya se comentó en la sección anterior, el grupo de servicios asegurados (*assured*) pretende ofrecer un objetivo de QoS basado sólo en el caudal cursado por flujos TCP, sin ningún tipo de restricciones temporales, como puedan ser el retardo de los paquetes o la variación del *jitter*. Para su soporte se ha propuesto un grupo de PHBs denominados genéricamente Assured Forwarding (AF) que básicamente se diferencian en la probabilidad de descarte a la que pueden verse sometidos los paquetes. Además, a la entrada de la red es necesario disponer un sistema que realice las funciones de clasificación, verificación de la conformidad con el perfil de tráfico contrato y posiblemente sobreescritura del campo DSCP (re- marcado). En caso de congestión, los nodos internos de la red utilizan un mecanismo de descarte selectivo que comienza a descartar paquetes, primero, de los flujos AF con mayor prioridad de descarte. Los flujos TCP que sufren descarte reducen su ventana de transmisión para ajustarse al ancho de banda disponible.

Para implementar las funciones de descarte selectivo, la mayoría de los trabajos de investigación utilizan mecanismos de gestión activa de memoria. Su objetivo es evitar condiciones de congestión y en caso de producirse, minimizar su intensidad, propagación y duración. Típicamente están pensados para tráfico TCP, que es capaz de reaccionar ante situaciones de congestión. La mayoría están basados en el algoritmo Random Early Detecction (RED) [29]. El algoritmo RED propuesto inicialmente por Floyd y Jacobson para el soporte de RED ha sido mejorado considerablemente por diferentes contribuciones.

Dado que en un dominio DiffServ es necesario manejar diferentes prioridades de descarte para soportar diferentes servicios del grupo de servicios asegurados, se utiliza una versión generalizada de algoritmo RED , que en el caso más sencillo de sólo dos prioridades de descarte se denomina RIO (*Random Early Drop with In/Out bit*) [31].

Respecto a los algoritmos de marcado que deben ser implementados a la entrada de la red, diferentes trabajos de investigación han permitido progresar considerablemente en la comprensión de los fenómenos involucrados. Entre las contribuciones realizadas al marcado, acordes con la arquitectura DiffServ, se destacan los trabajos desarrollados en [31] y [8] y que servirán de referencia para el desarrollo del presente documento.

En el primer documento se proponen un conjunto de mecanismos que ofrecen, a la

arquitectura de Servicios Diferenciados, soporte para una asignación precisa y robusta del ancho de banda. Las propuestas más destacables del documento fueron los algoritmos RIO que es empleado por los routers interiores del dominio Diffserv y el algoritmo TSW (*Time Slidding Window*), un algoritmo probabilístico que se emplea en el acondicionado del tráfico y que junto con el Token Bucket, se tomará como punto de partida. En el segundo documento, se realizan varias propuestas de acondicionadores de tráfico. El primero de ellos es el acondicionador de tráfico denominado CB (*Counters Based*), acondicionador basado en contadores, cuya principal ventaja que aporta esta propuesta, frente al algoritmo presentado en [31], es su sencilla implementación y configuración. Por otra parte se analizarán los acondicionadores CBM (*Counters Based Modified*) y el acondicionador de tráfico PETER (Proportional Excess Traffic conditionER), donde se expondrá las principales ventajas que ofrecen éstos con respecto a los anteriores.

En este sentido, el objetivo principal de este trabajo, es el de analizar mediante simulación las prestaciones proporcionadas por lo acondicionares de tráfico presentados en los documentos mencionados. Por una parte, se comprobará que cada uno de los esquemas propuestos es capaz de asegurar los contratos del usuario final con mayor o menor exactitud. Por otra parte, se analizará la forma en la que se realiza el reparto del ancho de banda en exceso entre las fuentes que componen el agregado, que se debe realizar de la forma más justa posible, tema que suscita cierta controversia entre la comunidad investigadora, pues algunos consideran que un reparto justo es aquel, en el que el ancho de banda excedente se reparte de manera equitativa entre todas las fuentes de tráfico. Por otra parte, hay quienes consideran, que el ancho de banda excedente debería repartirse de manera proporcional a los contratos de cada fuente. En este estudio, se resaltará cuál de ellos es más adecuado para cada uno de los dos enfoques mencionados.

Finalmente, debe considerarse, que para realizar este análisis de prestaciones, la herramienta de simulación empleada debe facilitar esta tarea en la medida de lo posible. En este sentido, uno de los principales aspectos que ha de tener es que debe proporcionar la suficiente flexibilidad para agregar los elementos necesarios, en el caso que no se disponga de ellos.

1.3. Simulación de Acondicionadores de Tráfico

En la actualidad, existe una gran diversidad de simuladores: (OPNET, OMNET++, JSim, QualNet, GloMoSim, etc..), sin embargo, el Network Simulator versión 2 o simplemente *ns-2* se ha convertido en un estándar de facto debido a su amplia utilización en la comunidad investigadora. Probablemente una de las principales razones de su éxito es el hecho de que la distribución posee licencia GPL[34], condición que impulsa el desarrollo libre del mismo.

Para la realización de este proyecto se ha optado por emplear el simulador de redes *ns-2*, además de por las razones anteriormente expuestas, porque cumple las siguientes

características:

- Al tratarse de un simulador de código abierto, se facilita su depuración y la ampliación a campos no contemplados actualmente en la herramienta. Es decir, permite el estudio del funcionamiento del simulador y la posibilidad de modificarlo para perfeccionarlo o ampliarlo. Una de los principales inconvenientes que se ponen al desarrollo de estudios basados en la simulación es que los sistemas implementados no se ajustan completamente a la realidad. La forma directa de solucionar este problema es tener controlados todos los elementos que intervienen en la simulación, y el acceso al código es un elemento fundamental para este control.

- Además, el carácter abierto del código se traduce en la existencia de numerosos grupos de investigación en todo el mundo que realizan trabajos con la herramienta. Esto, unido al carácter abierto y cooperante, permite compensar ciertas limitaciones, como falta de manuales o servicio de mantenimiento, que los paquetes comerciales no sufren.

- Su programación está orientada a objetos, lo que flexibiliza y simplifica la tarea de programación, tanto de su código fuente como de su interfaz.

- Simula diversos entornos, gran variedad de protocolos(transporte, multicast, encaminamiento, ...), topologías simples o complejas, agentes, distintas fuentes de generación de tráfico (FTP, Telnet, Web, ...), simulación de aplicaciones, diversas políticas de gestión de colas (Tail Drop, RED, ...), modelado de errores, redes de área local, redes inalámbricas, etc

- Se han realizado numerosos estudios sobre calidad de servicio empleando *ns-2*, hecho que ha contribuido a que el simulador disponga de un modulo de servicios diferenciados relativamente amplio.

- Disponibilidad de herramientas de análisis y visualización desarrolladas para su libre uso.

La utilización del simulador *ns-2* trae consigo una serie de limitaciones. La principal es que en la actualidad, se trata de un producto que se encuentra descontinuado. Además, la mayoría de las aportaciones de los diferentes centros de investigación están validadas sólo para una versión, y es probable que no funcionen de forma correcta con versiones anteriores o posteriores. Por último, al no ser un producto comercial la ayuda no está convenientemente desarrollada. Esto hace que, aunque el desarrollo de pequeñas simulaciones con objetos ya existentes sea sencillo, la realización de simulaciones complejas como la implementación de nuevos protocolos sea complicada.

1.3.1. Escenarios de Simulación

La topología empleada para realizar las simulaciones se muestra en la Figura 1.11. Como puede observarse, esta topología consta de 10 fuentes TCP Reno que generan tráfico a la

máxima velocidad del enlace, establecida en este caso a 100 Mbps. Con respecto al tráfico generado se hacen las siguientes suposiciones:

- El tamaño de los paquetes es de 9188 bytes, que se corresponde con IP sobre el modo de transferencia asíncrono (Asynchronous Transfer Mode, ATM), y que podría representar DiffServ sobre el estándar de conmutación por etiquetas multi-protocolo (*Multi-Protocol Label Switching*, MPLS).
- Las fuentes de tráfico son FTP de tipo *long-lived*, es decir, disponen de datos ilimitados para enviar.
- En la mayoría de casos, se asumirá que la suma del ancho de banda contratado por todas las fuentes es inferior al ancho de banda disponible, con el fin de evaluar como realiza el reparto del ancho de banda excedente cada uno de los algoritmos de acondicionado de tráfico evaluados (en general, y salvo que se indique lo contrario, al 60 % de la carga de la red).
- La transferencia de datos se realiza de manera unidireccional. A pesar de tratarse de una suposición poco realista, se asumirá que no existe pérdida de reconocimientos (ACKs).

Por simplicidad, se ha considerado, que el acondicionador de tráfico se encuentra situado junto a la fuente generadora de tráfico, pero fuera del alcance del usuario.

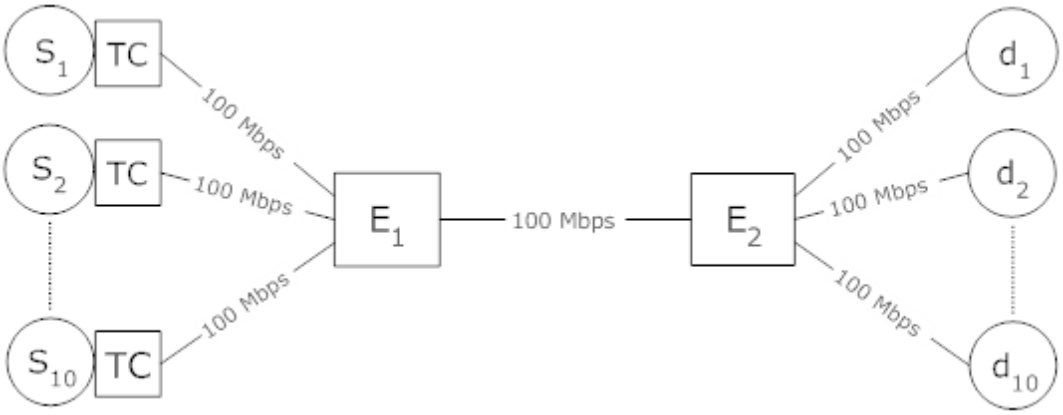

Figura 1.11: Topología de la red simulada

EL algoritmo RIO se encuentra implementado en los routers que multiplexan el tráfico proveniente de las ocho fuentes. Los parámetros de configuración se presentarán de la forma (min_{in}, max_{in}, $Pdrop_{in}$) para los paquetes dentro del perfil y (min_{out}, max_{out}, $Pdrop_{out}$) para los paquetes fuera del perfil, donde los umbrales se expresarán, por comodidad, en número de paquetes y la probabilidad de descarte como una fracción.

En general, y salvo que se indique lo contrario, se contemplarán tres escenarios de simulación diferentes, todos ellos con una carga de red del 60 % de la capacidad del enlace (suma de contratos igual a 60 Mbps). En este supuesto, se podrá analizar como se distribuye el 40 % del ancho de banda en exceso (no contratado), entre las diferentes fuentes que componen el agregado:

- **Escenario A:** Todas las conexiones tienen el mismo contrato y el mismo RTT. Los valores escogidos en este caso es de 6 Mbps para el caso de los contratos y 20 ms para el RTT para todas las fuentes de tráfico. Este es el escenario más simple y el más utilizado en estudios relacionados con acondicionado de tráfico y Servicios Diferenciados.
- **Escenario B:** Todas las conexiones tienen diferente contrato pero el mismo RTT. Los valores seleccionados en este caso de es 2, 2, 4, 4, 6, 6, 8, 8, 10 y 10 Mbps de S_1 a S_{10} respectivamente. El RTT, al igual que en el escenario anterior, queda fijado en 20 ms. El objetivo en este escenario es el de evaluar la influencia que tiene la diversidad de contratos en las prestaciones del acondicionador de tráfico.
- **Escenario C:** Todas las conexiones tienen diferente contrato y diferente RTT. En particular, las fuentes con menor contrato tendrán los RTT más bajos. Los valores seleccionados son de 2, 2, 4, 4, 6, 6, 8, 8, 10 y 10 Mbps con un RTT de 10, 10, 20, 20, 40, 40, 60, 60, 80 y 80 ms de S_1 a S_{10} respectivamente. En este caso se analizará los efectos de los contratos y RTT de manera simultánea.

1.3.2. Índice de Justicia

Tal y como se ha comentado, uno de los objetivos del presente documento es analizar la forma en la que se realiza el reparto del ancho de banda en exceso entre las fuentes que componen el agregado, que se debe realizar de la forma más justa posible. A fin de obtener una medida cuantitativa de la distribución del ancho de banda excedente, se utilizará el índice de justicia de Jain [15][16]. El valor del índice de justicia *f (fair Index)* se obtiene a partir de la expresión 1.7.

$$f = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \cdot \sum_{i=1}^{n} x_i^2} \leq 1 \tag{1.7}$$

siendo *n* el número de fuentes que componen el agregado. A medida que el valor del índice de justicia se acerque a la unidad, más justo será el reparto del ancho de banda no contratado. Tal y como se ha comentado, existen dos visiones distintas en lo que a la definición de justicia se refiere. En ambos casos, el valor de x_i en la expresión 1.7 será:

- En el primer enfoque, considerando justa una distribución **equitativa** entre las distinta fuentes, x_i representa el goodput (caudal sin considerar paquetes retransmitidos) excedente obtenido por la fuente *i*.

- Si se considera justa una distribución **proporcional** al contrato de cada una de las distinta fuentes, x_i representa en este caso el caudal en exceso que obtiene la fuente *i* dividido por el contrato de esta fuente (ver expresión 1.8).

$$x_i = \frac{caudal\ \ total\ \ fuente\ \ i - contrato\ \ fuente\ \ i}{contrato\ \ fuente\ \ i} \tag{1.8}$$

Nótese que por la propia expresión empleada para definir el índice de justicia, pequeñas variaciones en el mismo suponen una gran mejora en las prestaciones finales.

Capítulo 2

Análisis de Acondicionadores de Tráfico Clásicos en Diffserv

2.1. Introducción

Actualmente, y desde hace ya algún tiempo, nos encontramos en una situación de gran crecimiento del tráfico de datos como consecuencia, principalmente, de la generalización del uso de Internet. Este aumento de tráfico no viene sólo determinado por el cada vez mayor número de usuarios conectadas a la red, sino también por la diversidad de servicios que se proporcionan, muchos de ellos con altos requerimientos en cuanto a volumen de tráfico y cierta calidad de servicio. Internet no fue diseñada en sus orígenes para trabajar en este contexto, sino que más bien se limitaba principalmente al ámbito académico y de divulgación científica.

Una de las propuestas de mayor aceptación para garantizar ciertos requisitos de calidad de servicio es la arquitectura de Servicios Diferenciados. En gran medida, su nivel de aceptación se debe a la escalabilidad, fácil implementación y capacidad de ofrecer tratamiento diferenciado a una red en condiciones de congestión. Tal y como se comenta en el capítulo 1, la idea de esta arquitectura radica en clasificar los paquetes entrante en distintos niveles de servicio para enseguida aplicarle un comportamiento agregado para todos los flujos de una determinada clase de servicio. Estas funciones de clasificado y acondicionado del tráfico (medición, marcado, conformado, etc.), más complejas, sólo es necesario implementarlas en los nodos frontera, transladando de esta forma la complejidad a la periferia de un dominio determinado. Cada comportamiento es identificado por un único DSCP. A este comportamiento, en general, se le denomina PHB (*Per Hop Behavior*) o comportamiento por salto. En el núcleo de la red, los paquetes son encaminados según el PHB asociado son su DSCP.

El IETF ha estandarizado algunos PHB entre los que destacan el EF PHB y el AF PHB. El PHB Asegurado (AF PHB) [22] ofrece hasta cuatro diferentes niveles de servicio, lo que se denominan instancias AF. Dentro de cada instancia AF, un paquete IP puede pertenecer a uno de tres niveles de precedencia. Para implementar AF se puede emplear un medidor de

tráfico situado en los nodos frontera de la red y un sistema de gestión activa de colas. El medidor de tráfico compara el tráfico de cada flujo con el que está especificado en su perfil de cliente e indica al marcador qué paquetes están fuera de perfil (paquetes *out*, que no cumplen las especificaciones) y cuales dentro (paquetes *in*, que cumplen las especificaciones). El marcador asignará a los paquetes fuera de perfil un grupo de tratamiento con mayor prioridad de descarte asociada, que el que le asigna a los que cumplen con su perfil contratado. De esta forma, el servicio asegurado se diseñó para asegurar al usuario final un caudal mínimo (tasa contratada), incluso durante periodos de congestión de la red, además de permitir consumir más ancho de banda cuando la carga de la red es baja. Este ancho de banda no contratado (ancho de banda excedente) se deberá distribuir de modo justo entre las distintas fuentes que componen el agregado. Tal y como se comenta en el capítulo 1, en este trabajo se entiende por justicia el reparto proporcional del ancho de banda en exceso.

En este contexto, uno de los elementos esenciales en la arquitectura de Servicios Diferenciados es el acondicionador de tráfico. Como podrá comprobarse, resulta evidente la relación que existe el trabajo que desempeña y la calidad de servicio percibida por el usuario final.

El objetivo de este capítulo es analizar mediante simulación, las prestaciones de distintos tipos de acondicionadores de tráfico. En el análisis de prestaciones se deducirá la capacidad que tiene cada uno de ellos para garantizar los objetivos del Servicio Asegurado (*AF*). En gran medida, los resultados obtenidos con cada uno de ellos (número de paquetes marcados como *in* o como *out*) viene condicionado por una serie de factores como puedan ser la diversidad de contratos, el retardo, etc. en las distintas fuentes de tráfico. Otro factor clave, es el uso de mecanismo de control de la congestión empleado por los routers de la red, en este caso se utilizará RIO [31]. La tasa de envío de estará condicionada por este elemento, dado que el descarte de paquetes que lleva a cabo en fase de congestión produce una reducción de la tasa de transmisión de la fuentes de tráfico.

Para la realización de las simulaciones se empleará la topología mostrada en la Figura 1.11. La configuración escogida para los routers RIO empleados es (40, 70, 0.02) para los paquetes *in* y (10, 40, 0.2) para los paquetes *out*. El valor del peso de la función promedio de la longitud de la cola (w_q), para ambos casos, se establecerá en 0.002, tal y como se recomienda en [14].

2.2. Acondicionador de Tráfico Basado en una Estimación de la Tasa. TSW (*Time Sliding Window*)

En esta sección se analiza el algoritmo acondicionador de tráfico introducido por D. Clark y W. Fang en [30], denominado algoritmo de ventana deslizante TSW. Este algoritmo consta de dos componentes independientes: Un estimador de tasa que obtiene una estimación media

de la velocidad de envío de las fuentes de tráfico cada cierto periodo de tiempo y marcador que etiqueta los paquetes acorde a la tasa media obtenida por el estimador. En la Figura 2.1 se muestra la relación lógica entre el estimador de tasa y el marcador. Así mismo en las figuras 2.2 y 2.3 se observa el pseudo-código de estos dos componentes.

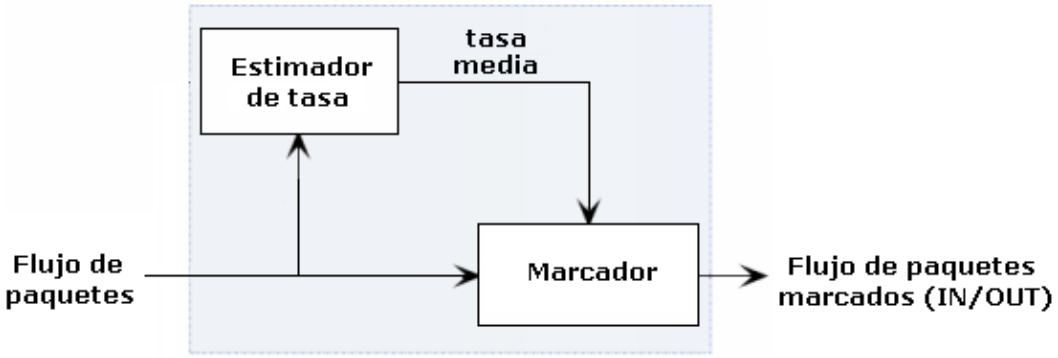

Figura 2.1: Acondicionador de tráfico TSW

Como puede observarse en la Figura 2.2 el funcionamiento del algoritmo es muy simple, dado que únicamente requiere ejecutar, para cada paquete, cuatro líneas de código. TSW emplea tres variables locales: *winlength* que se mide en unidades de tiempo, la estimación media de la tasa de envío que se calcula a la llegada de cada paquete y *t_front* que indica el instante de tiempo en el que llegó el último paquete que fue marcado. Nótese que el parámetro *winlength* es el único que ha de ser configurado, el resto se actualiza de forma automática para cada paquete entrante. En esencia, TSW realiza una estimación de la tasa de velocidad de la fuente en cada llegada de un paquete y mantiene un intervalo de historia pasada en el parámetro *winlength*. Tal y como se indica en [30], un acondicionador ideal debería mantener una conexión TCP oscilando desde 0,66 a 1,33 veces su velocidad contratada, de manera que en media la conexión alcance el contrato establecido.

```
Inicialmente:
winlength = una constante;
tasa_media = tasa contratada por la fuente;
t_front = 0;

En cada llegada de un paquete:
bytes_en_TSW = tasa_media * winlength;
nuevos_bytes = bytes_en_TSW + tamaño_paq;
tasa_media = nuevos_bytes/(ahora - t_front + winlength);
t_front = ahora
```

Figura 2.2: Pseudo-código estimador de tasa de TSW

En cuanto al marcado de paquetes, se pueden emplear dos procedimiento distintos. El

primero de ellos dispone de una mayor memoria y los paquetes se marcan como *out* con probabilidad *p* (ecuación 2.1) cuando la tasa media estimada excede el contrato. En el segundo caso, se dispone de una menor memoria y los paquetes se marcarán como *out* con probabilidad *p* cuando estimación media excede β veces el contrato, donde $\beta \geq 1$. Nótese, que para el caso particular en el que $\beta = 1$, el primer y segundo método coinciden. En las simulaciones se optará por el segundo método, pues es el recomendado por sus creadores.

$$p = \frac{tasa\ media - CIR}{tasa\ media} \tag{2.1}$$

En general, y como podrá comprobarse en las simulaciones realizadas, el acondicionador es TSW es especialmente sensible a este umbral β debido al mecanismo de control de la congestión de TCP, que es el que determina la tasa de envío instantánea de la conexión. Por tanto, las prestaciones de este acondicionador, dependerán en gran medida de la elección que se haga de este umbral y del parámetro *winlength*

```
En cada llegada de un paquete:
if tasa_media > beta * tasa contratada por la fuente
   marcar paquete out con probabilidad p
else
   marcar paquete in
```

Figura 2.3: Pseudo-código algoritmo marcado de TSW

2.2.1. Configuración

Una vez introducidos los elementos del acondicionador TSW y las particularidades de cada uno de ellos, se analizarán los valores que se deben escoger para realizar una configuración lo más óptima posible. En este caso, se entenderá por óptima, aquella configuración que proporciona un caudal de paquetes *in* tal que garantiza los contratos de las distintas fuentes de tráfico. A simple resulta obvio que debe analizarse el impacto que tiene, sobre los resultados obtenidos, la elección de un determinado umbral β o el valor que debe tomar el parámetro *winlength*.

El escenario de simulación en este caso, para simplificar, es similar al mostrado en la Figura 1.11, con la excepción que en este caso consideraremos únicamente 8 fuentes de tráfico FTP, con unos enlaces de 33 Mbps, con iguales RTT (50 ms) y unos contratos para las fuentes de 1, 1, 2, 2, 3, 3, 4 y 4 Mbps. A fin de obtener una primera aproximación al problema, se realizarán múltiples simulaciones en las que se variarán el valor del parámetro *winlength* y el umbral β. En la Figura 2.4 se muestran el caudal de paquetes *in* obtenido para la fuente de menor y mayor contrato respectivamente. A partir de estas gráficas se obtienen las siguientes conclusiones:

- A medida que aumenta el tamaño de la ventana *winlength*, mayor es el número de paquetes que se marcan como *in*.
- A medida que β aumenta, mayor es el número de paquetes marcados como *in*.

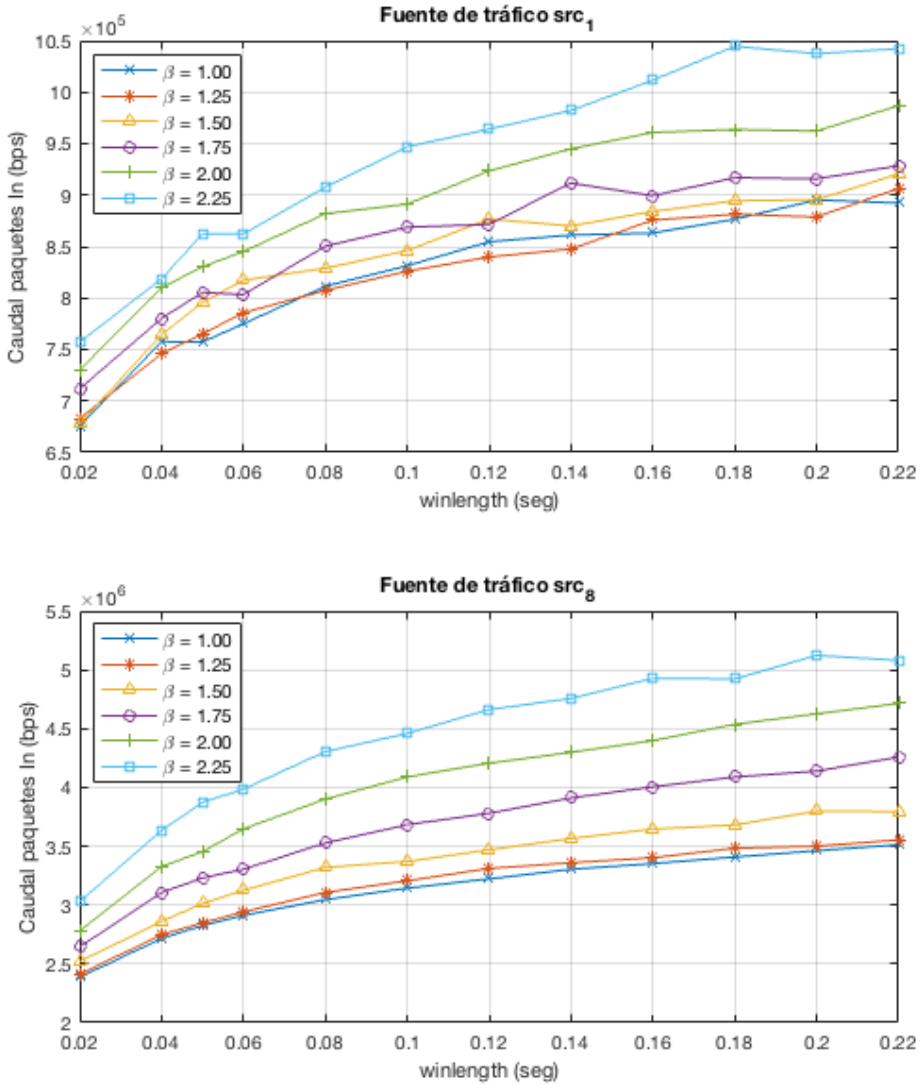

Figura 2.4: Winlegth vs β para las fuentes de tráfico de menor y mayor contrato

Resulta lógico, que a partir de esta información resulta bastante complicado, por no decir imposible, deducir cuales deben ser los valores que debe tomarse para que el caudal de paquetes *in* garantice de manera estricta los contratos de las distintas fuentes de tráfico. Sin embargo, si que es posible deducir que para $1 \leq \beta \leq 2$ y $160\ ms \leq winlength \leq 200\ ms$ se produce una relativa estabilidad en el caudal de paquetes *in*.

Llegados a este punto, se precisa realizar nuevas simulaciones a fin de obtener valores específicos para cada uno de estos parámetros configurables. Para ello se debe fijar un valor del tamaño de ventana *winlength* que cumpla los siguientes requisitos:

- Debe encontrarse en el rango de valores en el que caudal de paquetes *in* alcance cierta estabilidad.
- De todos los valores posibles, debe procurar escogerse aquel que proporcione un mayor caudal de paquetes *in*.

A raíz de lo anterior, se decide fijar el valor del tamaño de ventana *winlength* a 200 ms. A partir de este valor, las nuevas simulaciones deberán reflejar la evolución del caudal de paquetes *in* en función del valor de β (ver Figura 2.5).

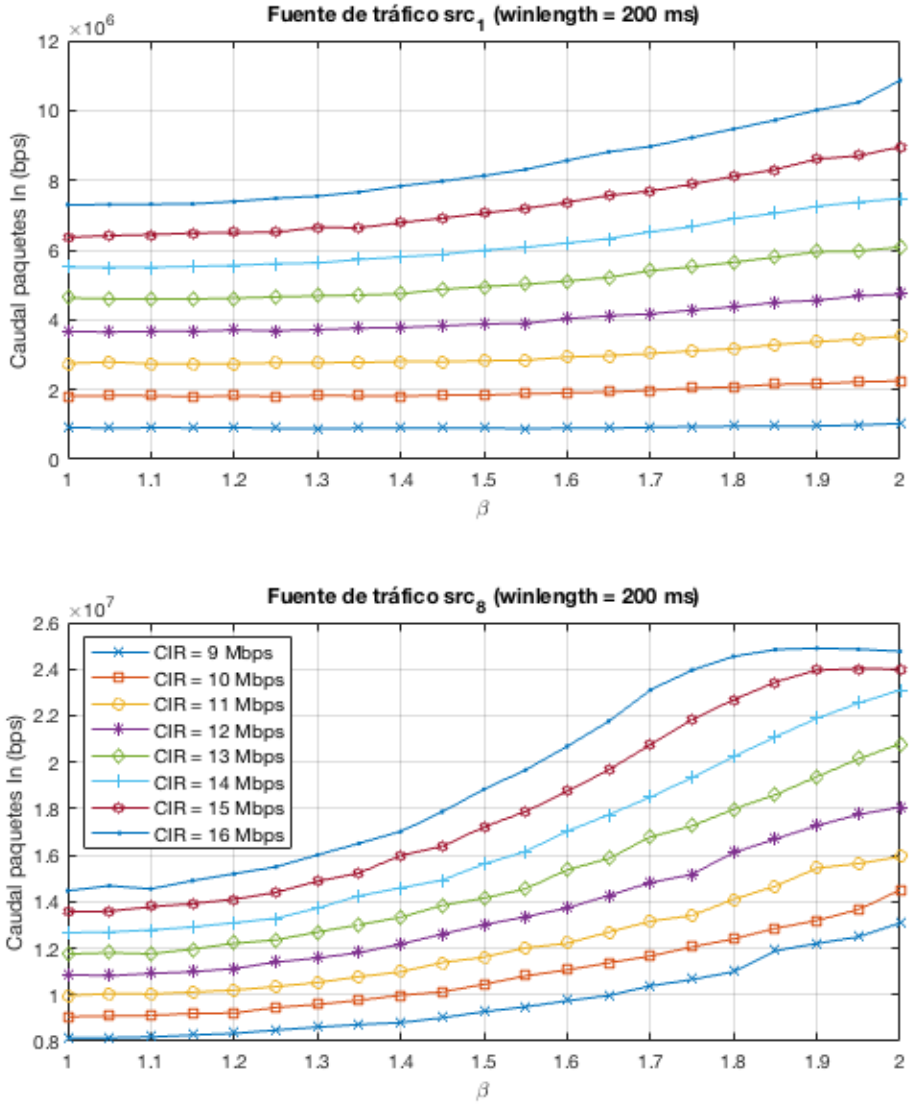

Figura 2.5: Configuración del parámetro β para las fuentes de menor y mayor contrato

A fin de facilitar la configuración del acondicionador TSW, y partiendo de la información proporcionada en la Figura 2.5, se construye el gráfico mostrado en la Figura 2.6, donde se refleja, para un tamaño de *winlength* de 200 ms, el valor que se debe escoger para el umbral β con el fin obtener unos resultados óptimos.

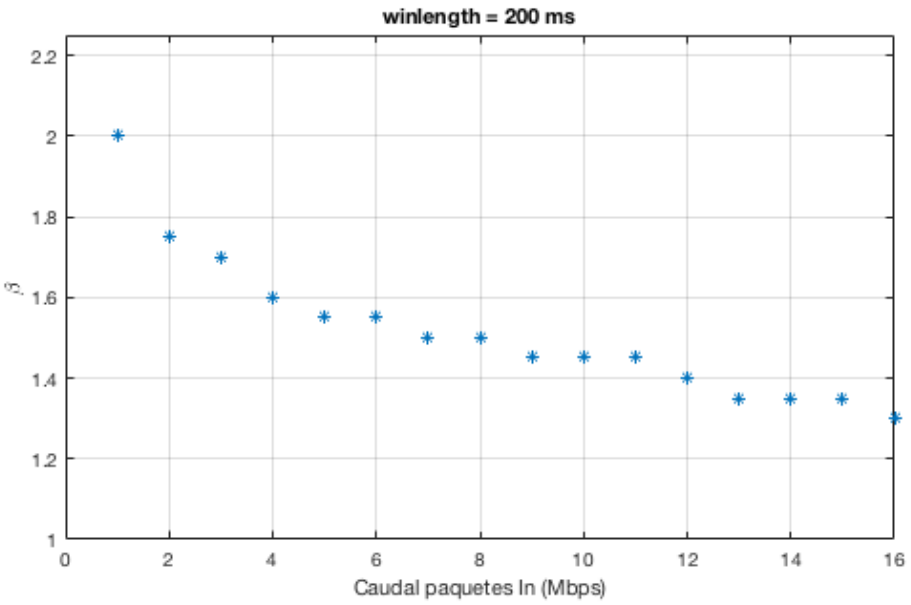

Figura 2.6: Guía de configuración del acondicionador de tráfico TSW

2.2.2. Análisis de Prestaciones

Una vez resuelto el problema de configuración del acondicionador TSW, se procederá a analizar las prestaciones que presenta en términos de garantías de contrato y reparto del ancho de banda excedente. En este caso los escenarios a considerar son los mostrados en la sección 1.3, es decir, ocho conexiones TCP. El tamaño de ventana *winlength* queda fijado a 200 ms y los valores de β se seleccionarán en función de los valores proporcionados por la gráfica de la Figura 2.6.

En las tablas 2.1, 2.2, 2.3 se muestran los resultados cuantitativos en lo eu respecta a caudal de paquetes *in*, caudal total y reparto de ancho de banda excedente para los tres escenarios descritos en la sección 1.3, todos ellos con una carga de red del 60 % de la capacidad del enlace. En esta situación se analizará cómo se distribuye el 40 % del ancho de banda en exceso entre las distintas fuentes, y la influencia de los contratos y el RTT de cada fuente en los resultados obtenidos.

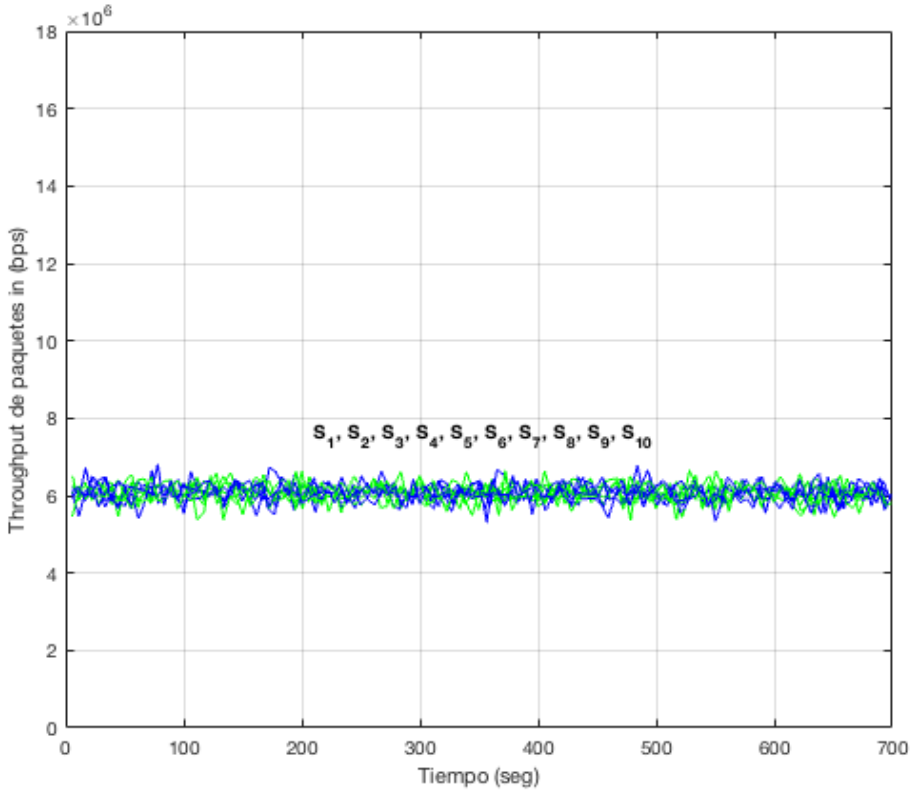

Figura 2.7: TSW - Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es de 6 Mbps para todas las conexiones. RTT de 20 ms para todas las conexiones)

Src	**RTT** ***(ms)***	β	**Contrato** ***(Mbps)***	**Caudal** ***in*** ***(Mbps)***	**Caudal Total** ***(Mbps)***	**Ancho Banda Exceso**	
						Mbps	%
1	20	1,55	6,00	6,07	9,89	3,81	9,53 %
2	20	1,55	6,00	6,08	9,92	3,83	9,58 %
3	20	1,55	6,00	6,11	9,82	3,72	9,29 %
4	20	1,55	6,00	6,07	9,85	3,78	9,45 %
5	20	1,55	6,00	6,11	9,90	3,79	9,46 %
6	20	1,55	6,00	6,09	9,90	3,81	9,53 %
7	20	1,55	6,00	6,09	9,87	3,78	9,46 %
8	20	1,55	6,00	6,09	9,86	3,77	9,43 %
9	20	1,55	6,00	6,08	9,79	3,70	9,26 %
10	20	1,55	6,00	6,07	10,01	3,94	9,85 %
		Total	60,00	60,87	98,80	37,94	94,84 %

Tabla 2.1: TSW - Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 20 ms para todas las conexiones)

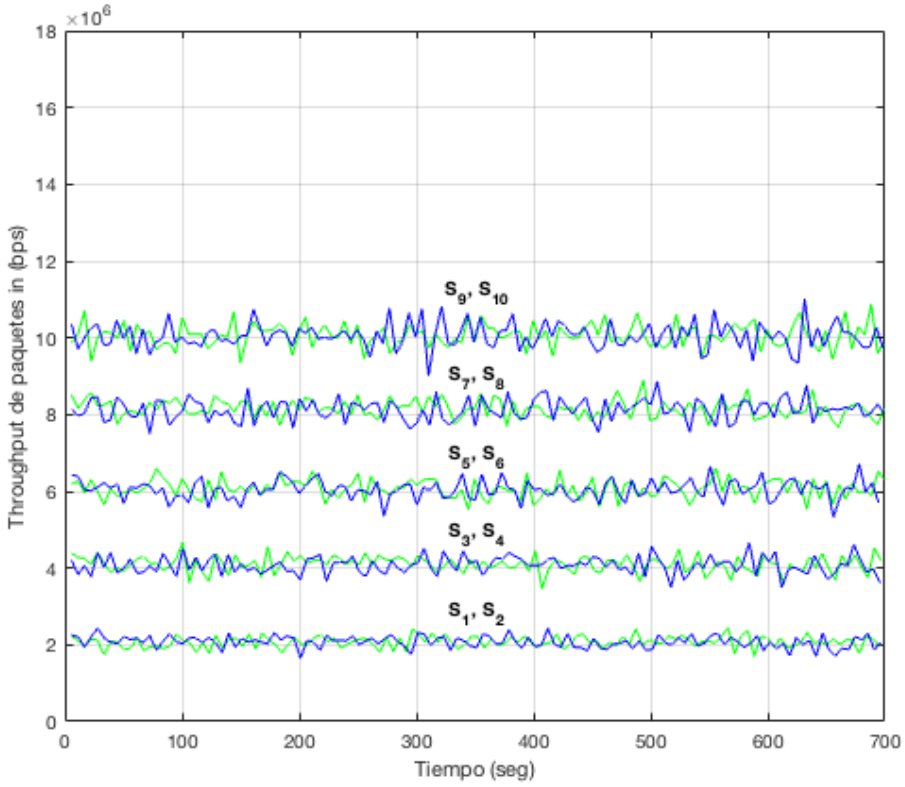

Figura 2.8: TSW - Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 20 ms para todas las conexiones)

Src	RTT *(ms)*	β	Contrato *(Mbps)*	Caudal *in* *(Mbps)*	Caudal Total *(Mbps)*	Ancho Banda Exceso	
						Mbps	%
1	20	1,75	2,00	2,09	5,96	3,87	9,67 %
2	20	1,75	2,00	2,08	5,95	3,87	9,67 %
3	20	1,60	4,00	4,10	8,14	4,04	10,10 %
4	20	1,60	4,00	4,14	7,97	3,83	9,57 %
5	20	1,55	6,00	6,08	9,96	3,88	9,71 %
6	20	1,55	6,00	6,08	9,93	3,84	9,61 %
7	20	1,50	8,00	8,15	11,88	3,73	9,32 %
8	20	1,50	8,00	8,17	11,83	3,66	9,15 %
9	20	1,45	10,00	10,05	13,60	3,55	8,89 %
10	20	1,45	10,00	10,07	13,54	3,47	8,68 %
		Total	60,00	61,01	98,76	37,75	94,37 %

Tabla 2.2: TSW - Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 20 ms para todas las conexiones)

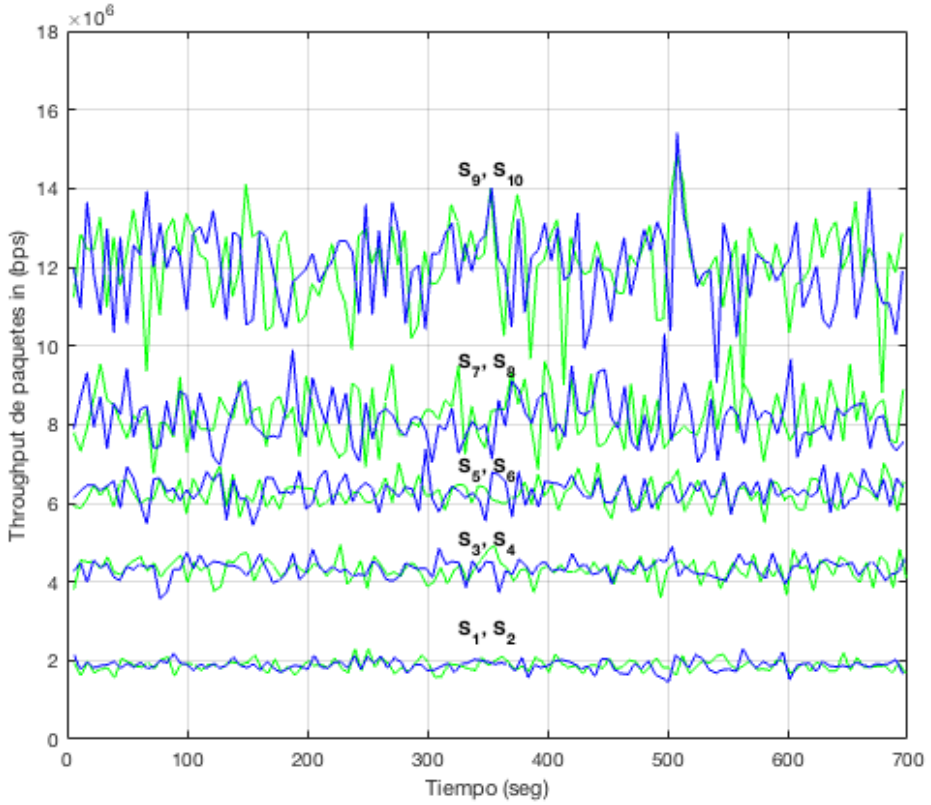

Figura 2.9: TSW - Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms)

Src	**RTT** ***(ms)***	β	**Contrato** ***(Mbps)***	**Caudal** ***in*** ***(Mbps)***	**Caudal Total** ***(Mbps)***	**Ancho Banda Exceso**	
						Mbps	%
1	10	1,75	2,00	1,86	8,38	6,52	16,31 %
2	10	1,75	2,00	1,88	8,13	6,25	15.63 %
3	20	1,60	4,00	4,35	8,95	4,60	11,51 %
4	20	1,60	4,00	4,36	8,88	4,52	11,30 %
5	40	1,55	6,00	6,38	8,69	2,32	5,79 %
6	40	1,55	6,00	6,31	8,64	2,33	5,83 %
7	60	1,50	8,00	8,25	9,63	1,38	3,44 %
8	60	1,50	8,00	8,18	9,48	1,30	3,26 %
9	80	1,45	10,00	12,83	13,60	0,69	1,71 %
10	80	1,45	10,00	12,75	13.54	0.67	1,69 %
		Total	60,00	65,79	96,37	30,58	76,46 %

Tabla 2.3: TSW - Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms)

Como puede comprobarse el acondicionador de tráfico TSW-RIO manifiesta algunas dificultades de diseño. En líneas generales, en este caso, el caudal de paquetes *in* es mucho más irregular en el algoritmo TSW que para otras propuestas que analizaremos más adelante. Además de esto hay que destacar que el caudal de paquetes *in* no cumple, en algunos casos, con los contratos fijados tal y como puede observarse en la Tabla 2.3. Además, se puede comprobar que las fuentes de tráfico con un menor RTT y menor contrato se ven más favorecidas que el resto.

En la Figura 2.10 aparece representado el valor del índice de justicia para cada uno de los escenarios descritos en la sección 1.3 con una variación de la carga de la red entre el 10 % y el 90 %. A partir de esta figura, se deduce que en las condiciones descritas en los escenarios A y B, el acondicionador tiene unos valores muy superiores a 0.9. Sin embargo en un escenario más heterogéneo como el C, el reparto del ancho de banda deja de ser justo y además empeora a medida que aumenta la carga de la red. Esto puede deberse a las propias características de las conexiones TCP, puesto que para aquellas conexiones con RTT más alto transcurre más tiempo antes de recibir los reconocimientos, lo que además aumenta el tiempo de detección de pérdida de un paquete y retransmisión del mismo. Por otra parte, también se ha demostrado en [8] y en [12] que las fuentes de menor contrato son capaces de introducir en la red un mayor número de paquetes marcados como *out*.

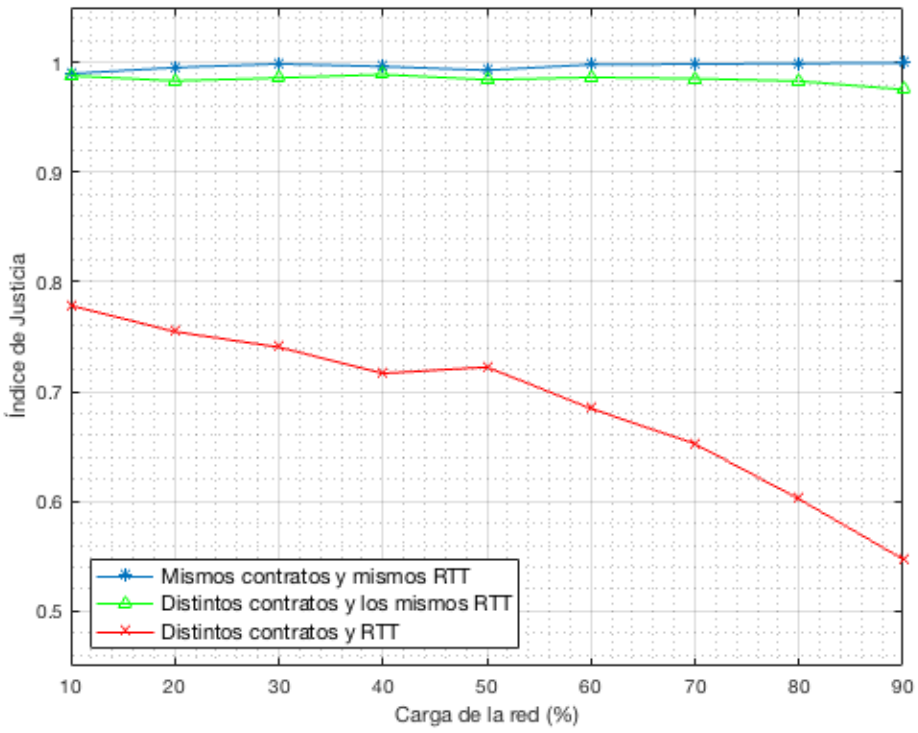

Figura 2.10: TSW - Variación del índice de justicia para los escenarios de simulación A, B y C con una variación de la carga de la red entre el 10 % y 90 %

2.3. Acondicionadores de Tráfico Basados en *Tokens*

Los acondicionadores más empleados para las implementaciones software son el *Token Bucket* de dos y tres niveles de precedencia, ya que aunque puedan resultar relativamente complejos, permiten mayor versatilidad a la hora de definir los perfiles de tráfico. Un claro ejemplo de algoritmo de acondicionado de tráfico basado en *Token Bucket* con dos niveles de precedencia es el algoritmo propuesto en [8] , *Counters Based*, y cuyas prestaciones se analizaran en esta sección. En cuanto a acondicionadores con tres niveles de precedencia basados en *Token Bucket*, los más destacables son SRTCM (*Single Rate Three Color Marker*) y TRTCM (*Two Rate Three Color Marker*) definidos en [24] y [25] respectivamente.

2.3.1. Acondicionador de Tráfico *Token Bucket*

El algoritmo *Token Bucket* dispone de un generador de testigos que produce una tasa constante de CIR testigos por segundo y los deposita en el cubo de testigos cuya capacidad máxima viene dada por CBS testigos. Si el cubo de testigos se llena, los nuevos testigos que son generados son descartados automáticamente (ver figuras 2.12 y 2.11).

```
Inicialmente:
numero_tokens = 0;
instante_ultima_llegada = 0;
CBS = constante;

En cada llegada de un paquete:
nuevos_tokens = CIR * (ahora - instante_ultima_llegada);

if numero_tokens + nuevos_tokens ≤ CBS
numero_tokens += nuevos_tokens;
else
numero_tokens = CBS;

instante_ultima_llegada = ahora;

if (numero_tokens ≥ tamaño_paquete){
numero_token -= tamaño_paquete;
marcar paquete como in;
}

else
marcar paquete como out;
```

Figura 2.11: Pseudo-código del algoritmo Token Bucket

Dado un paquete de longitud *L* bytes, éste se considerará conforme a un determinado perfil únicamente si hay suficientes testigos disponibles en el cubo ($L \leq$ b testigos). En este caso, el paquete se marcará como *in* y se retirarán *L* testigos, quedando en este caso un total de $b - L$ testigos. En caso contrario, el paquete se marcará como *out*, y el número de testigos permanecerá inalterado.

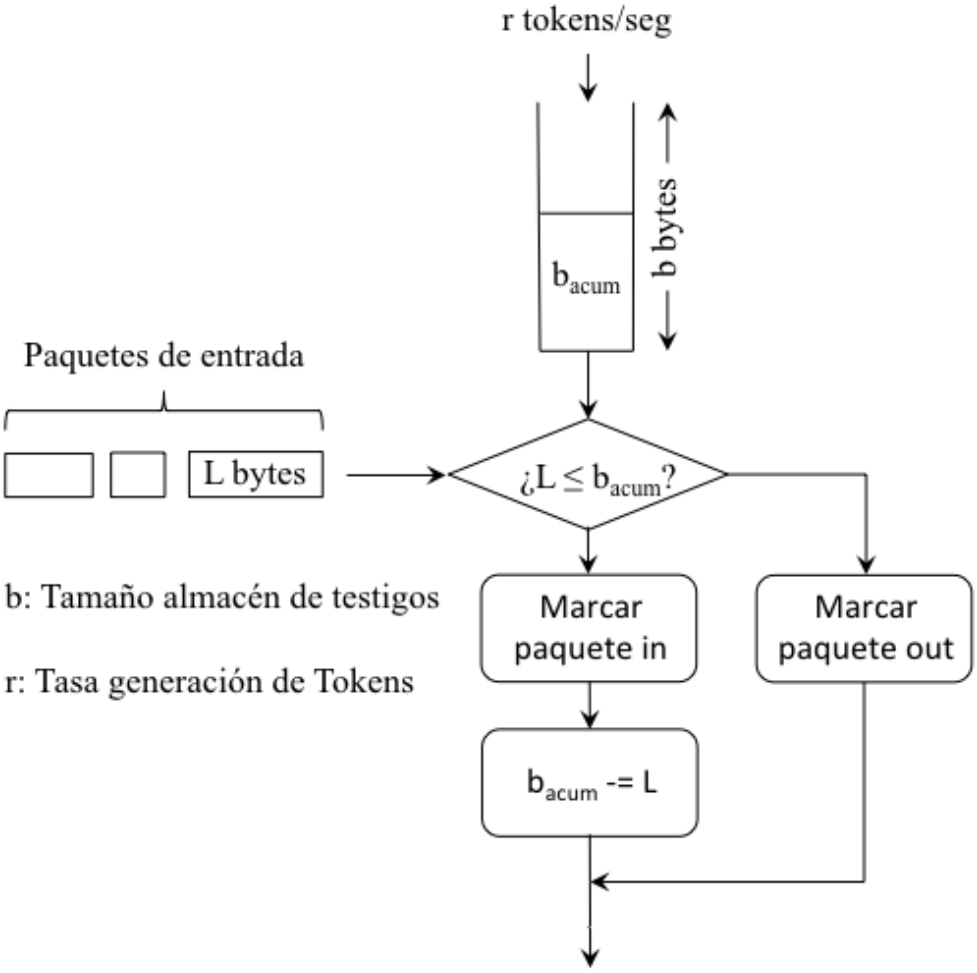

Figura 2.12: Acondicionador de tráfico Token Bucket

Como puede comprobarse el número de paquetes marcados como *in* o como *out* está directamente relacionado con la capacidad máxima del cubo de testigos. A mayor capacidad, mayor será el número de paquetes marcados como *in*. Si se desea garantizar de modo estricto el caudal contratado por las distintas fuentes de tráfico, el tamaño de este cubo debe cumplir la relación 2.2.

$$CBS = \frac{CIR(bps)}{1\ seg} = CIR\ bits = \frac{CIR}{8}\ bytes \tag{2.2}$$

En las figuras 2.13 y 2.14 se representa el caudal de paquetes *in* alcanzado por las fuentes de tráfico S_1 y S_{10} para los algoritmos Token Bucket y TSW. La topología empleada en este caso es la de la Figura 1.11 con unos valores de los contratos de 2, 2, 4, 4, 6, 6, 8, 8, 10 y 10 Mbps de S_1 a S_{10}. Las simulaciones se realizaron variando el RTT en las distintas simulaciones pero manteniéndolo igual para todas las fuentes. En líneas generales, y como se observa en los resultados obtenidos, el caudal de paquetes *in* es mucho más irregular en el algoritmo TSW que en el *Token Bucket*. Además de esto hay que destacar que, en muchos casos, el caudal de paquetes *in* no cumple con los contratos fijados de manera tan estricta como lo hace el algoritmo *Token Bucket*.

Por otra parte, tal y como se refleja en la tercera y cuarta columna de la tabla 2.4, cuando todas las fuentes tienen un RTT similar, los contratos se alcanzan con independencia del valor que éste tenga. En caso de diversidad de RTT (ver tablas 2.5 y 2.6), los resultados obtenidos son también satisfactorios, dado que el caudal de paquetes *in* cumple también con los contratos fijados. Estas características se pueden intuir también a partir de la evolución temporal del caudal de paquetes *in* en cada uno de los algoritmos y para los escenarios propuestos (ver figuras 2.15, 2.16, 2.17, 2.7, 2.8 y 2.9). Estos hechos, son los que permiten afirmar que el algoritmo *Token Bucket* ofrece mejor servicio.

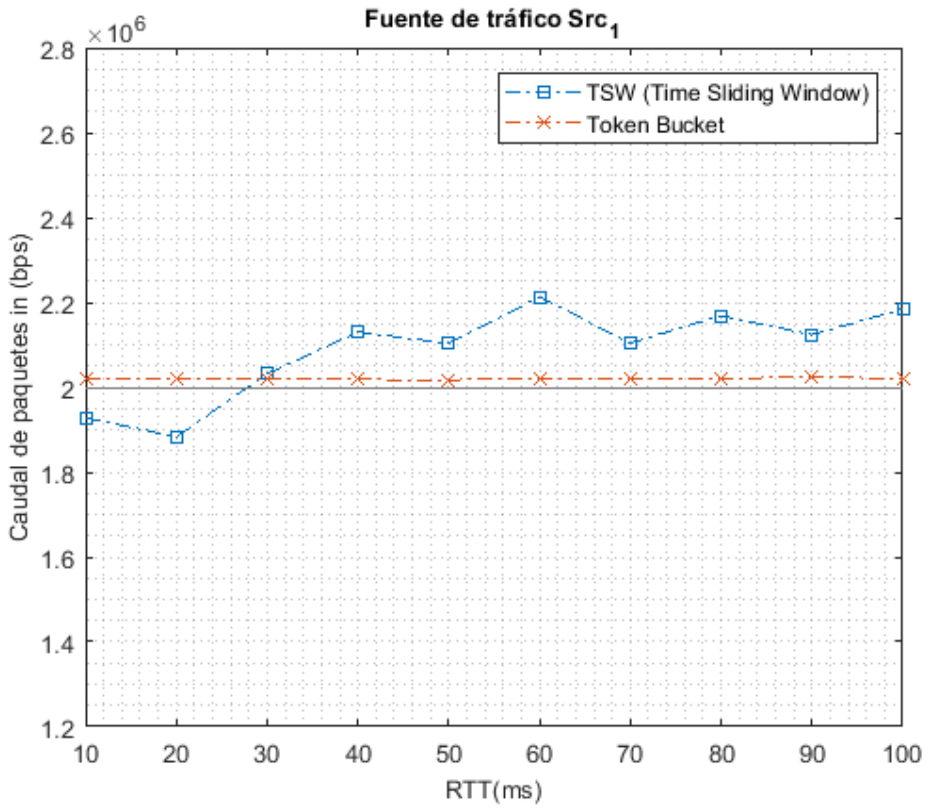

Figura 2.13: Throughput de paquetes *in* para los algoritmos Token Bucket y TSW en función del RTT. Fuente de tráfico TCP S_1 y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps)

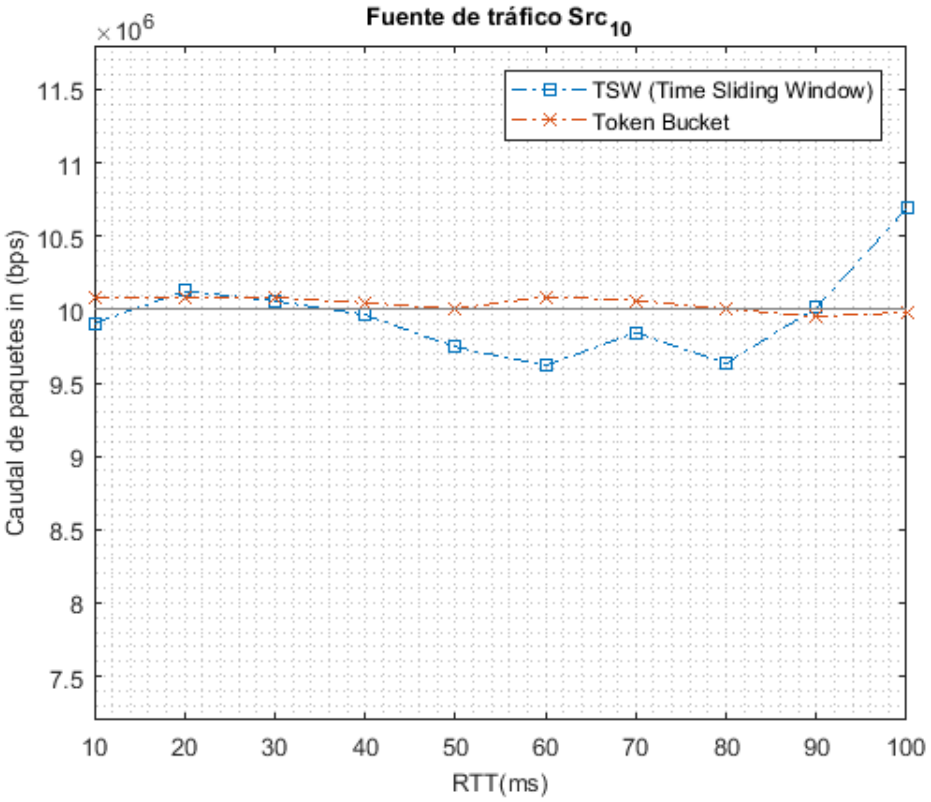

Figura 2.14: Throughput de paquetes *in* para los algoritmos Token Bucket y TSW en función del RTT. Fuente de tráfico TCP S_{10} y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps)

En cuanto a la distribución del ancho de banda excedente, se observa en las Tabla 2.4, que en el caso en que todas las fuentes tienen un RTT similar, el exceso de ancho de banda se reparte de modo equitativo entre las fuentes. Los resultados son incluso levemente más justos que los obtenidos con el algoritmo *TSW*. Sin embargo, en las tablas 2.5 y 2.6 se aprecia que hay una clara carencia de justicia cuando las fuentes tienen RTT distintos como ocurría con el algoritmo anterior, donde por lo general, aquellas fuentes con un menor RTT se veían más favorecidas. Tal y como puede observase en la Tabla 2.5, este reparto del ancho de banda se vuelve más injusto cuando las fuentes de menor contrato son las que tienen un menor RTT (ver Figura 2.18, en la que se representa el valor del índice de justicia en función de la carga de la red).

La conclusión a la que se llega en este caso, es que aquellas fuentes con un menor RTT se ven más favorecidas. Este inconveniente se debe, tal y como se comenta en la sección 2.2, al propio comportamiento de las fuentes TCP [8], dado que para aquellas conexiones con RTT más alto transcurre más tiempo antes de recibir los reconocimientos, lo que además contribuye a un aumento en el tiempo de detección de pérdida de un paquete y retransmisión del mismo, es decir, aquellas fuentes con un mayor RTT requieren más tiempo para recuperarse de las pérdidas de paquetes [32].

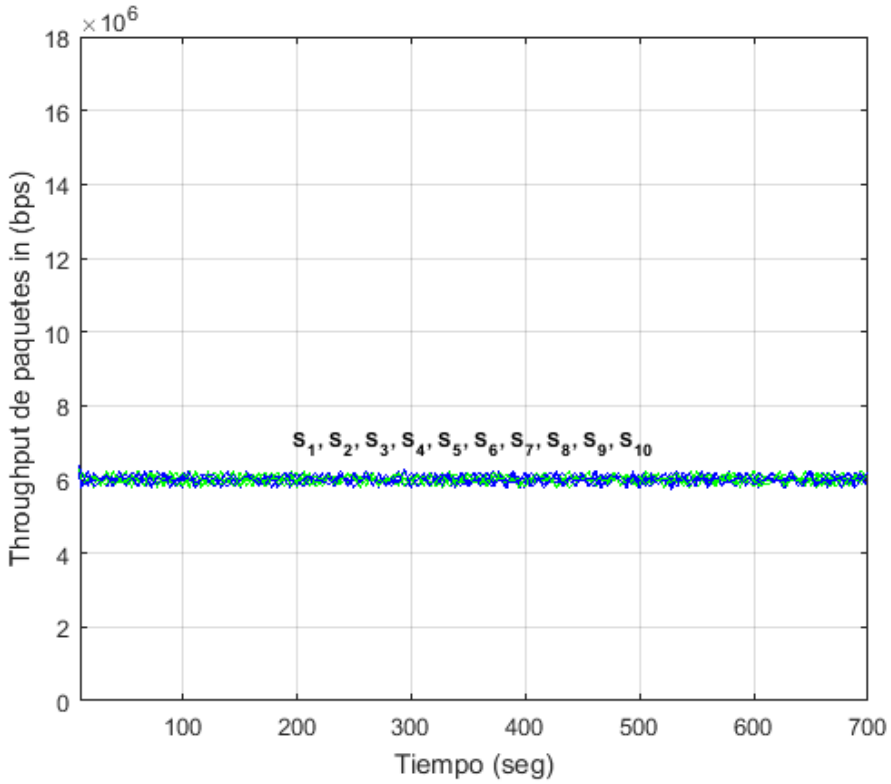

Figura 2.15: Token Bucket - Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 20 ms para todas las conexiones)

Src	RTT *(ms)*	Contrato *(Mbps)*	Caudal *in* *(Mbps)*	Caudal Total *(Mbps)*	Ancho Banda Exceso	
					Mbps	%
1	20	6,00	6,06	9,74	3,68	9,36 %
2	20	6,00	6,07	9,79	3,72	9,47 %
3	20	6,00	6,07	9,75	3,68	9,37 %
4	20	6,00	6,07	9,75	3,68	9,37 %
5	20	6,00	6,07	9,72	3,65	9,29 %
6	20	6,00	6,07	9,64	3,57	9,08 %
7	20	6,00	6,08	9,75	3,68	9,35 %
8	20	6,00	6,08	9,65	3,57	9,10 %
9	20	6,00	6,07	9,78	3,71	9,44 %
10	20	6,00	6,07	9,67	3,60	9,17 %
	Total	60,00	60,71	97,25	36,54	93,00 %

Tabla 2.4: Token Bucket - Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 20 ms para todas las conexiones)

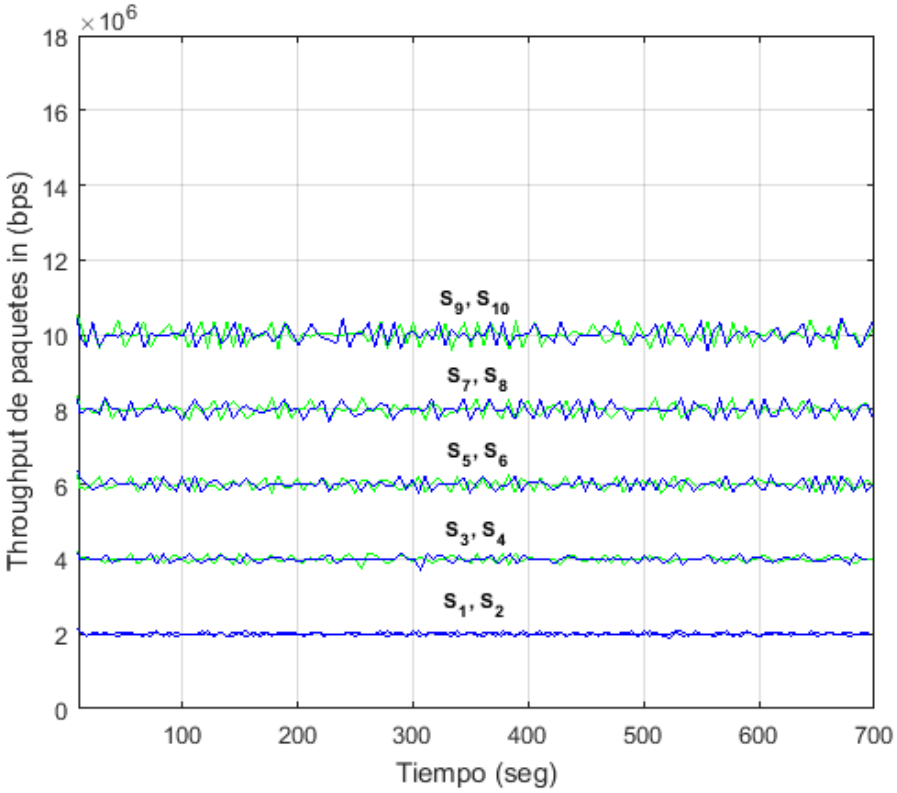

Figura 2.16: Token Bucket - Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 20 ms para todas las conexiones)

Src	RTT *(ms)*	Contrato *(Mbps)*	Caudal *in* *(Mbps)*	Caudal Total *(Mbps)*	Ancho Banda Exceso	
					Mbps	%
1	20	2,00	2,02	6,29	4,27	10,86 %
2	20	2,00	2,02	6,35	4,33	11,01 %
3	20	4,00	4,05	7,86	3,81	9,70 %
4	20	4,00	4,05	7,97	3,92	9,96 %
5	20	6,00	6,07	9,75	3,68	9,34 %
6	20	6,00	6,08	9,69	3,61	9,18 %
7	20	8,00	8,10	11,47	3,37	8,57 %
8	20	8,00	8,09	11,42	3,33	8,47 %
9	20	10,00	10,10	13,45	3,35	8,52 %
10	20	10,00	10,09	13,33	3,24	8,24 %
	Total	60,00	60,67	97,58	36,91	93,85 %

Tabla 2.5: Token Bucket - Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 20 ms para todas las conexiones)

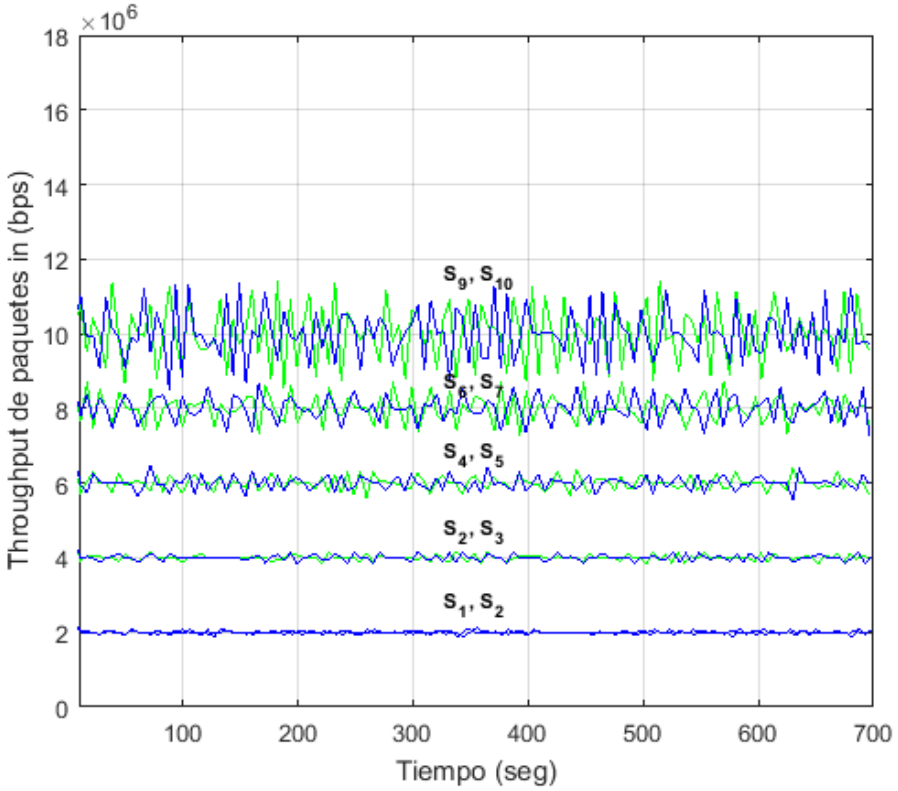

Figura 2.17: Token Bucket - Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms)

Src	RTT *(ms)*	Contrato *(Mbps)*	Caudal *in* *(Mbps)*	Caudal Total *(Mbps)*	Ancho Banda Exceso	
					Mbps	%
1	20	2,00	2,03	9,26	7,23	18,40 %
2	20	2,00	2,03	9,49	7,46	18,98 %
3	20	4,00	4,06	9,18	5,11	13,00 %
4	20	4,00	4,06	9,15	5,09	12,95 %
5	20	6,00	6,08	8,65	2,57	6,54 %
6	20	6,00	6,07	8,85	2,78	7,07 %
7	20	8,00	8,09	9,55	1,46	3,73 %
8	20	8,00	8,07	9,45	1,38	3,50 %
9	20	10,00	10,08	10,92	0,83	2,12 %
10	20	10,00	10,10	10,85	0,76	1,92 %
	Total	60,00	60,68	95,36	34,69	88,21 %

Tabla 2.6: Token Bucket - Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Contratos: 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms)

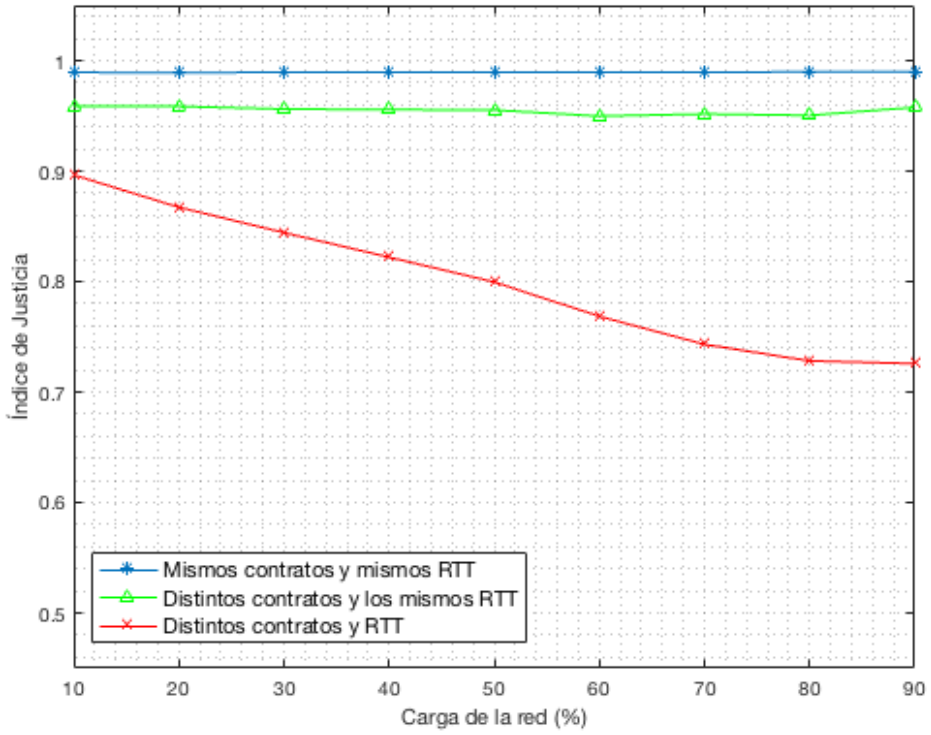

Figura 2.18: Token Bucket - Variación del índice de justicia para los escenarios de simulación A, B y C con una variación de la carga de la red entre el 10 % y 90 %

2.3.2. Acondicionador de Tráfico CB *(Counters Based)*

Tal y como se ha comentado anteriormente, uno de los acondicionadores de tráfico que se va a analizar en este capítulo es el propuesto en [8], denominado *Counters Based* (CB). Tal y como puede apreciarse en la Figura 2.19, el algoritmo basa su funcionamiento en dos contadores, denominados C1 y C2. C1 se inicializa a cero con el propósito que el primer paquete que llegue al acondicionador se marque siempre como *in*. Por otra parte, C2 inicialmente se configura como la relación entre la capacidad del enlace y el contrato de la fuente de tráfico. Cada unidad de tiempo, C2 decrece una unidad. Si C2 alcanza un valor nulo, se incrementa C1 en una unidad y C2 vuelve a su valor inicial. Cuando se produce la llegada de un paquete, se comprueba el valor de C1. Si C1 es mayor que cero, entonces el paquete se marca como *in* y se resta una unidad a C1. En caso contrario, el paquete se marca como *out*.

En las figuras 2.20 y 2.21 se pone de manifiesto la superioridad que presenta este algoritmo frente a los dos anteriores, pues el caudal de paquetes *in* garantiza de modo estricto los contratos de las distintas fuentes de tráfico con independencia del contrato y RTT de cada una de ellas. De manera más concreta, se puede apreciar, que el caudal de paquetes *in* garantiza los contratos con apenas variaciones del 1 %.

```
Inicialmente:
C1 = 1;
C2 = velocidad enlace/contrato;

Para cada unidad de tiempo:
C2--;

if(C2 ≤ 0){
   C1++;
   C2 = velocidad enlace/contrato;
}

if llega un paquete
   if C1 > 0
      marcar paquete como in
   else
      marcar paquete como out
```

Figura 2.19: Pseudo-código del algoritmo *Counters Based*

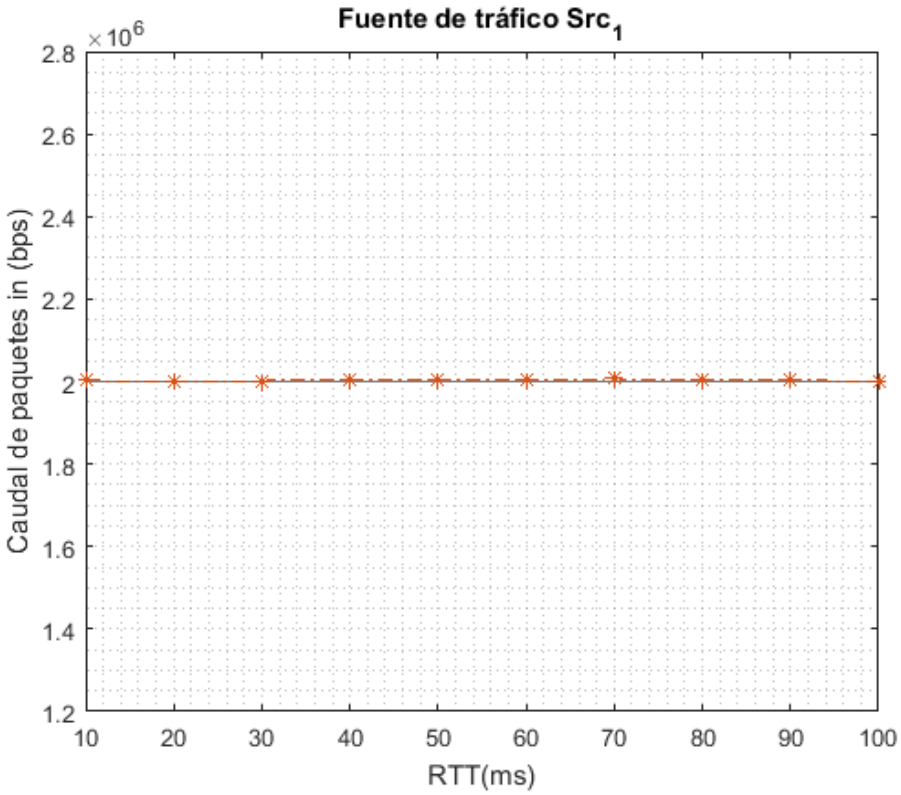

Figura 2.20: Throughput de paquetes *in* para el algoritmo CB en función del RTT. Fuente de tráfico TCP S_1 y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps)

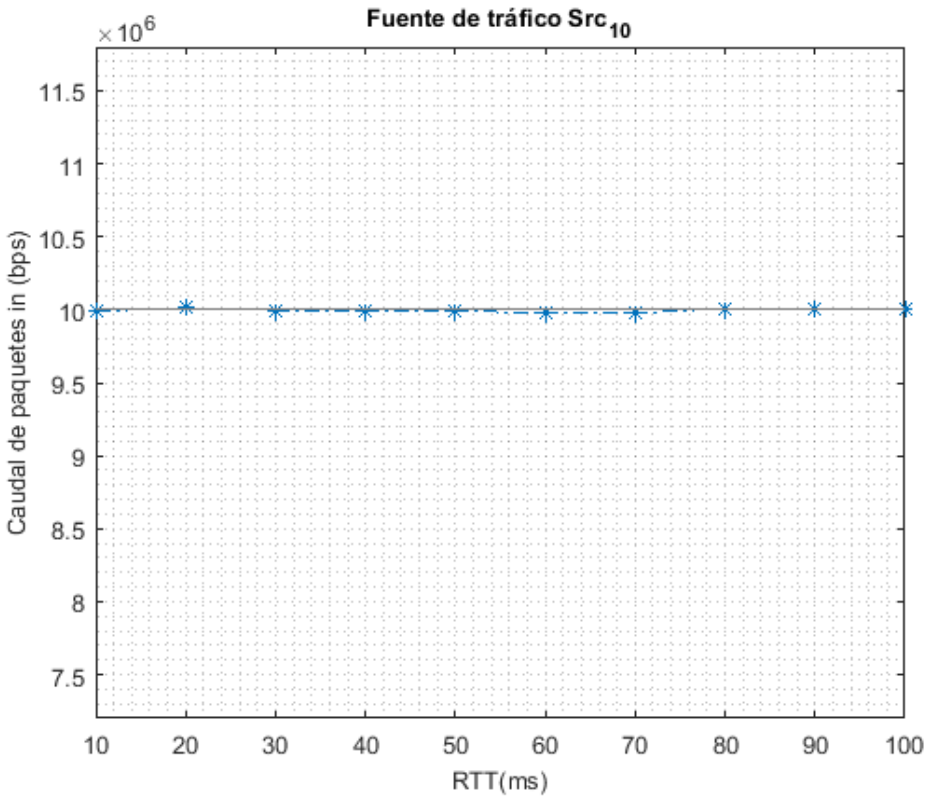

Figura 2.21: Throughput de paquetes *in* para el algoritmo CB en función del RTT. Fuente de tráfico TCP S_{10} y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps)

En las figuras 2.22, 2.23 y 2.24 aparecen reflejada la evolución temporal del caudal de paquetes *in* de todas las fuentes de tráfico para los escenarios A, B y C respectivamente. Por otra parte en las tablas 2.7, 2.8 y 2.9 se recogen los valores del caudal de paquetes *in*, caudal total y exceso de ancho de banda consumido por cada una de ellas (en Mbps y porcentaje sobre el total del ancho de banda excedente). Estos datos y los mostrados en las figuras 2.20 y 2.21 ponen de manifiesto la superioridad que presenta este algoritmo frente al analizado en la sección 2.2 y en menor medida con el presentado en la sección 2.3.1.

En cuanto a la distribución del ancho de banda excedente, se observa en las Tabla 2.7, que en el caso en que todas las fuentes tienen un RTT similar, el exceso de ancho de banda se reparte de modo equitativo entre las fuentes. Los resultados son incluso levemente más justos que los obtenidos con el algoritmo *TSW*. Sin embargo, en las tablas 2.8 y 2.9 se aprecia que hay una clara carencia de justicia cuando las fuentes tienen RTT distintos como ocurría con el algoritmo anterior, donde por lo general, aquellas fuentes con un menor RTT se veían más favorecidas. Tal y como puede observase en la Tabla 2.8, este reparto del ancho de banda se vuelve más injusto cuando las fuentes de menor contrato son las que tienen un menor RTT (ver Figura 2.25).

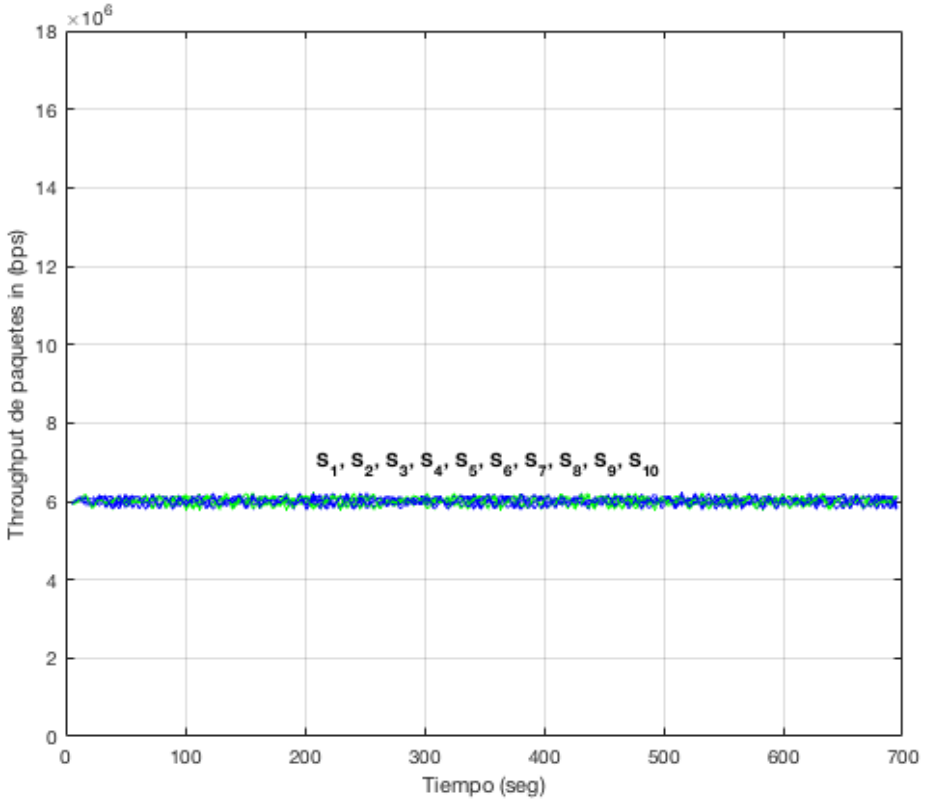

Figura 2.22: CB - Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 20 ms para todas las conexiones)

Src	RTT *(ms)*	Contrato *(Mbps)*	Caudal *in* *(Mbps)*	Caudal Total *(Mbps)*	Ancho Banda Exceso	
					Mbps	%
1	20	6,00	6,00	9,78	3,78	9,45 %
2	20	6,00	6,00	9,71	3,71	9,28 %
3	20	6,00	6,00	9,77	3,77	9,42 %
4	20	6,00	6,00	9,60	3,60	9,00 %
5	20	6,00	6,00	9,84	3,84	9,60 %
6	20	6,00	6,00	9,90	3,90	9,74 %
7	20	6,00	6,00	9,75	3,75	9,37 %
8	20	6,00	6,00	9,79	3,79	9,47 %
9	20	6,00	6,00	9,72	3,72	9,29 %
10	20	6,00	6,00	9,75	3,75	9,38 %
	Total	60,00	60,01	97,61	37,59	93,99 %

Tabla 2.7: CB - Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 20 ms para todas las conexiones)

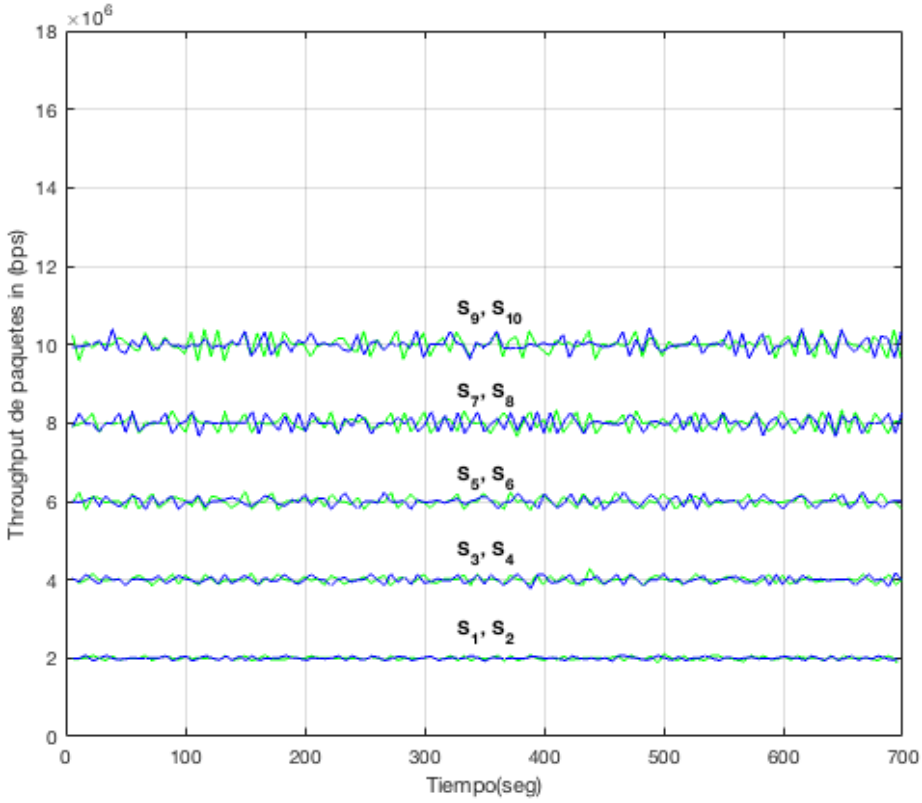

Figura 2.23: CB - Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 20 ms para todas las conexiones)

Src	**RTT** *(ms)*	**Contrato** *(Mbps)*	**Caudal** *in* *(Mbps)*	**Caudal Total** *(Mbps)*	**Ancho Banda Exceso**	
					Mbps	%
1	20	2,00	2,00	6,25	4,25	10,63 %
2	20	2,00	2,00	6,34	4,34	10,85 %
3	20	4,00	4,00	8,03	4,03	10,07 %
4	20	4,00	4,00	7,92	3,92	9,80 %
5	20	6,00	6,00	9,74	3,73	9,34 %
6	20	6,00	6,01	9,85	3,84	9,60 %
7	20	8,00	8,00	11,52	3,53	8,81 %
8	20	8,00	8,00	11,52	3,52	8,80 %
9	20	10,00	10,00	13,42	3,42	8,54 %
10	20	10,00	10,00	13,37	3,37	8,42 %
	Total	60,00	60,02	97,95	37,33	94,84 %

Tabla 2.8: CB - Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 20 ms para todas las conexiones)

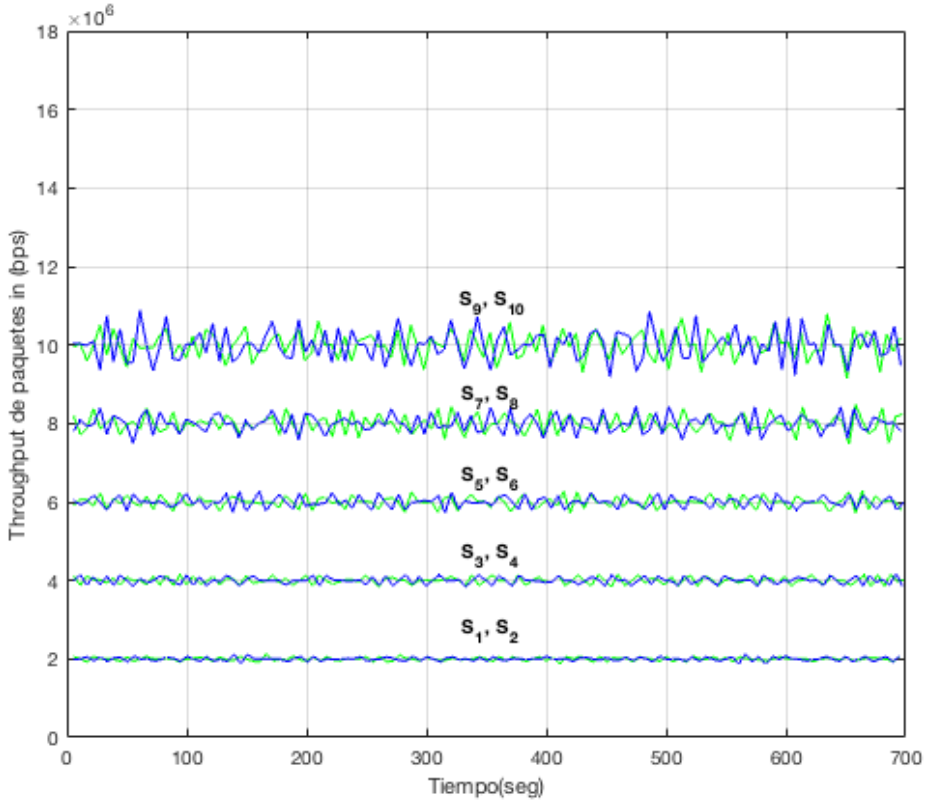

Figura 2.24: CB - Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms)

Src	RTT *(ms)*	Contrato *(Mbps)*	Caudal *in* *(Mbps)*	Caudal Total *(Mbps)*	Ancho Banda Exceso	
					Mbps	%
1	20	2,00	2,00	9,26	7,26	18,15 %
2	20	2,00	2,00	9,49	7,49	18,73 %
3	20	4,00	4,00	9,45	5,45	13,62 %
4	20	4,00	4,01	9,35	5,34	13,36 %
5	20	6,00	6,00	8,55	2,55	6,37 %
6	20	6,00	6,00	8,61	2,60	6,51 %
7	20	8,00	8,01	9,40	1,39	3,48 %
8	20	8,00	8,00	9,38	1,38	3,45 %
9	20	10,00	9,99	10,63	0,64	1,61 %
10	20	10,00	9,99	10,76	0,76	1,90 %
	Total	60,00	60,01	94,88	34,87	87,19 %

Tabla 2.9: CB - Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms)

En la Figura 2.25 aparece representado el valor del índice de justicia para cada uno de los escenarios descritos en la sección 1.3 con una variación de la carga de la red entre el 10 % y el 90 %. A partir de esta figura, y al igual que ocurría con los acondicionadores analizados en las secciones anteriores, se deduce que en las condiciones descritas en los escenarios A y B, el acondicionador tiene unos valores muy superiores a 0.9. Sin embargo en un escenario más heterogéneo como el C, el reparto del ancho de banda deja de ser justo y además empeora a medida que aumenta la carga de la red.

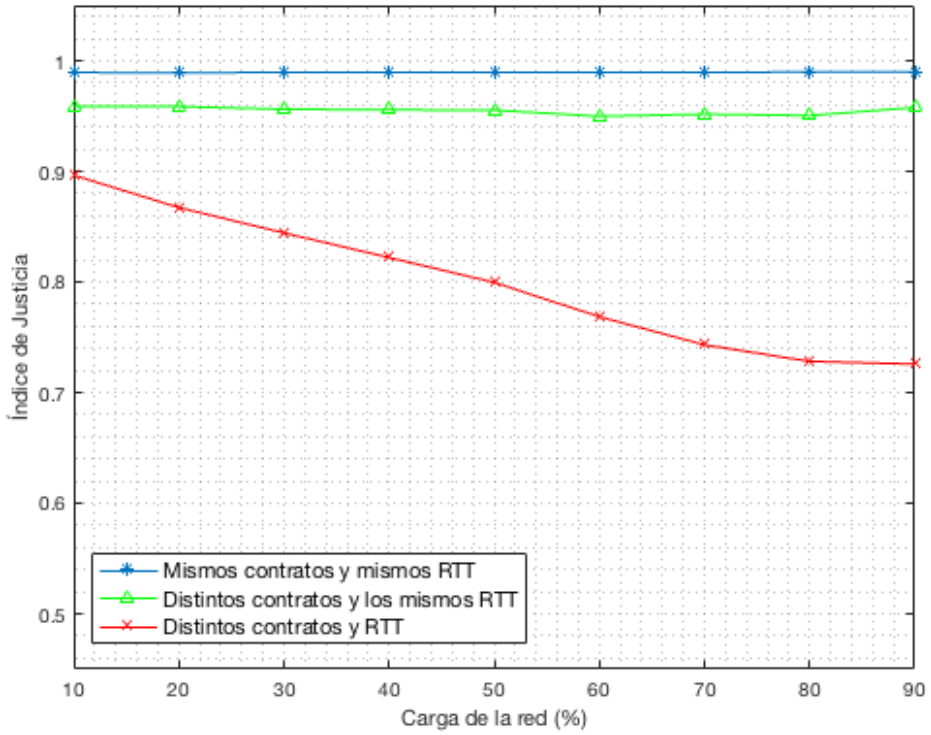

Figura 2.25: CB - Variación del índice de justicia para los escenarios de simulación A, B y C con una variación de la carga de la red entre el 10 % y 90 %

2.4. Conclusiones

En este capítulo se ha analizado mediante simulación las prestaciones de los acondicionadores de tráfico TSW, *Token Bucket* y CB, con el objetivo de deducir la capacidad que tiene cada uno de ellos para garantizar los objetivos del Servicio Asegurado. Para llevar a cabo este análisis se ha empleado la topología mostrada en la Figura 1.11 que consta de 10 fuentes de tráfico Reno multiplexadas con contratos fijados en cada uno de los escenarios de simulación considerados.

En relación al acondicionador TSW, pese a ser un algoritmo duramente criticado en [32], se ha considerado interesante realizar un análisis con el fin de comprobar las

posibilidades que puede aportar. Una de las críticas principales que pueden encontrarse en [32] argumenta que no resulta adecuado para aplicaciones que no funcionen sobre TCP, puesto que el número de paquetes que se marcan como *in* se encuentra muy por encima de lo que realmente le correspondería. En este capítulo, aparte del análisis de prestaciones, se ha descrito el procedimiento empleado para conseguir una configuración relativamente óptima. Sin embargo, puede apreciarse que a diferencia del algoritmo *Token Bucket* o *Counters Based*, el caudal de paquetes *in* no garantiza de un modo tan exacto los contratos de las distintas fuentes de tráfico, si no que más bien presenta ciertas irregularidades.

Para el algoritmo *Token Bucket*, se observaba que el caudal de paquetes *in* alcanzaba o excedía mínimamente los contratos para cada una de las fuentes. Así mismo se demostró la capacidad que tiene para garantizar los contratos aún en situaciones en la que exista disparidad de RTT, sin embargo se llegó a la conclusión que el acondicionador de tráfico carecía de justicia, puesto que aquellas fuentes con un RTT menor se veían más favorecidas.

Finalmente, se estudió el acondicionador de tráfico *Counters Based*. Una de las características más reseñables de este algoritmo es la sencillez, pues carece de parámetros configurables. En cuanto al análisis de prestaciones se pone de manifiesto la superioridad que presenta este este algoritmo frente a los dos anteriores, dado que el caudal de paquetes *in*, garantiza de modo estricto los contratos de las fuentes de tráfico con apenas variaciones del 1 %. Sin embargo, a la hora de realizar un reparto equitativo del ancho de banda excedente, el algoritmo carece de justicia.

De manera general, se ha podido verificar que ninguno de los acondicionadores de tráfico anteriores es capaz de realizar un reparto equitativo del ancho de banda sobrante. En todos ellos, las fuentes de menor tráfico se ven más favorecidas en este reparto. La razón principal es que aquellas fuentes de tráfico con un menor contrato, son capaces de inyectar una mayor cantidad de paquetes *out* en la red. Por otra parte se ha podido comprobar, que aquellas fuentes con un menor RTT obtenía una mayor proporción del ancho de banda sobrante. La razón principal radica en el propio funcionamiento del protocolo TCP, donde para aquellas fuentes con RTT más alto transcurre más tiempo antes de recibir los reconocimientos, lo que además contribuye a un aumento en el tiempo de detección de pérdida de un paquete y retransmisión del mismo.

Capítulo 3

Técnicas para un Reparto Equitativo del Ancho de Banda Excedente.

Una de las principales conclusiones que se extrajeron en el capítulo anterior es que no es posible, con los acondicionadores analizados, realizar un reparto equitativo de los recursos no contratados. No obstante, de este estudio se desprende, que en caudal de paquetes *in* en el acondicionador de tráfico *Counters Based*, se garantiza de modo estricto los contratos con apenas variaciones del 1 %. Esta característica y su sencillez de configuración, no dispone de parámetros configurables, fue lo que lo hizo adecuado en [8] para el desarrollo de nuevas técnicas de acondicionado de tráfico que fuesen capaces de repartir de modo más equitativo el ancho de banda excedente. Una de estas propuestas, orientadas a conseguir un reparto equitativo de los recursos no contratados, fue el algoritmo denominado CBM (*Counters Based Modified*). Como podrá comprobarse, emplea el algoritmo *Counters Based* para llevar a cabo el marcado de paquetes, dada su sencillez. Sin embargo, debe hacerse notar que puede emplearse cualquiera algoritmos de acondicionado de tráfico analizados en el capítulo anterior.

3.1. Acondicionador de Tráfico Basado en las Condiciones de la Red. CBM (*Counter Based Modified*)

El objetivo de esta sección es el de analizar el acondiconador de tráfico CBM que basa su funcionamiento en el descarte probabilístico de paquetes fuera de perfil que realiza en función del contrato de cada fuente, del ancho de banda en exceso y de una estimación del RTT medio, de forma que cuando se utiliza CBM en combinación con RIO, consigue distribuir de manera justa el ancho de banda en exceso de la red entre las fuentes TCP que componen el agregado (entendiéndose en este caso un reparto equitativo), garantizando además de modo estricto los contratos de las fuentes. Con esta propuesta, se mitiga parcialmente el efecto que tiene sobre las fuentes TCP la diversidad en contratos y el tiempo de ida y vuelta (RTT).

3.1.1. Funcionamiento del Acondicionador

Partiendo de la suposición que todos los paquetes que se inyectan en la red tienen un tamaño similar, se puede afirmar que si las fuentes introducen el mismo número de paquetes entonces cada fuente obtiene la misma porción de ancho de banda. Extendiendo este hecho a los paquetes fuera de perfil, es posible afirmar que si todas las fuentes introducen el mismo número de paquetes *out* entonces se consigue un reparto equitativo del ancho de banda sobrante.

Como se ha podido comprobar, este comportamiento ideal se ve afectado por el diferente funcionamiento de cada fuente TCP, que se ve influenciada por el efecto de diferentes RTT o diferentes contratos de tráfico. Además, se ha debe tener en cuenta la interacción con el mecanismo RIO empleado para la gestión de las colas en los routers. En particular, y como una primera aproximación al problema, se considerarán los resultados mostrados en las figuras 3.2 y 3.3, donde se representa el número de paquetes *out* generados entre dos paquetes *in* consecutivos por las fuentes Src_1 y Src_{10} respectivamente. Para ilustrar esta situación, se ha utilizado el acondicionado de tráfico CB-RIO sobre la topología de Figura 1.11, donde cada fuente, de Src_1 a Src_{10}, contrata un ancho de banda de 2, 2, 4, 4, 6, 6, 8, 8, 10 y 10 Mbps y el RTT se establece a 20 ms en todas ellas (Escenario B). A partir de estas figuras, se llega a la conclusión que el número de paquetes *out* inyectados por la fuente con contrato más bajo es notablemente más alto. Por lo que es fácilmente deducible, que esta fuente de tráfico obtendrá más recursos de la red.

La situación descrita fue, la que motivó al desarrollo del algoritmo CBM en [8]. De manera resumida, el funcionamiento de este algoritmo es el siguiente (ver figuras 3.1 y 3.4): Cada acondicionador de tráfico, situado junto a la fuente TCP, fuera del alcance del usuario final, dispone de una variable que cuenta el número de paquetes *out* entre dos paquetes *in* consecutivos. Cada vez que un paquete se marca como *out*, el acondicionador de tráfico CBM comprueba esta variable. Si la variable no sobrepasa un umbral mínimo denominado min, entonces el paquete *out* se inyecta en la red. Si la variable excede un umbral máximo denominado max, entonces el paquete *out* se elimina. En cambio, si la variable permanece entre estos dos umbrales min y max, el paquete se descarta con una probabilidad p.

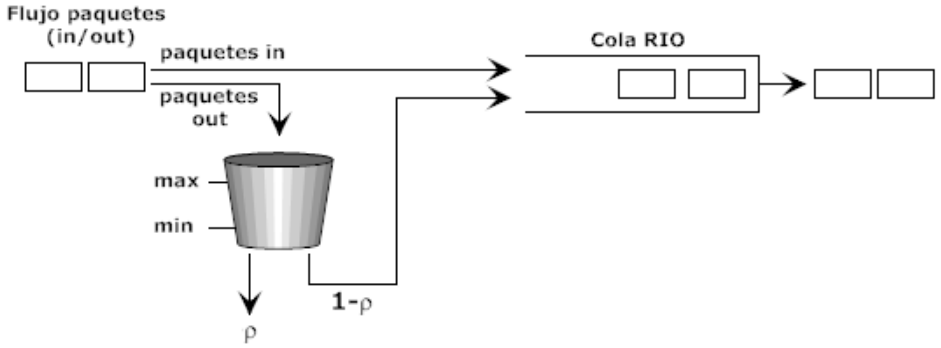

Figura 3.1: Descarte anticipado de paquetes en el acondicionador CBM

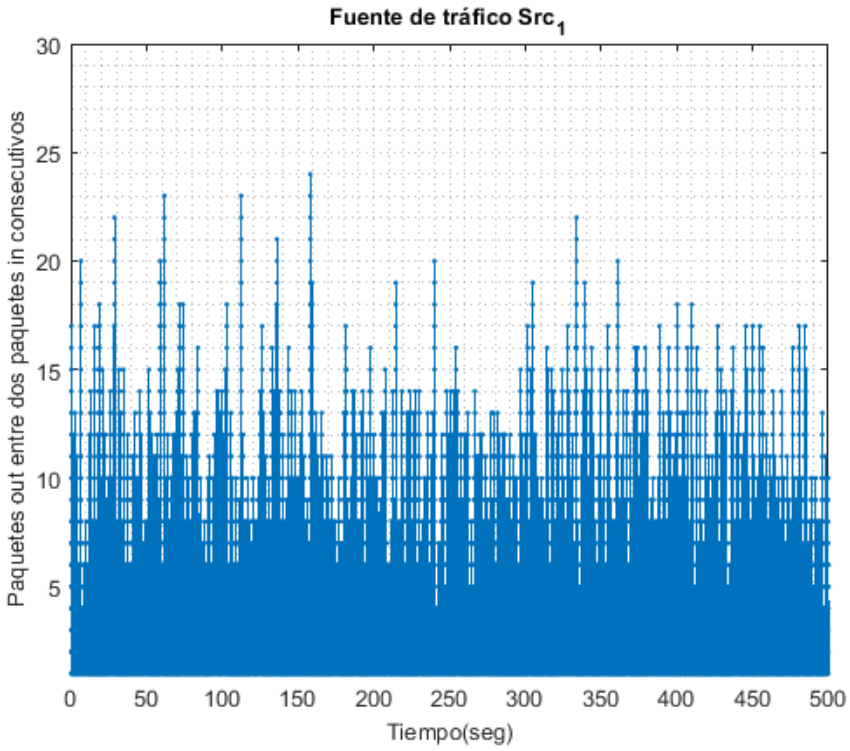

Figura 3.2: Número de paquetes *out* generados entre paquetes *in* consecutivos por la fuente Src_1

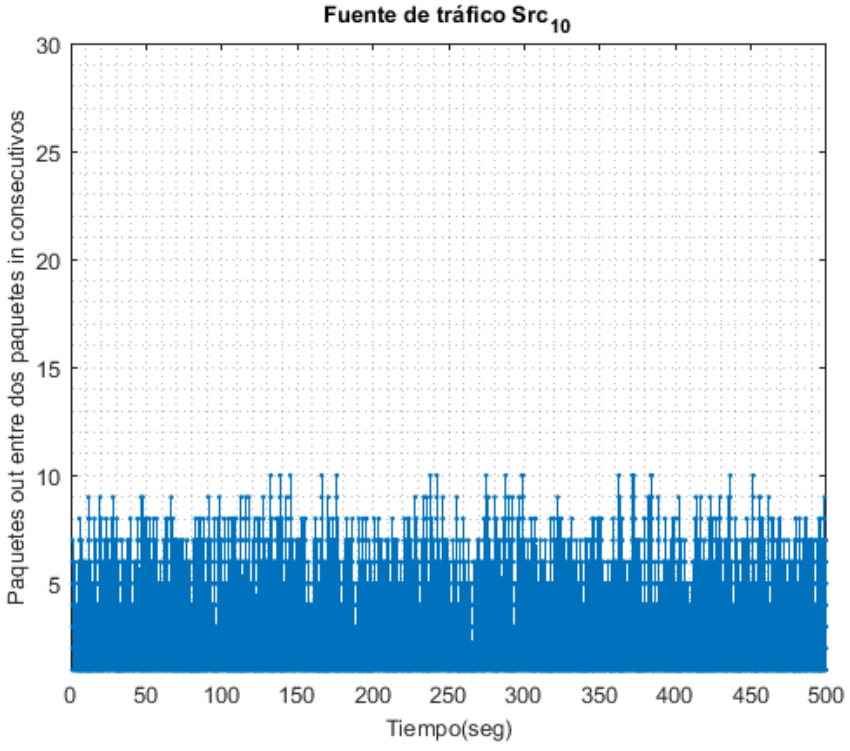

Figura 3.3: Número de paquetes *out* generados entre paquetes *in* consecutivos por la fuente Src_{10}

```
Inicialmente:
C1 = 1;
C2 = capacidad del enlace/contrato;
C3 = 0;
Calcular el valor de la probabilidad p;
Calcular los valores de los límites max y min;

Para cada unidad de tiempo:
C2--;

if (C2 ≤ 0){
    C1++;
    C2 = capacidad del enlace/contrato;
}

if (llegada de paquete){
    if (C1 > 0){
       paquete marcado como in;
       C1--;
       C3 = 0;
    }
    else{
       paquete marcado como out;
       C3++;

       if (tiempo > tiempo inicio de descarte){
         if (C3 > max)
           paquete out descartado;
         else
           if (C3 > min)
              paquete out descartado con probabilidad p;
           else
              paquete out se envía a la red;
       }
    }
}
```

Figura 3.4: Pseudo-código del algoritmo CBM

El uso de este algoritmo requiere por tanto la configuración de los límites min y max y el cálculo de la probabilidad de descarte *p*. Como se explica en [8], para obtener los valores de min y max se utilizan las ecuaciones 3.2 y 3.1, donde MSS es el acrónimo de tamaño máximo de segmento (Maximum Segment Size). El ancho de banda en exceso se puede imaginar como el correspondiente a otra fuente TCP cuya ventana máxima de transmisión fuese el producto ancho de banda en exceso por el RTT medio. De este modo, el umbral

máximo max quedaría establecido a dicho valor. Una fuente que inyectase un número de paquetes *out* cercano a este límite consumiría casi por completo el ancho de banda sobrante. Además, en el caso en el que una conexión superase dicho límite significaría que no sólo consume todo el ancho de banda en exceso sino que además podría estar robando parte del ancho de banda correspondiente al contrato garantizado de otra conexión. En consecuencia, este umbral no debe sobrepasarse nunca. Es conocido el hecho de que en la arquitectura TCP/IP, un algoritmo de crecimiento aditivo y de disminución multiplicativo satisface las condiciones necesarias de convergencia para alcanzar un estado eficiente en la red, siendo utilizado para implementar mecanismos de prevención de la congestión. Por esta razón, se ha optado por un valor mínimo min como la mitad del umbral máximo.

$$max = \left\lceil \frac{bandwidth_{exceso} \cdot RTT_{medio}}{MSS} \right\rceil \tag{3.1}$$

$$min = \left\lceil \frac{max}{2} \right\rceil \tag{3.2}$$

Finalmente, la probabilidad de descarte p se calcula mediante la expresión 3.3. Como puede observarse, cada fuente tiene un valor diferente de *p* entre 0 y 1, basándose en la tasa contratada. En [8] se concluyó, entre los diferentes valores probados, que la ecuación 3.3 resultaba ser la más apropiada para el mecanismo de descarte en CBM.

$$p = 2 \cdot \frac{\frac{tasa\ \ contratada}{capacidad\ \ enlace}}{1 + \frac{tasa\ \ contratada}{capacidad\ \ enlace}} \tag{3.3}$$

Como desventaja se puede señalar, que la probabilidad de descarte de un paquete *out* se determina asumiendo que el acondicionador de tráfico conoce la cantidad de ancho de banda en exceso y una aproximación del RTT medio de las conexiones. Este hecho implica que sea necesario algún tipo de señalización. Sin embargo, ésta es más sencilla que la empleada en otras propuestas de acondicionadores de tráfico.

Una vez introducidos los conceptos básicos del acondicionador de tráfico, en las figuras 3.5 y 3.6 se refleja el número de paquetes *out* entre paquetes *in* consecutivos una vez realizado el descarte probabilístico de paquetes fuera de perfil. Por comparación con las figuras 3.2 y 3.3, se deduce el control que tiene CBM sobre la generación de paquetes *out*. En particular, se observa que cuando la fuente de tráfico supera el límite máximo (max) de paquetes *out* entre paquetes consecutivos *in*, aquellos paquetes se descartan. Las conexiones TCP reflejan estas pérdidas disminuyendo su velocidad, y se inyecta más tráfico en exceso (paquetes *out*) de otras fuentes. Puesto que CBM es capaz de controlar el número de paquetes *out* que se introducen en la red, hecho que contribuye en este caso, a realizar un reparto más equitativo del ancho de banda en exceso.

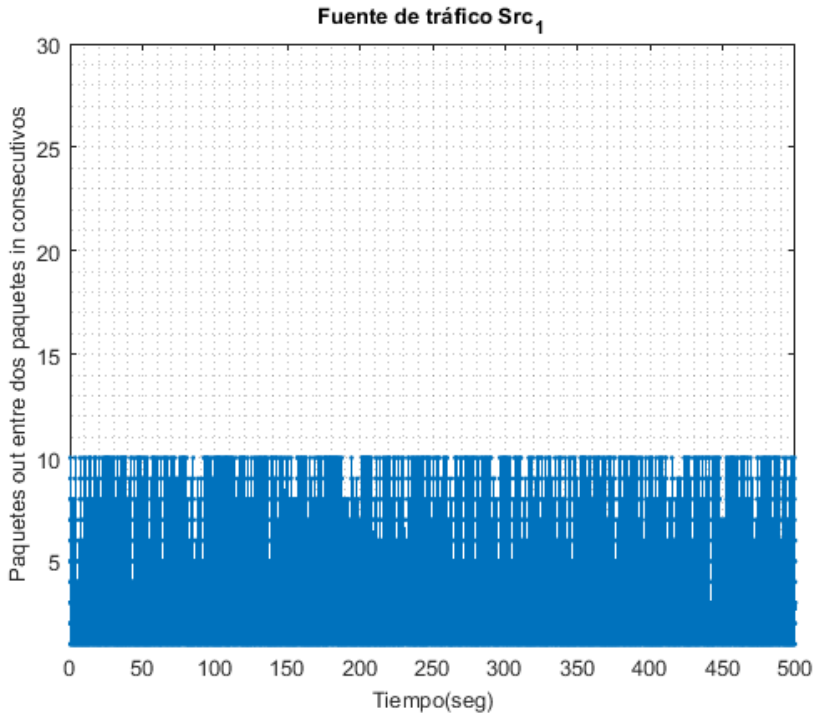

Figura 3.5: Número de paquetes *out* generados entre paquetes *in* consecutivos por la fuente Src_1 tras aplicar el descarte de parquetes

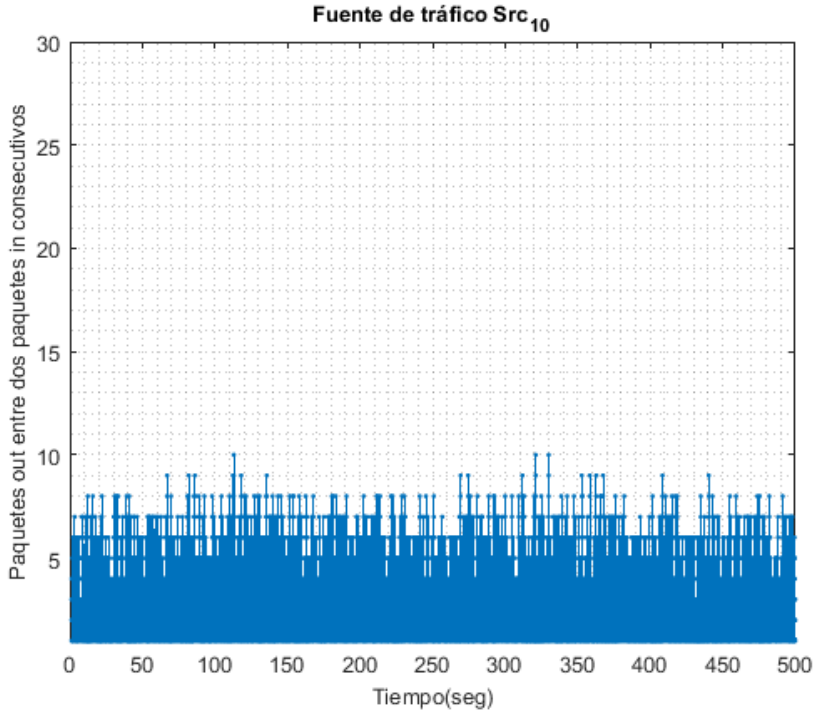

Figura 3.6: Número de paquetes *out* generados entre paquetes *in* consecutivos por la fuente Src_{10} tras aplicar el descarte de parquetes

3.1.2. Análisis de Prestaciones

Para realizar el estudio de las prestaciones del algoritmo CBM se emplearán la topología de la Figura 1.11, y variando los parámetros de cada una de las fuentes, se analizará el efecto que tiene en las prestaciones la existencia de distintos contratos y/o RTT.

En la Figura 3.7 se representa el caudal de paquetes *in* de las fuentes de menor y mayor contrato, S_1 y S_{10} respectivamente, en las condiciones descritas en el escenario C, es decir, con unos valores de los contratos de 2, 2, 4, 4, 6, 6, 8, 8, 10 y 10 Mbps de S_1 a S_{10}. Las simulaciones se realizaron variando el RTT en las distintas simulaciones pero manteniéndolo igual para todas las fuentes.

Por otra parte, en las figuras 3.8, 3.9 y 3.10 se puede observar la evolución del caudal de paquetes *in* con el tiempo para cada uno de los escenarios propuestos. Como puede comprobarse, el descarte probabilístico de paquetes *out* en el acondicionador de tráfico CBM y la interacción de éste con la gestión de cola RIO no tiene ningún efecto sobre las prestaciones en términos de garantías de contratos. Como ilustran las figuras, los contratos se garantizan al 100 % (ver tablas 3.1, 3.2 y 3.3). CBM se presenta por tanto como un avance importante para lograr un servicio asegurado con las suficiente garantías, gracias al control que ejerce sobre la generación de paquetes *out*.

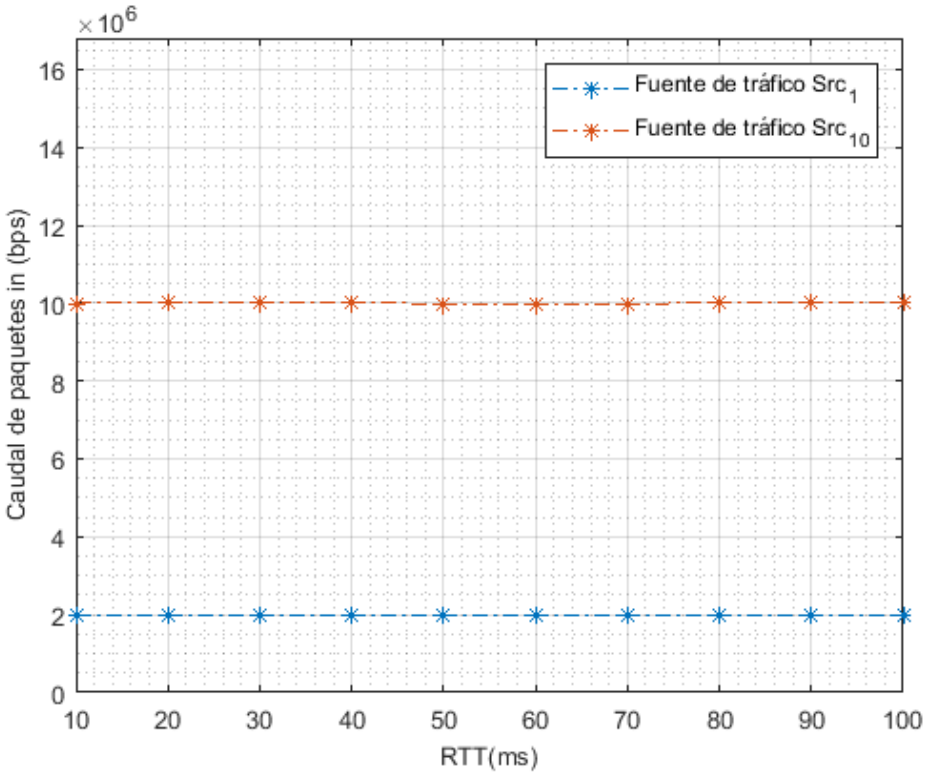

Figura 3.7: PETER - Throughput de paquetes *in* en función del RTT para las fuentes de tráfico TCP S_1 y S_{10} y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps)

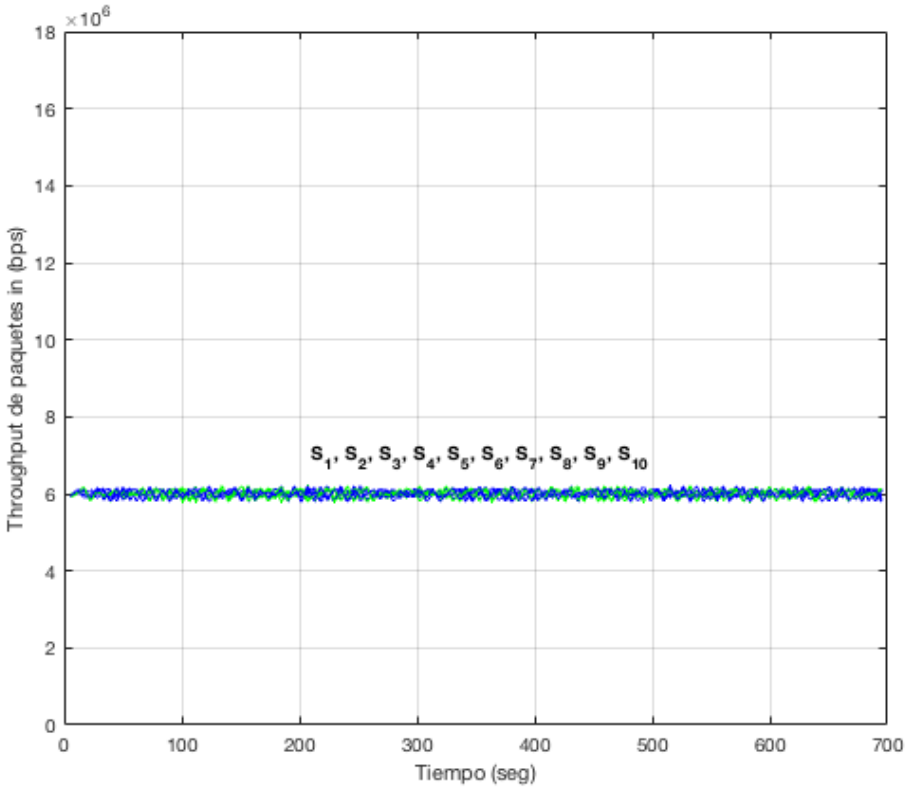

Figura 3.8: CBM - Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 20 ms para todas las conexiones)

Src	RTT *(ms)*	Contrato *(Mbps)*	Caudal *in* *(Mbps)*	Caudal Total *(Mbps)*	Ancho Banda Exceso	
					Mbps	%
1	20	6,00	6,01	9,85	3,84	9,60 %
2	20	6,00	6,00	9,82	3,82	9,55 %
3	20	6,00	6,00	9,82	3,82	9,55 %
4	20	6,00	6,00	9,80	3,80	9,50 %
5	20	6,00	6,00	9,92	3,92	9,80 %
6	20	6,00	6,00	9,83	3,83	9,57 %
7	20	6,00	6,00	9,82	3,82	9,55 %
8	20	6,00	6,00	10,00	4,00	9,99 %
9	20	6,00	6,00	9,85	3,85	9,63 %
10	20	6,00	6,00	9,97	3,97	9,91 %
	Total	60,00	60,01	98,69	38,68	96,65 %

Tabla 3.1: CBM - Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 20 ms para todas las conexiones)

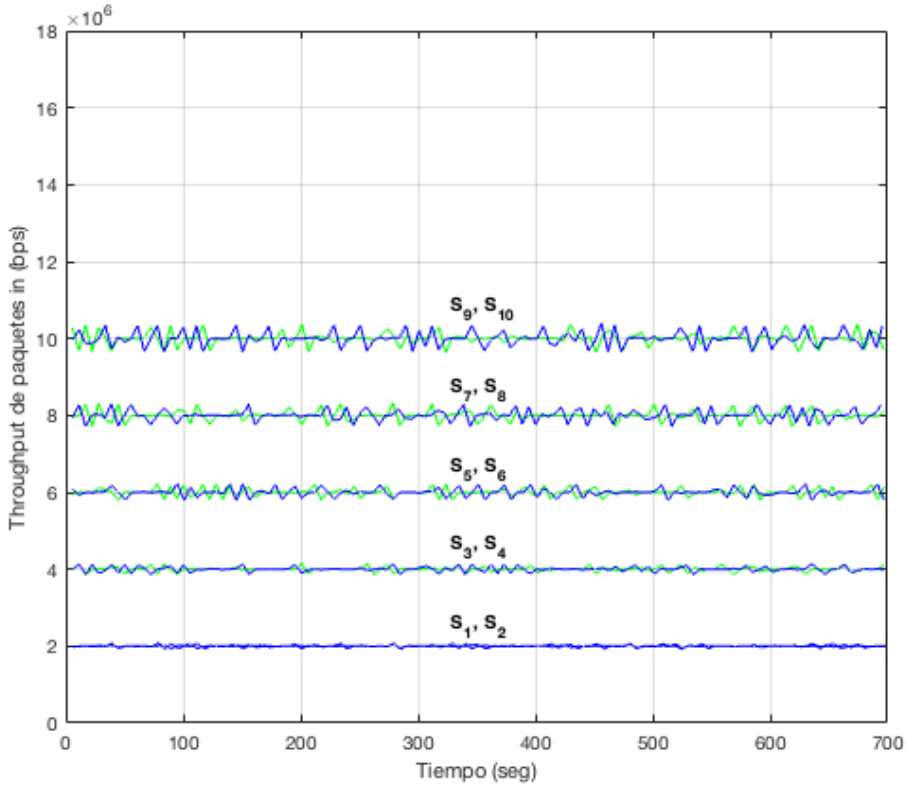

Figura 3.9: CBM - Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 20 ms para todas las conexiones)

Src	RTT *(ms)*	Contrato *(Mbps)*	Caudal *in* *(Mbps)*	Caudal Total *(Mbps)*	Ancho Banda Exceso	
					Mbps	%
1	20	2,00	2,00	5,25	3,25	8,12 %
2	20	2,00	2,00	5,21	3,21	8,02 %
3	20	4,00	4,00	7,39	3,39	8,47 %
4	20	5,00	4,00	7,24	3,24	8,10 %
5	20	6,00	6,00	9,45	3,45	8,62 %
6	20	6,00	6,00	9,38	3,38	8,45 %
7	20	8,00	8,00	11,57	3,57	8,92 %
8	20	9,00	8,00	11,68	3,68	9,19 %
9	20	10,00	10,00	13,48	3,48	8,71 %
10	20	10,00	10,00	13,69	3,69	9,23 %
	Total	60,00	60,00	94,34	34,34	85,83 %

Tabla 3.2: CBM - Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 20 ms para todas las conexiones)

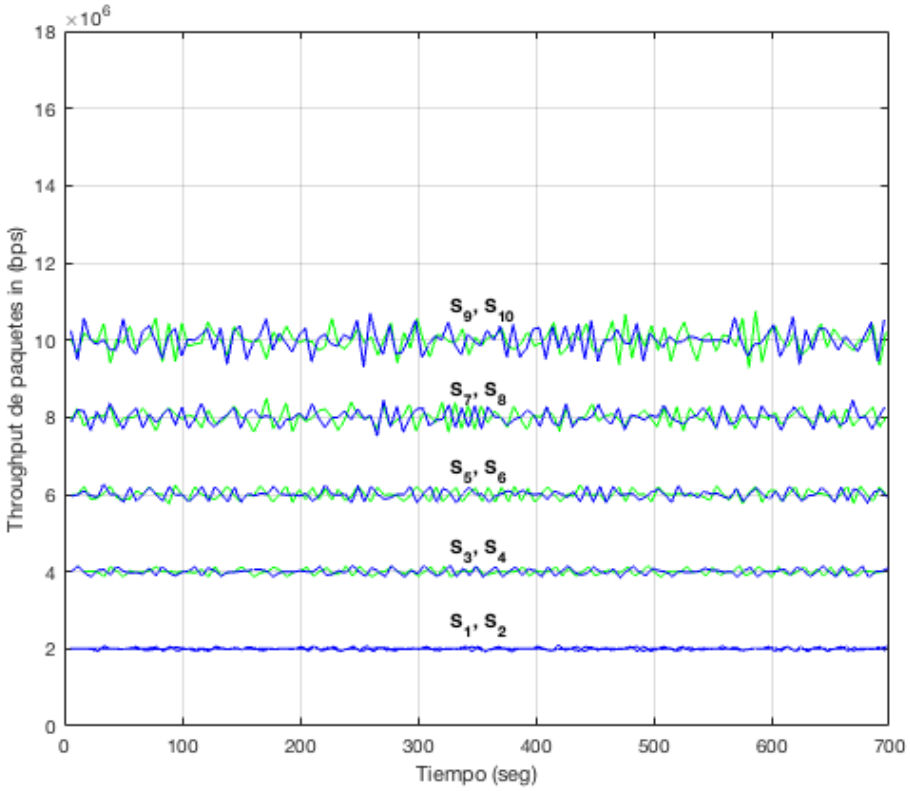

Figura 3.10: CBM - Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms)

Src	RTT *(ms)*	Contrato *(Mbps)*	Caudal *in* *(Mbps)*	Caudal Total *(Mbps)*	Ancho Banda Exceso	
					Mbps	%
1	10	2,00	2,01	4,97	2,97	7,41 %
2	10	2,00	2,00	4,92	2,92	7,30 %
3	20	4,00	4,00	8,47	4,47	11,18 %
4	20	4,00	4,00	8,34	4,34	10,85 %
5	40	6,00	6,01	10,29	4,29	10,71 %
6	40	6,00	6,01	10,42	4,41	11,04 %
7	60	8,00	8,01	11,11	3,10	7,76 %
8	60	8,00	8,01	11,30	3,29	8,23 %
9	80	10,00	10,00	12,36	2,37	5,91 %
10	80	10,00	10,01	12,57	2,56	6,40 %
	Total	60.00	60,05	94,77	34,72	86,78 %

Tabla 3.3: CBM - Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms)

Respecto al reparto del ancho de banda sobrante, también pueden apreciarse las mejoras, en términos de justicia, que introduce este algoritmo con respecto a los acondicionadores de tráfico analizados las secciones 2.2 y 2.3.1. En la Figura 3.11 se muestra los valores de los índices de justicia, para cada uno de los escenario simulados y con una variación de la carga de la red entre el 10 % y 90 %. Incluso en el peor escenario posible, puede apreciarse, que es posible asegurar una distribución justa del ancho de banda en exceso cuando se utiliza el acondicionador de tráfico CBM, alcanzando valores superiores a 0.86 en el peor de los casos. Particularmente el efecto de tener distintos RTT para cada fuente determina la mejora de este acondicionador con respecto a los anteriores. Claramente, la interacción acondicionador de tráfico (TSW o CB)-RIO en un escenario heterogéneo, diferentes RTTs y diferentes CIR, origina una distribución injusta del ancho de banda sobrante. En este contexto, el uso de un descarte probabilístico consigue unos resultados mucho mejores.

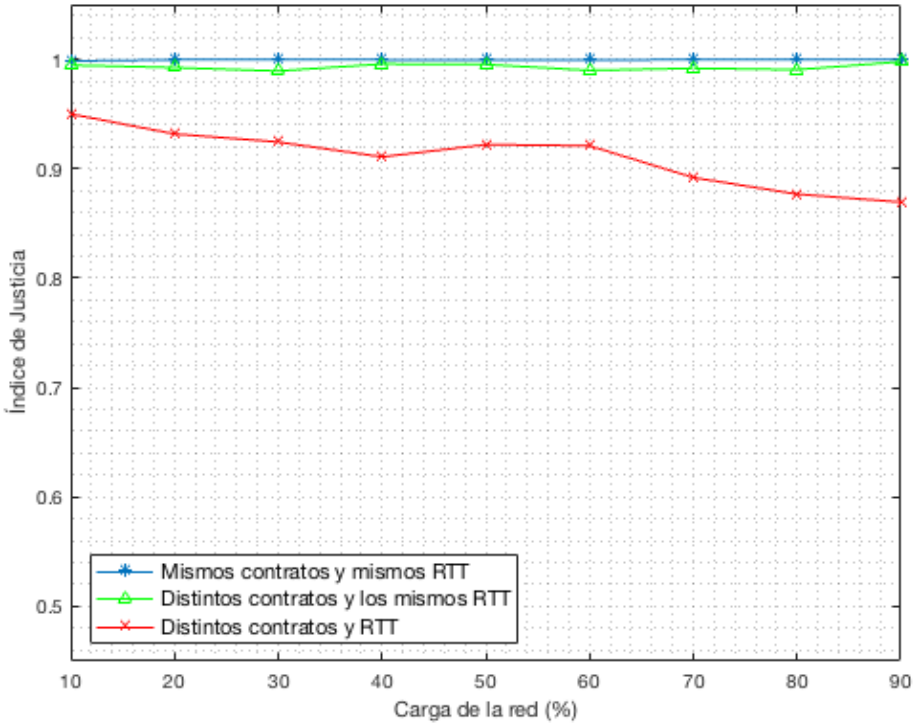

Figura 3.11: CBM - Variación del índice de justicia para los escenarios de simulación A, B y C con una variación de la carga de la red entre el 10 % y 90 %

3.2. Conclusiones

En este capítulo se ha analizado una propuesta orientada a conseguir un reparto equitativo del ancho de banda en exceso. Este acondicionador se denomina CBM (*Counters Based Modified*) y cuya función principal es la de conseguir un reparto justo (equitativo) a través del uso de una función policía que descarta de manera probabilística paquetes que

están calificados como *out*. Empleando este mecanismo, es posible mantener la complejidad en los nodos frontera, utilizando exclusivamente RIO para implementar el PHB. La probabilidad de descarte de un paquete *out* se determina asumiendo que el acondicionador de tráfico conoce la cantidad de ancho de banda sobrante y una aproximación del RTT medio de las conexiones. En la Figura 3.12, en la que se muestra una comparativa del valor del índice de justicia para el escenario más heterogéneo y para distintas cargas de red, se pone de manifiesto la superioridad de este acondicionador, en lo relativo a un reparto equitativo del ancho de banda en exceso, con respecto a los otros acondicionadores de tráfico analizados.

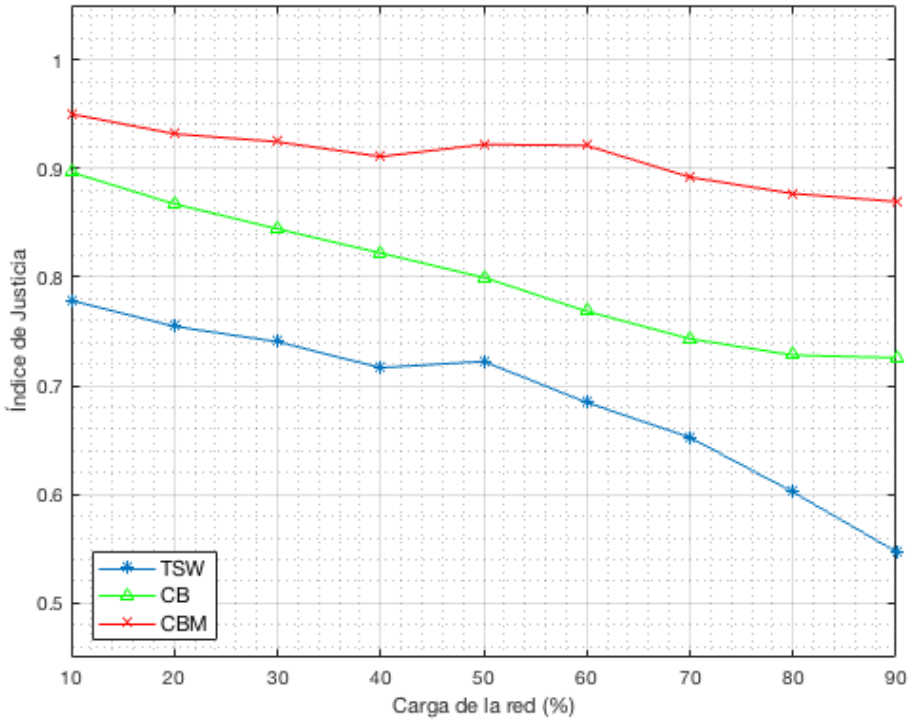

Figura 3.12: Comparativa del índice de justicia para el escenario de simulación C para los acondicionadores de tráfico TSW, CB y CBM y con una variación de la carga de la red entre el 10 % y 90 %

A fin de poder establecer una comparativa entre los distintos algoritmos de acondicionado de tráfico analizados en este capítulo, se han incluido también los índices de justicia alcanzados, obtenidos con la expresión de Jain (ecuación 1.7), para cada uno de los escenario simulados con las propuestas clásicas TSW, TB (*Token Bucket*) y el acondicionador CB (*Counters Based*). Como puede apreciarse, es posible asegurar una distribución justa del ancho de banda en exceso cuando se utiliza el acondicionador de tráfico CBM, con un valor medio de índice de justicia superior a 0.92 para una carga de la red del 60 %, o lo que es lo mismo, como se realiza el reparto del 40 % del ancho restante.

Acondicionador	Escenario A	Escenario B	Escenario C
TSW - RIO	0,994	0,965	0,685
TB - RIO	0,995	0,954	0,697
CB - RIO	0,996	0,971	0,769
CBM	0,997	0,983	0,921

Tabla 3.4: Acondicionador de tráfico CBM. Comparativa de los índices de justicia para una carga de la red del 60 %

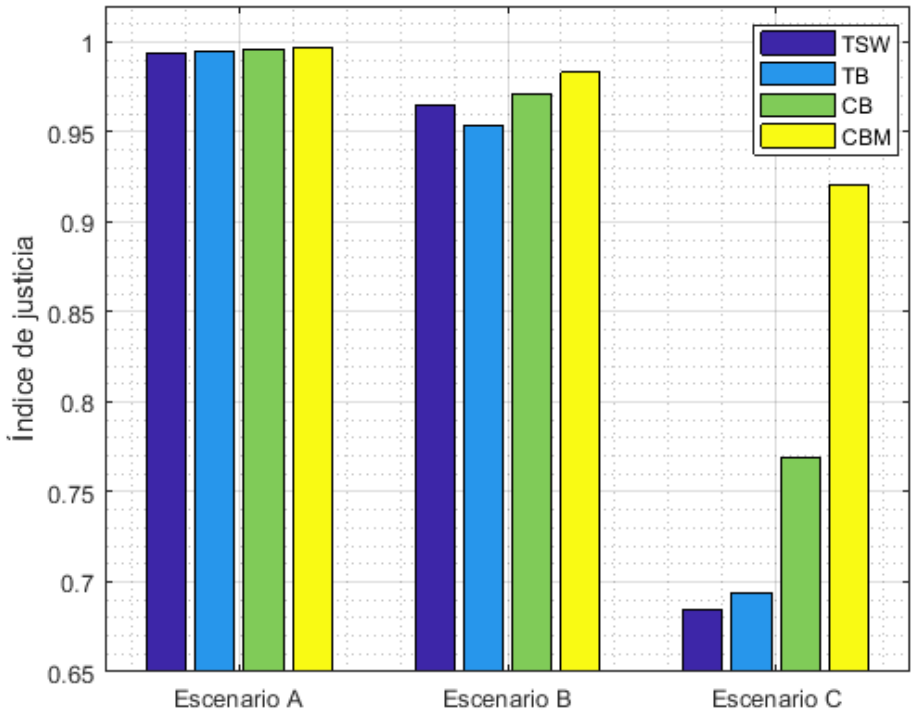

Figura 3.13: Acondicionador de tráfico CBM. Comparativa de los índices de justicia para una carga de la red del 60 %

Capítulo 4

Técnicas para un Reparto Proporcional del Ancho de Banda Excedente. Algoritmo de Acondicionado PETER

En los capítulos 2 y 3 se analizaron diversos acondicionares tráfico en función de la capacidad que tiene cada de uno de ellos para hacer frente a los dos objetivos del AF. Tal y como pudo comprobarse, éstos acondicionadores de tráfico consiguen, en mayor o menor medida, el primer objetivo que es el de asegurar un caudal mínimo o CIR (normalmente el ancho de banda contratado). Ademas, de este estudio se desprende, que el caudal de paquetes *in* en el acondicionador de tráfico *Counters Based*, se garantiza de modo estricto los contratos con apenas variaciones del 1 %. En lo referente al segundo objetivo, permitir el uso de ancho de banda en exceso si la carga de la red es baja, los acondicionadores analizados han considerado únicamente un reparto justo como un reparto equitativo, llegando incluso a obtener buenos resultados en el caso del CBM. Sin embargo, tal y como se comenta en la sección 1.2, existe cierta controversia en la definición de justicia, pues hay quienes consideran, que un reparto justo del ancho de banda en exceso es aquel que se realiza de manera proporcional al contrato de cada una de las fuentes.

En este contexto, el objetivo de este capítulo es el análisis del algoritmo de acondicionado PETER (Proportional Excess Traffic conditionER) presentado en [8] y cuya función es la de ofrecer un reparto proporcional del ancho de banda no contratado dentro del servicio asegurado AF. Es decir, a través de esta propuesta, los usuarios finales obtienen un ancho de banda en exceso proporcional a sus contratos. El acondicionador de tráfico se sitúa junto a la fuente de tráfico (donde se establece el contrato) pero fuera del alcance del usuario final. Esencialmente, este acondicionador marca paquetes con dos niveles de precedencia (*in* o *out*), utilizando el marcador *CB* por ejemplo, aunque podrían utilizarse otros como el TSW o TB, para posteriormente aplicar la función policía PETER que ajusta el throughput de las fuentes de tráfico, adaptándolas a las condiciones de la red mediante descarte de paquetes si fuera necesario.

A fin de evaluar las prestaciones de este acondicionador de tráfico mediante simulación,

será necesario implementar en el núcleo del simulador la función policía PETER.

4.1. Descripción del Algoritmo

Tal y como se comenta en el capítulo el Servicio Asegurado (AF) asegura un trato preferente. Se definen cuatro clases posibles pudiéndose asignar a cada clase una cantidad de recursos en los routers (ancho de banda, espacio en buffers, etc..). Para cada clase se pueden definir hasta tres categorías de descarte de paquetes (probabilidad alta, media y baja). En este sentido, el esquema propuesto en [8] emplea dos clases o niveles de precedencia (*in* y *out*). Además se define como α_{ideal} a la relación mostrada en la expresión 4.1, donde *n* el número de fuentes que componen el agregado, *c* es la capacidad del enlace y *b* la suma de los contratos de todas las fuentes de tráfico que se multiplexan en un mismo nodo frontera, $(c - b)$ del numerador representa el ancho de banda excedente y el denominador representa el ancho de banda total contratado (ver Figura 4.1).

$$\alpha_{ideal} = \frac{capacidad\ \ enlace - \sum_{i=1}^{n} contrato_i}{\sum_{i=1}^{n} contrato_i} = \frac{c-b}{b} \tag{4.1}$$

Nótese que el valor de α_{ideal} no sufre ninguna variación a menos que nuevos usuarios contraten ancho de banda para acceder a la red a través de este nodo frontera, un usuario se dé de baja del servicio o modifique su contrato o se aumente la capacidad de la red.

Por otra parte, en un intervalo de tiempo determinado $[t_1, t_2]$, se define α_m como el ratio paquetes *out* entre paquetes *in* que salen del nodo frontera hacia el interior del dominio.

$$\alpha_{ideal} = \frac{\sum_{t_1}^{t_2} tamaño\ \ paquete\ \ out_i}{\sum_{t_1}^{t_2} tamaño\ \ paquete\ \ in_i} \tag{4.2}$$

Por simplicidad si se supone que todos los paquetes tienen un tamaño similar, la expresión de α_m puede reducirse a:

$$\alpha_{ideal} = \frac{\sum_{t_1}^{t_2} paquetes\ \ out}{\sum_{t_1}^{t_2} paquetes\ \ in} \tag{4.3}$$

De lo que se deduce, que el numerador (suma de paquetes *out*) de la expresión 4.3 representa el consumo del ancho de banda no contratado, mientras que el denominador (suma de paquetes *in*) representa el ancho de banda contratado. Por tanto, si la utilización de la red es cercana al 100 %, los valores α_m y α_{ideal} deben ser similares. Si ambos valores no coinciden significa que no se está consumiendo todo el ancho de banda disponible.

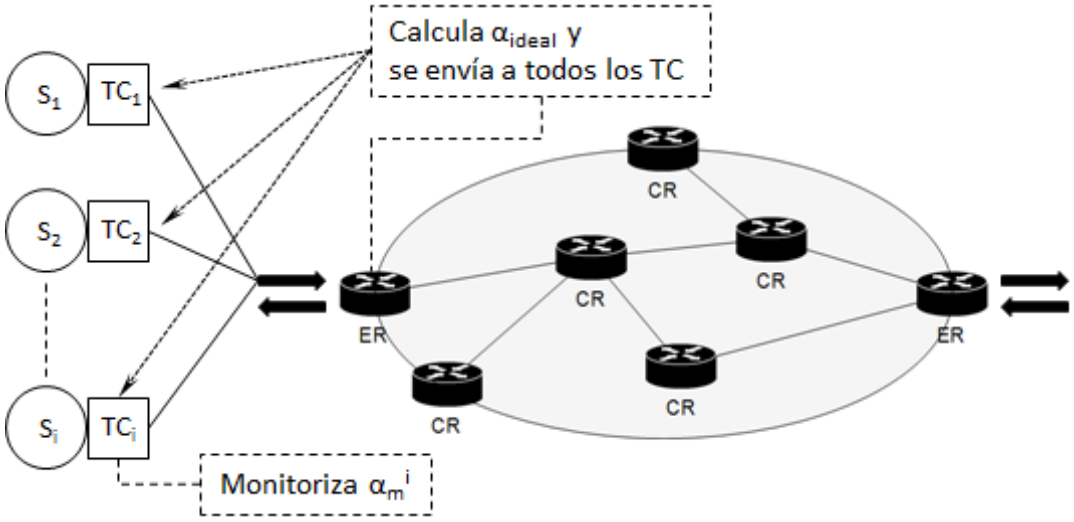

Figura 4.1: Funcionamiento PETER

Por otro lado, si se mide el ratio α_m en cada acondicionador de tráfico*i* (α_m^i), situados junto a las fuentes pero fuera del alcance del usuario final, se podría emplear el valor de α_{ideal} con el objetivo de alcanzar un reparto proporcional del ancho de banda en exceso. El valor de α_{ideal} se envía a todos los acondicionadores de tráfico, a fin de poder compararlo con el valor de α_m^i obtenido por cada uno de ellos, de forma que:

- Si $\alpha_m^i < \alpha_{ideal}$: La fuente no está consumiendo la porción total de ancho de banda que le corresponde (el contratado más el exceso de modo proporcional).
- Si $\alpha_m^i = \alpha_{ideal}$: La fuente está usando exactamente el ancho de banda que le corresponde.
- Si $\alpha_m^i > \alpha_{ideal}$: La fuente está consumiendo más ancho de banda del que debería. Por lo tanto, si se detecta esta situación ésta fuente deberá ser penalizada para que disminuya su throughput. Puesto que estamos trabajando con fuentes TCP Reno, un paquete perdido reduce la velocidad de las fuentes. Así, cuando un acondicionador detecte esta situación, descartará paquetes para conseguir que disminuya la velocidad de la fuente.

Resulta obvio, que para realizar esta tarea es necesario utilizar señalización entre el nodo frontera y los acondicionadores de tráfico. Puesto que el nodo frontera es el más cercano al usuario final, esta señalización no supone un problema dado que se corresponde

con el bucle de abonado (distancias cortas).

En cuanto al descarte de paquetes, éste se puede realizar de distintas formas. Una de ellas sería descartar paquetes con independencia si se trata de paquetes marcados como *in* o como *out*, siempre que se detecte la situación $\alpha_m^i > \alpha_{ideal}$. El problema que existe con esta solución, es que es posible que al eliminar paquetes de tipo *in* pueda afectar a la garantía de contratos. En este caso, la solución por la que se opta es descartar únicamente paquetes tipo *out* cuando se cumpla la condición anterior (ver Figura 4.2).

```
α_m^i = ratio (paquetes out/paquetes in)

if (α_m^i ≤ α_ideal) {
     no se descarta paquete
}
else {
     if paquete in
        no se descarta paquete
     else
        descartar paquete
}
```

Figura 4.2: Pseudo-código del algoritmo PETER

4.2. Análisis de prestaciones

Para realizar el análisis de prestaciones de este acondicionador de tráfico, emplearemos la topología mostrada en la Figura 1.11. En este caso, se dispone de 10 fuentes TCP Reno (S_1, S_2, S_3, $\cdots$, S_{10}) que transmiten a la velocidad del enlace, establecida en 100 Mbps. Todas las fuentes envían tráfico a sus destinos (d_1, d_2, d_3, $\cdots$, d_{10}) a través del nodo frontera E_1, donde se calcula el valor α_{ideal} y se envía a todos los acondicionadores de tráfico. El cuello de botella queda por tanto situado entre los nodos E_1 y E_1.

Por simplicidad, en este capítulo trabajamos con fuentes long-lived y se supondrá que no se produce ningún fallo. La configuración escogida para los routers RIO empleados es (40, 70, 0.02) para los paquetes *in* y (10, 40, 0.2) para los paquetes *out*. El valor del peso de la función promedio de la longitud de la cola (w_q), para ambos casos, se establecerá en 0.002, tal y como se recomienda en [14].

Se contemplarán cinco escenarios de simulación diferentes, todos ellos con una carga de red del 60 % de la capacidad del enlace (suma de contratos igual a 60 Mbps). En este supuesto, se podrá analizar como se distribuye el 40 % del ancho de banda en exceso (no contratado), entre las diferentes fuentes que componen el agregado:

- **Escenario A:** Todas las conexiones tienen el mismo contrato y el mismo RTT. Los valores escogidos en este caso es de 6 Mbps para el caso de los contratos y 20 ms para el RTT para todas las fuentes de tráfico. Este es el escenario más simple y el más utilizado en estudios relacionados con acondicionado de tráfico y Servicios Diferenciados.

- **Escenario B:** Todas las conexiones tienen diferente contrato pero el mismo RTT. Los valores seleccionados en este caso de es 2, 2, 4, 4, 6, 6, 8, 8, 10 y 10 Mbps de S_1 a S_{10} respectivamente. El RTT, al igual que en el escenario anterior, queda fijado en 20 ms. El objetivo en este escenario es el de evaluar la influencia que tiene la diversidad de contratos en las prestaciones del acondicionador de tráfico.

- **Escenario C:** Todas las conexiones tienen diferente contrato y diferente RTT. En particular, las fuentes con menor contrato tendrán los RTT más bajos. Los valores seleccionados son de 2, 2, 4, 4, 6, 6, 8, 8, 10 y 10 Mbps con un RTT de 10, 10, 20, 20, 40, 40, 60, 60, 80 y 80 ms de S_1 a S_{10} respectivamente. En este caso se analizará los efectos de los contratos y RTT de manera simultánea.

- **Escenario D:** Todas las conexiones tienen diferente contrato y diferente RTT. En particular, las fuentes con mayor contrato tendrán los RTT más bajos. Los valores seleccionados para el RTT es de 80, 80, 60, 60, 40, 40, 20, 20, 10 y 10 ms de S_1 a S_{10} respectivamente.

- **Escenario E:** Todas las conexiones tienen el mismo contrato pero distinto RTT. En particular las simulaciones se realizan con unos valores del RTT de 10, 10, 20, 20, 40, 40, 60, 60, 80 y 80 ms de S_1 a S_{10} respectivamente. En este caso se analizará la influencia del RTT.

4.2.1. Tasa de Velocidad de Paquetes *in*

Las simulaciones realizadas con el algoritmo PETER revelan que los caudales de paquetes *in* garantizan de modo estricto los contratos, con variaciones que apenas llegan al 1 %. En la Figura 4.3 se puede apreciar la gran exactitud con la que se aseguran los contratos empleando PETER para una carga de red del 60 %. En esta figura se representa el caudal de paquetes *in* de las fuentes S_1, S_3, S_5, S_7 y S_9 en las condiciones descritas en el escenario C, es decir, con unos valores de los contratos de 2, 2, 4, 4, 6, 6, 8, 8, 10 y 10 Mbps de S_1 a S_{10}. Las simulaciones se realizaron variando el RTT en las distintas simulaciones pero manteniéndolo igual para todas las fuentes.

Según los resultados obtenidos en simulaciones con distinto RTT, se puede deducir que este esquema es capaz de dar las máximas garantías a los usuarios de que alcanzarán sus contratos independientemente del retardo de cada una. Estas garantías de contrato se pueden observar también en las figuras 4.4, 4.5, 4.6, 4.7 y 4.8 donde se representa el caudal de paquetes *in* en función del tiempo de cada una de las fuentes para todos los escenarios de simulación.

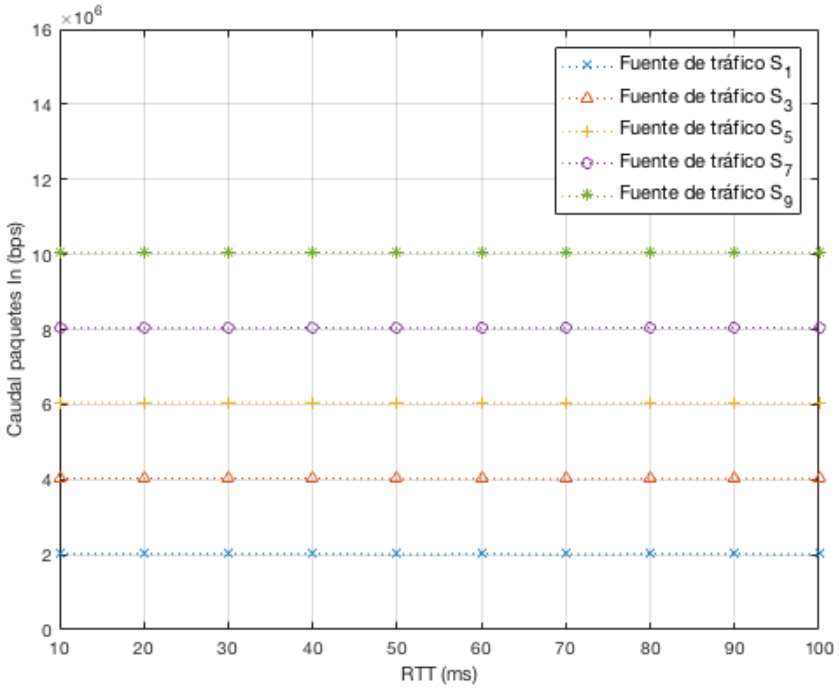

Figura 4.3: PETER - Throughput de paquetes *in* en función del RTT para las fuentes de tráfico TCP S_1, S_3, S_5, S_7 y S_9 y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps)

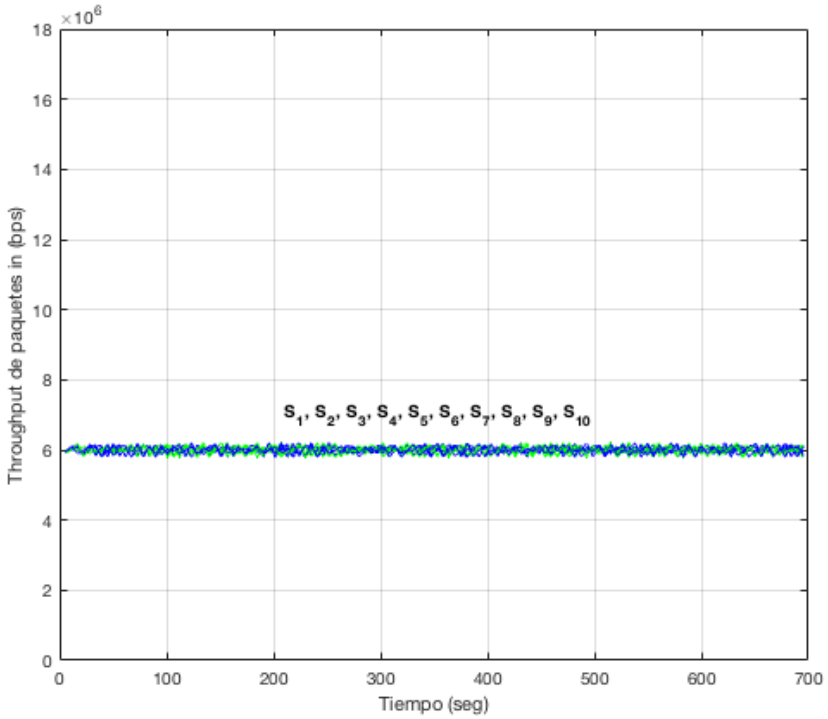

Figura 4.4: PETER - Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 20 ms para todas las conexiones)

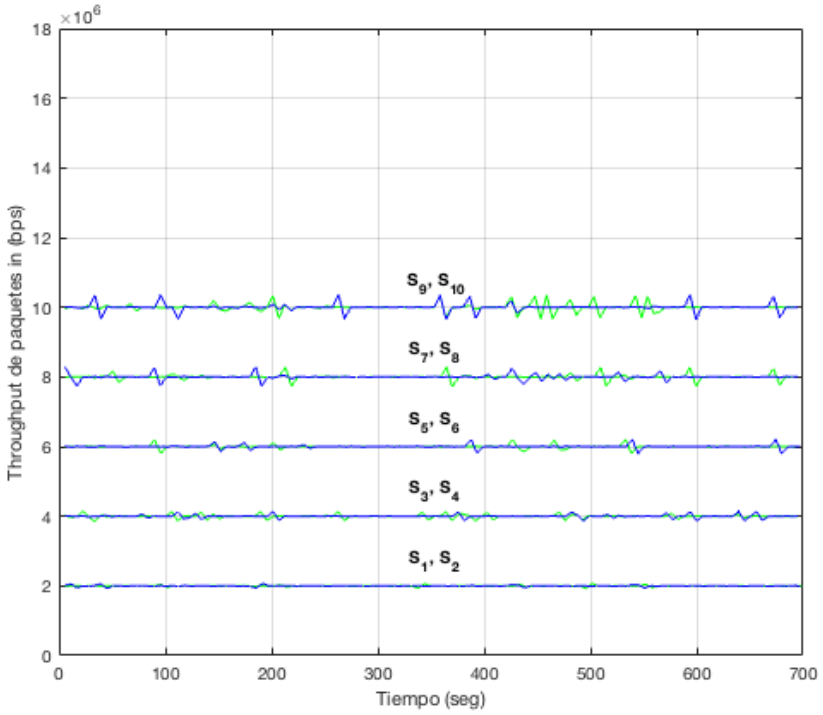

Figura 4.5: PETER - Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 20 ms para todas las conexiones)

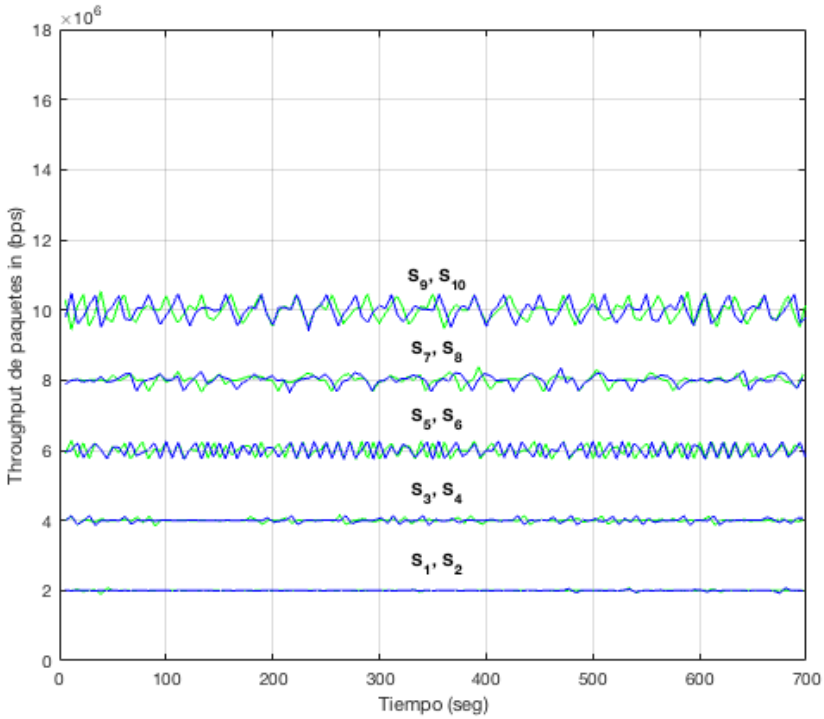

Figura 4.6: PETER - Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms)

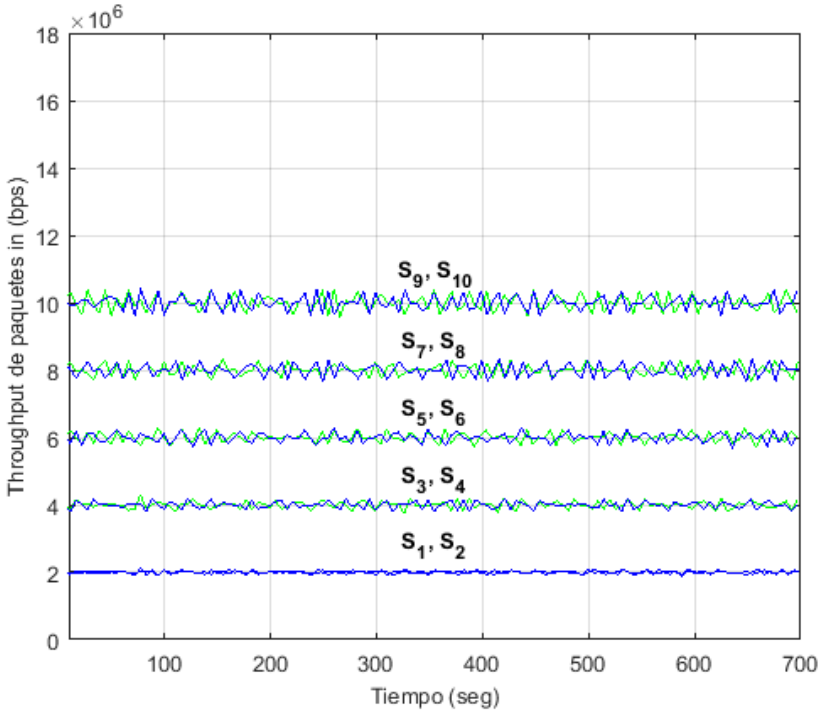

Figura 4.7: PETER - Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 80, 80, 60, 60, 40, 40, 20, 20, 10, 10 ms)

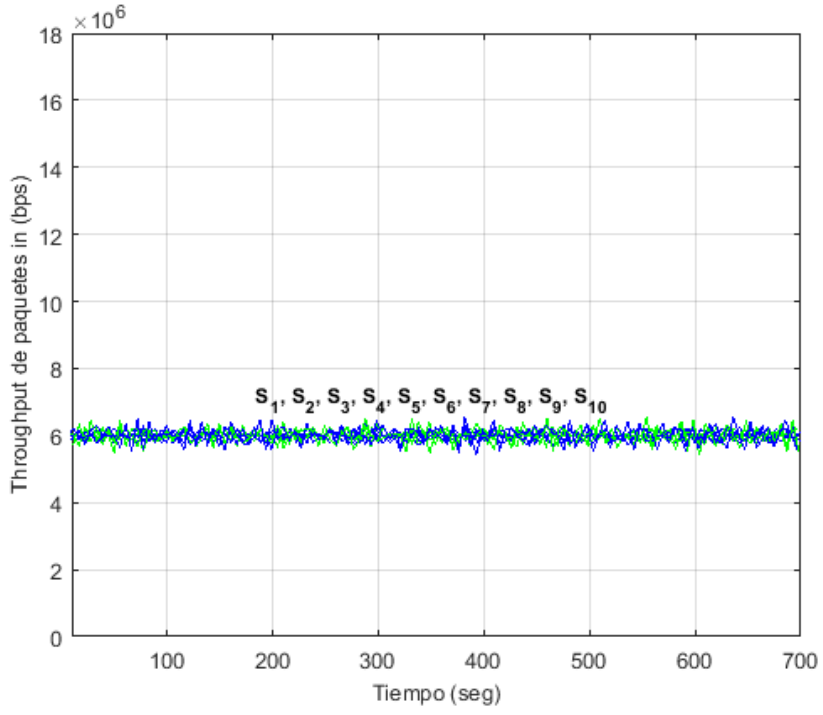

Figura 4.8: PETER - Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms)

4.2.2. Reparto del Ancho de Banda Excedente

En la Tabla 4.1 se recogen los resultados de la simulación: el caudal de paquetes *in*, *goodput* alcanzado o el caudal total, el ancho de banda excedente y porcentaje del mismo obtenido por cada una de las fuentes de tráfico en las condiciones descritas en el escenario A, esto es, 10 fuentes TCP Reno (S_1 a S_{10}) todas con el mismo ancho de banda contratado, para una carga de red del 60 % y empleando PETER. Puesto que todas las fuentes contratan el mismo ancho de banda (6 Mbps), el reparto del ancho de banda en exceso de modo proporcional hace que todas las fuentes obtengan la misma proporción de ancho de banda. El descarte de paquetes *out* en el propio acondicionador de tráfico cuando se detecta la desigualdad $\alpha_m^i > \alpha_{ideal}$, hace a las fuentes TCP disminuir su velocidad en su justa medida.

Src	**RTT** ***(ms)***	**Contrato** ***(Mbps)***	**Caudal *in*** ***(Mbps)***	**Caudal Total** ***(Mbps)***	**Ancho Banda Exceso**	
					Mbps	*%*
1	20	6,00	6,00	9,88	3,88	9,70 %
2	20	6,00	6,00	9,88	3,88	9,69 %
3	20	6,00	6,00	9,87	3,87	9,68 %
4	20	6,00	6,00	9,88	3,88	9,70 %
5	20	6,00	6,00	9,87	3,87	9,66 %
6	20	6,00	6,00	9,88	3,88	9,71 %
7	20	6,00	6,00	9,88	3,88	9,69 %
8	20	6,00	6,01	9,89	3,88	9,71 %
9	20	6,00	6,00	9,88	3,87	9,68 %
10	20	6,00	6,00	9,88	3,88	9,70 %
	Total	60,00	60,03	98,80	38,77	96,93 %

Tabla 4.1: PETER - Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 20 ms para todas las conexiones)

En la Tabla 4.2 se recogen los resultados del escenario de simulación B, es decir, se mantienen 10 fuentes de tráfico TCP Reno pero con diferentes contrato, que para una carga de la red del 60 %, las fuentes de S_1 a S_{10} contratan 2, 2, 4, 4, 6, 6, 8, 8, 10 y 10 Mbps respectivamente, todas ellas con un retardo de 20 ms. En este caso el objetivo es analizar los resultados que se obtienen en el caso en el que se tengan fuentes de tráfico con distintos contratos (diversidad de contratos). A partir de los resultados, se deduce de forma clara que la variación de contratos no afecta a las prestaciones de PETER, pues continúa garantizando con gran exactitud los contratos de los usuarios. Por otra parte, con respecto al reparto del ancho de banda excedente, se puede comprobar que todas las fuentes de tráfico obtienen su correspondiente parte proporcional del ancho de banda en exceso. Por tanto, el descarte de paquetes en el acondicionador de tráfico se utiliza para que las diferentes fuentes se adapten a las condiciones de la red a partir de la información proporcionada por α_{ideal} y α_m^i.

Src	RTT *(ms)*	Contrato *(Mbps)*	Caudal *in* *(Mbps)*	Caudal Total *(Mbps)*	Ancho Banda Exceso	
					Mbps	%
1	20	2,00	2,00	3,30	1,30	3,25 %
2	20	2,00	2,00	3,30	1,30	3,24 %
3	20	4,00	4,01	6,60	2,60	6,49 %
4	20	4,00	4,00	6,59	2,59	6,48 %
5	20	6,00	6,01	9,90	3,90	9,74 %
6	20	6,00	6,01	9,89	3,89	9,72 %
7	20	8,00	8,00	13,20	5,19	12,98 %
8	20	8,00	8,01	13,18	5,18	12,94 %
9	20	10,00	10,01	16,50	6,49	16,22 %
10	20	10,00	10,01	16,49	6,48	16,20 %
	Total	60,00	60,05	98,96	38,91	97,27 %

Tabla 4.2: PETER - Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 20 ms para todas las conexiones)

En la Tabla 4.3 se recogen los resultados de un escenario en el que todas las fuentes de tráfico contratan el mismo ancho de banda (6 Mbps), pero con retardos de 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms de S_1 a S_{10}, con el objetivo de analizar la influencia de tener fuentes con distintos retardos.

Src	RTT *(ms)*	Contrato *(Mbps)*	Caudal *in* *(Mbps)*	Caudal Total *(Mbps)*	Ancho Banda Exceso	
					Mbps	%
1	10	6,00	6,01	9,81	3,79	9,49 %
2	10	6,00	6,00	9,78	3,78	9,45 %
3	20	6,00	6,01	9,81	3,80	9,50 %
4	20	6,00	6,01	9,81	3,80	9,49 %
5	40	6,00	6,01	9,70	3,69	9,24 %
6	40	6,00	6,01	9,75	3,74	9,35 %
7	60	6,00	6,01	9,54	3,53	8,83 %
8	60	6,00	6,00	9,43	3,43	8,58 %
9	80	6,00	6,01	8,99	2,98	7,46 %
10	80	6,00	6,01	7,52	2,50	7,52 %
	Total	60.00	60,08	91,93	31,85	88,90 %

Tabla 4.3: PETER - Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms)

En el capítulo 2 se dedujo la influencia que el RTT tiene sobre el caudal total que es

capaz de alcanzar cada fuente. En particular, suele producirse una tendencia que favorece de las fuentes con menor RTT, a menos que este efecto sea contrarrestado de algún modo. En este sentido, y según los resultados incluidos en la Tabla 4.3, se observa como los contratos se garantizan y se alcanzan unos resultados aceptables en cuanto al reparto del ancho de banda excedente con PETER.

En las tablas 4.4 y 4.5 se recogen los resultados en un ambiente más heterogéneo, y por tanto, más realista. En este caso el escenario consta de diez fuentes TCP Reno con diferentes contratos y variedad en los tiempos de ida y vuelta, a fin de evaluar en este caso, el efecto de tener distintos contratos y retardos en las fuentes de manera simultánea. En particular, los contratos de las fuentes queda fijado en 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps de S_1 a S_{10}. En cuanto a los retardos de las fuentes, en la Tabla 4.4 las fuentes de menor contrato cuentan con los menores retardos (10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms de S_1 a S_{10}), mientras que, en la Tabla 4.5 las fuentes de mayor contrato tienen los menores retardos (80, 80, 60, 60, 40, 40, 20, 20, 10, 10 ms de S_1 a S_{10}).

Al igual que en los casos anteriores, PETER consigue garantizar, de la misma forma que en los casos anteriores, los contratos de un modo riguroso a todas las fuentes de tráfico con variaciones de apenas el 1 %. Por otra parte, queda demostrado también, que el descarte de paquetes en el acondicionador de tráfico se utiliza para que las diferentes fuentes se adapten a las condiciones de la red a partir de la información proporcionada por α_{ideal} y α_m^i, y conseguir de esta forma, un reparto proporcional a los contratos de las distintas fuentes de tráfico, incluso a pesar de la heterogeneidad de los escenarios cuyos resultados se reflejan en las tablas 4.4 y 4.5.

Src	**RTT** *(ms)*	**Contrato** *(Mbps)*	**Caudal** ***in*** *(Mbps)*	**Caudal Total** *(Mbps)*	**Ancho Banda Exceso**	
					Mbps	*%*
1	10	2,00	2,00	3,26	1,26	3,14 %
2	10	2,00	2,00	3,26	1,26	3,15 %
3	20	4,00	4,00	6,53	2,52	6,31 %
4	20	4,00	4,00	6,52	2,52	6,30 %
5	40	6,00	6,00	9,75	3,75	9,39 %
6	40	6,00	6,00	9,75	3,75	9,37 %
7	60	8,00	8,00	12,87	4,86	12,15 %
8	60	8,00	8,00	12,67	4,67	11,68 %
9	80	10,00	10,01	16,08	6,08	15,19 %
10	80	10,00	9,99	16,00	6,01	15,01 %
	Total	60.00	60,01	96,69	36,68	91,70 %

Tabla 4.4: PETER - Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms)

Src	RTT *(ms)*	Contrato *(Mbps)*	Caudal *in* *(Mbps)*	Caudal Total *(Mbps)*	Ancho Banda Exceso	
					Mbps	%
1	80	2,00	2,01	3,23	1,22	3,05 %
2	80	2,00	2,01	3,24	1,24	3,09 %
3	60	4,00	4,00	6,06	2,06	5,15 %
4	60	4,00	4,00	6,10	2,10	5,25 %
5	40	6,00	6,01	9,16	3,15	7,86 %
6	40	6,00	6,01	9,01	3,00	7,50 %
7	20	8,00	8,02	12,58	4,56	11,41 %
8	20	8,00	8,01	12,62	4,60	11,51 %
9	10	10,00	10,01	15,74	5,73	14,32 %
10	10	10,00	10,01	15,88	5,87	14,67 %
	Total	60.00	60,09	93,62	33,53	83,82 %

Tabla 4.5: PETER - Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 80, 80, 60, 60, 40, 40, 20, 20, 10, 10 ms)

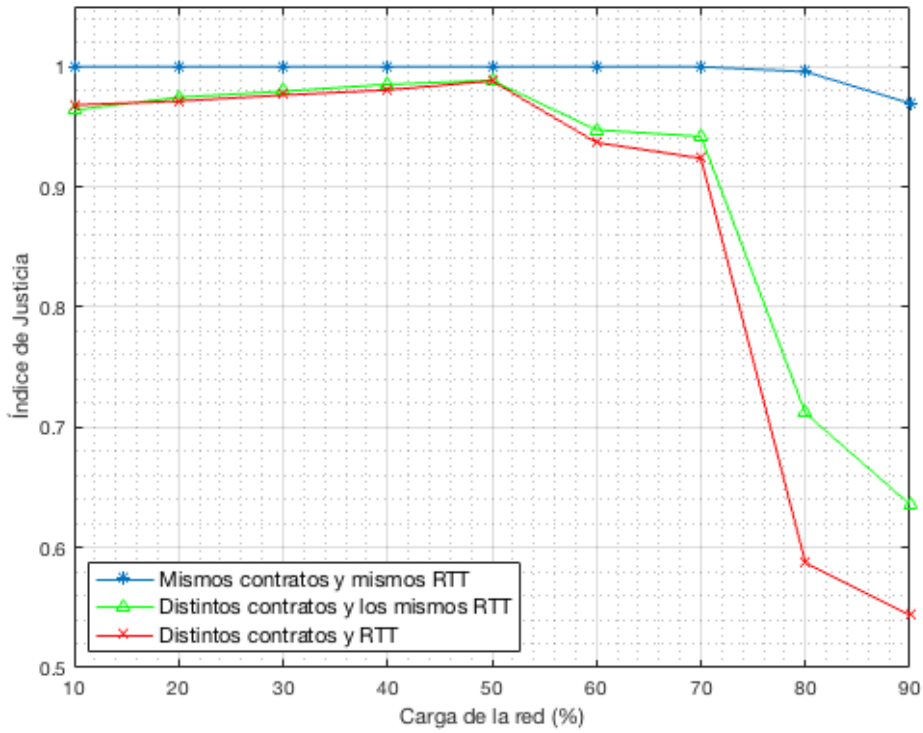

Figura 4.9: Comparativa del índice de justicia para los escenarios de simulación A, B y C para los acondicionador de tráfico PETER con una variación de la carga de la red entre el 10 % y 90 %

En la Figura 4.9 se representa el valor del índice de justicia en los escenarios de simulación A, B y C para un amplio abanico de valores de carga de red, que va desde el

10 % hasta el 90 %, donde se representa en su eje de abcisas el valor de la carga de la red y en su eje de ordenadas el índice de justicia alcanzado. Los resultados de simulación muestran un buen nivel de justicia para redes con una carga entre el 10 % y el 70 %, obteniéndose valores superiores a 0.9 pese a la heterogeneidad del escenario de simulación.

4.3. PETER-DQ (Dual Queue)

El esquema de gestión Dual Queuing Buffer (DQ) aparece con la idea de aplicar una gestión de buffer diferente a la ofrecida por el esquema RIO tradicional. Al igual que en el esquema RIO, la idea subyacente es dar un tratamiento distinto a los paquetes *in* y *out* en el router (ver figura 4.10). Sin embargo, cuando ambos tipos de paquetes comparten una misma cola se produce cierta interferencia entre ambos tipos de paquetes, lo que hace difícil su manejo para proporcionar un reparto justo del ancho de banda excedente.

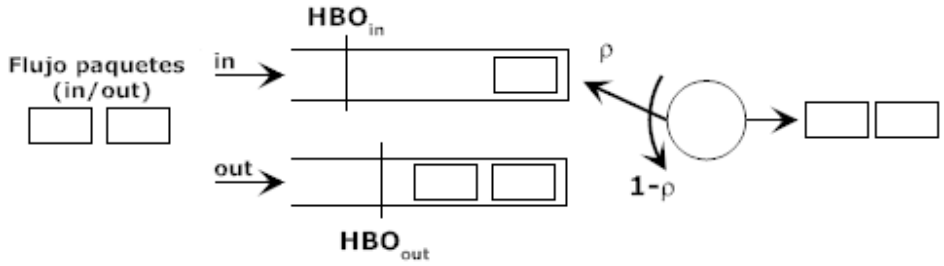

Figura 4.10: Nodo DQ

La gestión DQ consiste en encolar y reenviar los paquetes *in* y *out* en el router en dos colas diferentes. Conjuntamente se emplea un algoritmo de planificación para decidir qué cola se sirve primero. Con el fin de reducir la complejidad, su diseñador utilizó un algoritmo de tipo Round Robin ponderado (*Weighted Round Robin*, WRR), para tal propósito. De manera más concreta, el planificador sirve a la cola que contiene los paquetes *in* con una probabilidad que equivale a la carga de tráfico total contratada ρ, mientras que la cola que almacena los paquetes *out* se sirve con la probabilidad complementaria (1 - ρ).

$$\rho = \frac{\sum_{i=1}^{n} tasa\ \ contratada}{capacidad\ \ enlace} \tag{4.4}$$

Como puede observarse en la figura 4.10, ambas colas emplean un umbral para limitar el número máximo de paquetes que pueden ser almacenados. Estos umbrales reciben el nombre de HBO_{in} y HBO_{out} para las colas que contienen los paquetes *in* y *out* respectivamente, donde HBO es el acrónimo de ocupación máxima del buffer (High Buffer Occupancy). Para realizar las simulaciones los valores de cada uno de los parámetros

involucrados en este acondicionador de tráfico vienen dados por: $HBO_{in} = 60$, $HBO_{out} = 13$, $\rho = 0{,}6$.

4.3.1. Tasa de Velocidad de Paquetes *in*

En primer lugar se analizará la interacción entre el acondicionador de tráfico CB, la gestión de doble cola DQ y la función policía PETER. En la figura 4.11 se representa el caudal de paquetes *in* de las fuentes S_1, S_3, S_5, S_7 y S_9 en las condiciones descritas en el escenario C, es decir, con unos valores de los contratos de 2, 2, 4, 4, 6, 6, 8, 8, 10 y 10 Mbps de S_1 a S_{10}.

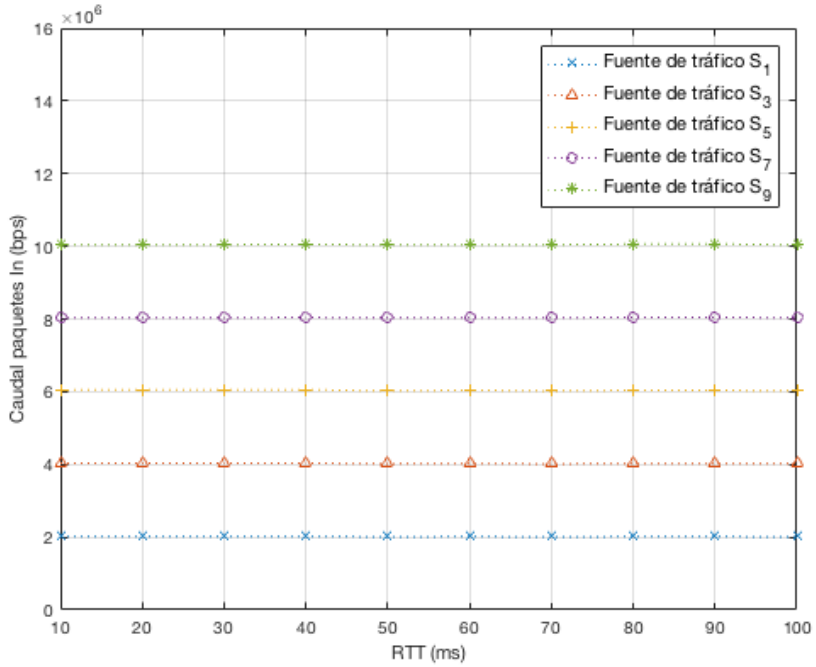

Figura 4.11: PETER DQ - Throughput de paquetes *in* en función del RTT para las fuentes de tráfico TCP S_1, S_3, S_5, S_7 y S_9 y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps)

A partir de estos resultados, obtenidos en simulaciones con distinto RTT, se puede deducir que este esquema es capaz, al igual que el sistema PETER-RIO, de garantizar a los usuarios que alcanzarán sus contratos independientemente del retardo. Estas garantías de contrato se pueden observar también en las figuras 4.12, 4.13, 4.14, 4.15 y 4.16 donde se representa el caudal de paquetes *in* en función del tiempo de cada una de las fuentes para todos los escenarios de simulación.

Es en este punto es donde interviene la elección de un valor u otro para el parámetro HBO_{in}, puesto que éste debe ser lo suficientemente alto para no descartar paquetes *in*, de lo contrario los contratos podrían verse comprometidos.

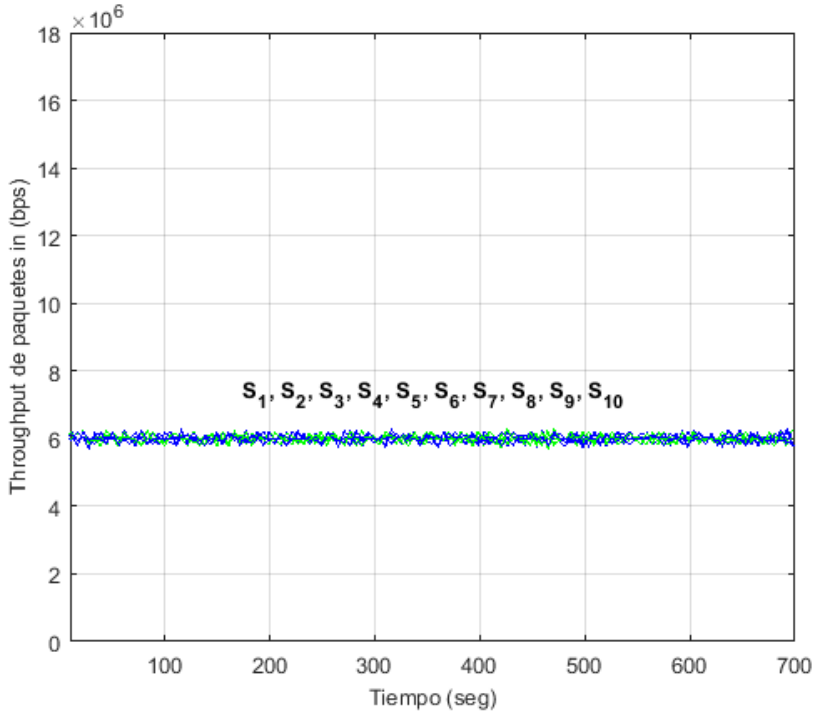

Figura 4.12: PETER DQ - Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 20 ms para todas las conexiones)

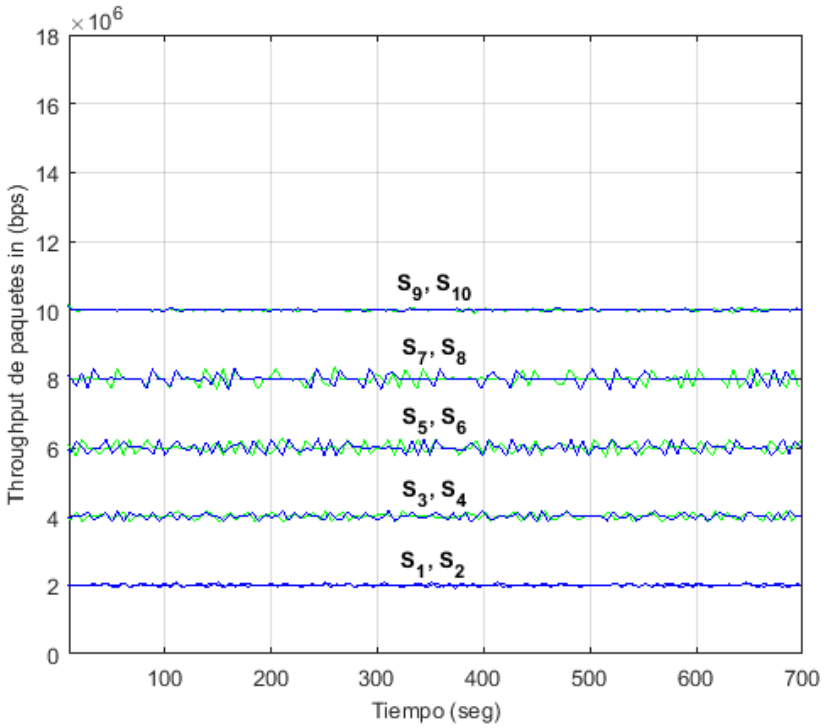

Figura 4.13: PETER DQ - Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Contratos de 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 20 ms para todas las conexiones)

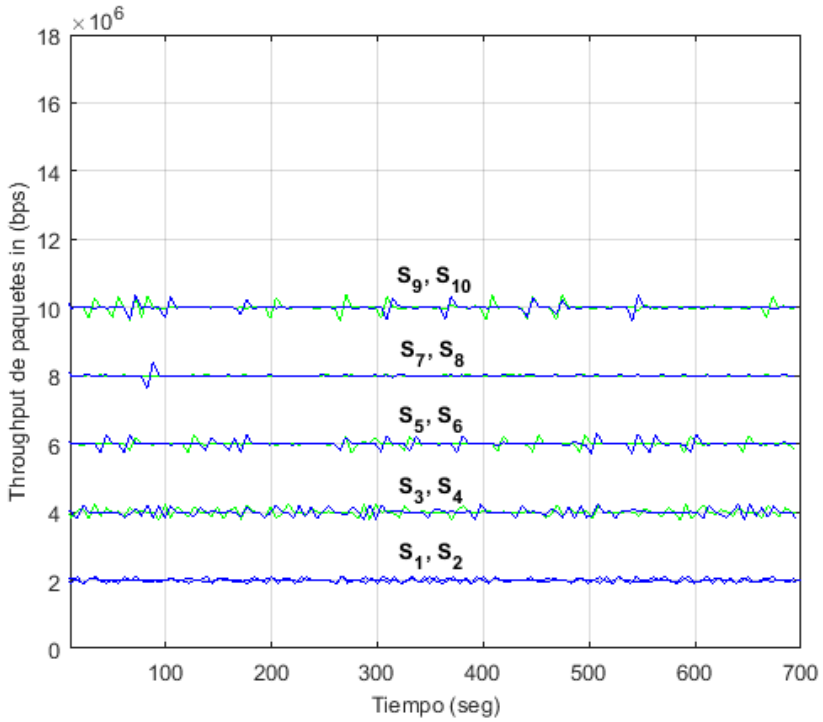

Figura 4.14: PETER DQ - Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Contratos de 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms)

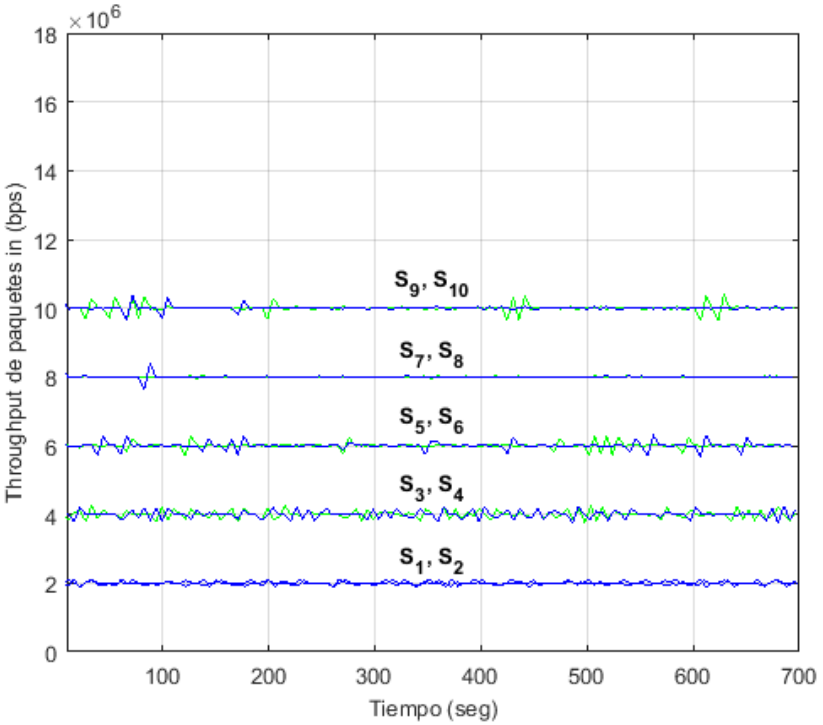

Figura 4.15: PETER DQ - Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Contratos de 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 80, 80, 60, 60, 40, 40, 20, 20, 10, 10 ms)

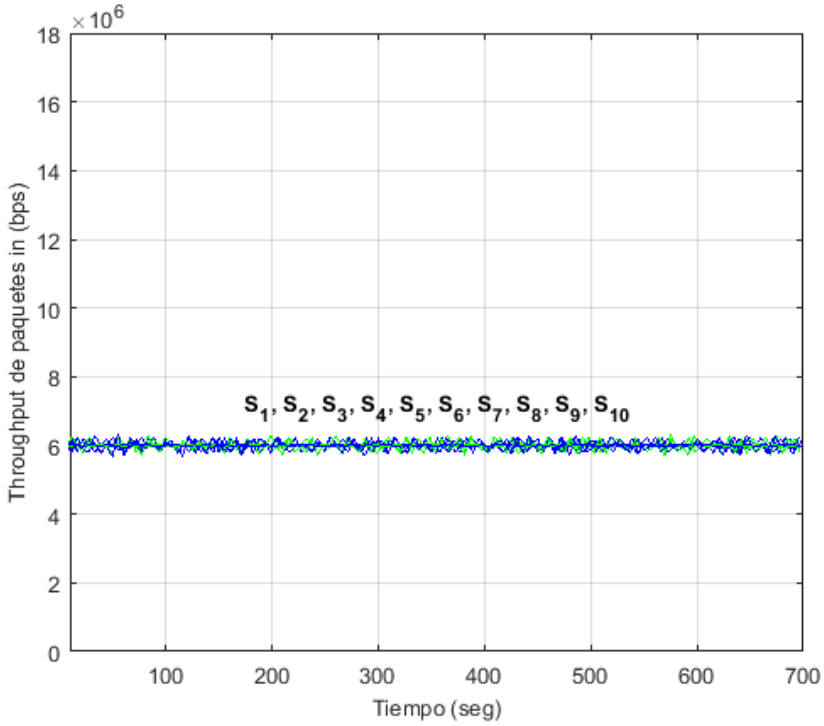

Figura 4.16: PETER DQ - Throughput de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms)

4.3.2. Reparto del Ancho de Banda Excedente

En el apartado 4.3.1 se puede comprobar a través de los resultados obtenidos, que la gestión de doble cola es capaz de cumplir el primer objetivo del servicio AF, esto es, garantizar los contratos de los usuarios. Así mismo, en los resultados recogidos en las tablas 4.6, 4.7, 4.8, 4.9 y 4.10, se puede comprobar que, en cada uno de los escenarios considerados en este capítulo (todos ellos con una carga de la red del 60 %), el caudal de paquetes *in* garantiza de modo exacto el contrato de cada una de las fuentes de tráfico independientemente de su contrato y RTT.

En esta sección, el análisis de prestaciones se centrará en analizar la capacidad que tiene el acondicionador de tráfico junto con la función policía PETER y el esquema de doble cola, para conseguir un reparto proporcional del ancho de banda no contratado.

Para el primer caso de estudio, el más simple de todos (escenario A), en el que todas las fuentes contratan el mismo ancho de banda (6 Mbps) y el retardo es el mismo en todas ellas (20ms), los valores paracen recogidos en la Tabla 4.6, donde se observa que no existen grandes diferencias con respecto al tradicional esquema RIO, por lo que es posible obtener unos valores altos del indice de justicia.

Src	RTT *(ms)*	Contrato *(Mbps)*	Caudal *in* *(Mbps)*	Caudal Total *(Mbps)*	Ancho Banda Exceso	
					Mbps	%
1	20	6,00	6,01	9,89	3,88	9,70 %
2	20	6,00	6,01	9,75	3,74	9,35 %
3	20	6,00	6,01	9,86	3,85	9,62 %
4	20	6,00	6,01	9,75	3,74	9,35 %
5	20	6,00	6,01	9,86	3,85	9,63 %
6	20	6,00	6,01	9,77	3,76	9,40 %
7	20	6,00	6,01	9,79	3,78	9,45 %
8	20	6,00	6,01	9,80	3,79	9,48 %
9	20	6,00	6,01	9,89	3,88	9,69 %
10	20	6,00	6,01	9,79	3,78	9,45 %
	Total	60,00	60,10	98,17	38,07	95,12 %

Tabla 4.6: PETER DQ - Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 20 ms para todas las conexiones)

En la tablas 4.7 y 4.8 se recogen los resultados de los escenarios de simulación donde se analiza la influencia que tiene la diversidad de contratos y tener fuentes con distintos retardos respectivamente. De estos resultados, se deduce que la variabilidad del retardo o del contrato, de manera separada, no influye en las prestaciones de PETER aún cuando utilizamos un esquema de gestión de cola diferente como es DQ.

Src	RTT *(ms)*	Contrato *(Mbps)*	Caudal *in* *(Mbps)*	Caudal Total *(Mbps)*	Ancho Banda Exceso	
					Mbps	%
1	20	2,00	2,00	3,33	1,33	3,33 %
2	20	2,00	2,00	3,33	1,33	3,33 %
3	20	4,00	4,00	6,66	2,65	6,64 %
4	20	4,00	4,01	6,68	2,68	6,69 %
5	20	6,00	6,01	10,02	4,01	10,02 %
6	20	6,00	6,01	10,01	4,00	10,00 %
7	20	8,00	8,01	13,36	5,35	13,38 %
8	20	8,00	8,00	13,34	5,34	13,36 %
9	20	10,00	10,00	16,60	6,60	16,50 %
10	20	10,00	10,01	16,61	6,60	16,49 %
	Total	60,00	60,05	99,95	39,90	99,73 %

Tabla 4.7: PETER DQ - Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 20 ms para todas las conexiones)

Src	RTT *(ms)*	Contrato *(Mbps)*	Caudal *in* *(Mbps)*	Caudal Total *(Mbps)*	Ancho Banda Exceso	
					Mbps	%
1	10	6,00	6,00	10,00	4,00	10,00 %
2	10	6,00	6,01	10,01	4,00	10,00 %
3	20	6,00	6,01	10,02	4,01	10,02 %
4	20	6,00	6,01	10,02	4,01	10,02 %
5	40	6,00	6,02	10,01	4,00	9,99 %
6	40	6,00	6,02	10,01	4,00	10,00 %
7	60	6,00	6,01	9,84	3,83	9,56 %
8	60	6,00	6,01	9,84	3,83	9,58 %
9	80	6,00	6,01	8,76	2,75	6,87 %
10	80	6,00	6,01	8,67	2,66	6,64 %
	Total	60.00	60,11	97,19	37,08	92,68 %

Tabla 4.8: PETER DQ - Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms)

En las tablas 4.9 y 4.10 se recogen los resultados en un ambiente más realista. En particular, se analiza el efecto de tener distintos contratos y retardos en las fuentes de manera simultánea. Los contratos de las fuentes queda fijado en 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps de S_1 a S_{10}. En cuanto a los retardos de las fuentes, en la Tabla 4.4 las fuentes de menor contrato cuentan con los menores retardos, mientras que en la Tabla 4.5 las fuentes de mayor contrato tienen los menores retardos.

Src	RTT *(ms)*	Contrato *(Mbps)*	Caudal *in* *(Mbps)*	Caudal Total *(Mbps)*	Ancho Banda Exceso	
					Mbps	%
1	10	2,00	2,00	3,34	1,33	3,33 %
2	10	2,00	2,00	3,34	1,33	3,33 %
3	20	4,00	4,00	6,67	2,66	6,66 %
4	20	4,00	4,01	6,67	2,66	6,66 %
5	40	6,00	6,01	10,02	4,01	10,02 %
6	40	6,00	6,01	10,01	4,00	10,00 %
7	60	8,00	8,02	13,18	5,16	12,90 %
8	60	8,00	8,02	13,30	5,28	13,20 %
9	80	10,00	10,03	13,71	3,68	9,21 %
10	80	10,00	10,03	13,98	3,96	9,89 %
	Total	60.00	60,13	94,21	34,08	85,19 %

Tabla 4.9: PETER DQ - Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms)

Src	RTT *(ms)*	Contrato *(Mbps)*	Caudal *in* *(Mbps)*	Caudal Total *(Mbps)*	Ancho Banda Exceso	
					Mbps	%
1	80	2,00	2,00	3,34	1,34	3,34 %
2	80	2,00	2,00	3,33	1,33	3,32 %
3	60	4,00	4,01	6,68	2,67	6,68 %
4	60	4,00	4,01	6,67	2,66	6,64 %
5	40	6,00	6,02	10,01	3,99	9,99 %
6	40	6,00	6,02	10,00	3,99	9,97 %
7	20	8,00	8,02	13,27	5,25	13,12 %
8	20	8,00	8,02	13,33	5,31	13,26 %
9	10	10,00	10,02	16,69	6,67	16,67 %
10	10	10,00	10,02	16,68	6,66	16,66 %
	Total	60.00	60,14	99,99	39,85	99,66 %

Tabla 4.10: PETER DQ - Resultados para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 80, 80, 60, 60, 40, 40, 20, 20, 10, 10 ms)

Según los resultados anteriores, y los reflejados en la figura 4.17, se deduce la capacidad que PETER, empleando un esquema de doble cola como es DQ, tiene para conseguir un reparto del ancho de banda en exceso proporcional a los contratos incluso en escenarios más heterogéneos.

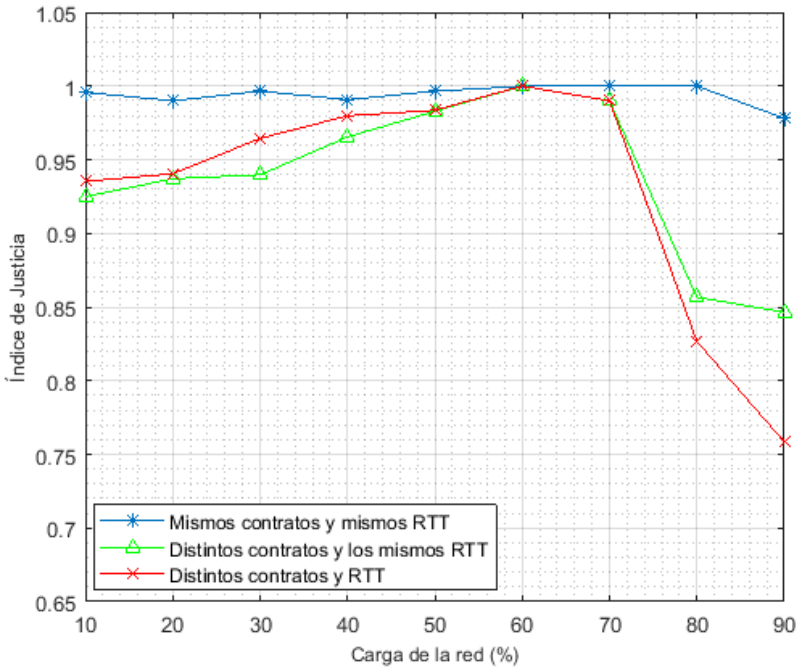

Figura 4.17: Comparativa del índice de justicia para los escenarios de simulación A, B y C para los acondicionador de tráfico PETER con una variación de la carga de la red entre el 10 % y 90 %

4.4. Incremento del Número de Fuentes

El objetivo de esta sección es analizar el efecto que tiene el incrementar el número de fuentes sobre las prestaciones recibidas por el usuario final. Para ello se considera una topología que consta de 20 fuentes TCP Reno, en donde el contrato de las 8 primeras fuentes (S_1 a S_8) queda fijado en 1,50 Mbps. El resto de fuentes (S_9 a S_{20}) contratan un ancho de banda de 4 Mbps cada una, de forma que la carga total de la red sea el 60 %.

En el primer escenario a considerar en este caso, toda las fuentes tienen un retardo de 20 ms para todas las fuentes. En la Tabla 4.11, se recoge el caudal de paquetes *in*, el caudal total y el porcentaje del ancho de banda excedente de cada una de las fuentes para cada uno de los esquemas analizados en este capítulo (PETER-RIO y PETER-DQ).

Src	RTT (*ms*)	Contrato (*Mbps*)	Caudal PETER-RIO			Caudal PETER-DQ		
			in (*Mbps*)	Total (*Mbps*)	Exceso (%)	*in* (*Mbps*)	Total (*Mbps*)	Exceso (%)
1	20	1,50	1,50	2,49	2,46 %	1,50	2,43	2,32 %
2	20	1,50	1,50	2,49	2,46 %	1,50	2,41	2,28 %
3	20	1,50	1,50	2,49	2,47 %	1,50	2,43	2,31 %
4	20	1,50	1,50	2,49	2,47 %	1,50	2,41	2,26 %
5	20	1,50	1,50	2,49	2,47 %	1,50	2,42	2,29 %
6	20	1,50	1,50	2,49	2,46 %	1,50	2,41	2,26 %
7	20	1,50	1,50	2,49	2,47 %	1,50	2,42	2,29 %
8	20	1,50	1,50	2,49	2,47 %	1,50	2,42	2,29 %
9	20	4,00	4,00	6,63	6,57 %	4,00	6,47	6,15 %
10	20	4,00	4,00	6,63	6,56 %	4,00	6,43	6,06 %
11	20	4,00	4,01	6,63	6,56 %	4,00	6,41	6,01 %
12	20	4,00	4,00	6,63	6,57 %	4,01	6,46	6,13 %
13	20	4,00	4,00	6,63	6,56 %	4,00	6,31	5,76 %
14	20	4,00	4,01	6,64	6,58 %	4,00	6,40	5,99 %
15	20	4,00	4,01	6,63	6,57 %	4,01	6,43	6,05 %
16	20	4,00	4,01	6,63	6,56 %	4,01	6,38	5,95 %
17	20	4,00	4,00	6,63	6,56 %	4,00	6,26	5,64 %
18	20	4,00	4,01	6,63	6,55 %	4,01	6,27	5,67 %
19	20	4,00	4,00	6,62	6,55 %	4,00	6,44	6,09 %
20	20	4,00	4,01	6,62	6,55 %	4,01	6,43	6,05 %
	Total	60,00	60,08	99,46	98,46 %	60,07	96,02	89,86 %

Tabla 4.11: PETER-RIO y PETER-DQ: Throughput total y de paquetes *in* para 20 fuentes de tráfico TCP y una carga de la red del 60 % (RTT es 20 ms para todas las conexiones)

Los valores recogidos en la tabla 4.11 reflejan que ambos esquemas presentan buenas

prestaciones a la hora de garantizar los contratos incluso cuando se incrementa el número de fuentes. Lo mismo ocurre con respecto al ancho de banda excedente, cuyo reparto se realiza de modo justo.

En la tablas 4.12 y 4.13 se recogen los resultados obtenidos para dos situaciones más complejas. En particular, en el primer caso el retardo que fijado en 10, 10, 10, 10, 20, 20, 20, 20, 40, 40, 40, 40, 60, 60, 60, 60, 80, 80, 80 y 80 ms de S_1 a S_{20}, mientras que en el segundo, el retardo es de 80, 80, 80, 80, 60, 60, 60, 60, 40, 40, 40, 40, 20, 20, 20, 20, 10, 10, 10 y 10 de S_1 a S_{20}. A partir de estos resultados y los anteriores, es posible deducir que con independencia del esquema que se utilice (RIO o DQ) en el nodo frontera de la red (u otros nodos), las prestaciones de PETER se mantienen en todos los aspectos. Esto resulta esencial de cara a obtener un sistema escalable cuando se utiliza este tipo de algoritmos.

Src	**RTT** (***ms***)	**Contrato** (***Mbps***)	**Caudal PETER-RIO**			**Caudal PETER-DQ**		
			in (***Mbps***)	**Total** (***Mbps***)	**Exceso** (**%**)	***in*** (***Mbps***)	**Total** (***Mbps***)	**Exceso** (**%**)
1	10	1,50	1,50	2,42	2,30 %	1,50	2,42	2,29 %
2	10	1,50	1,50	2,41	2,28 %	1,50	2,43	2,31 %
3	10	1,50	1,50	2,42	2,29 %	1,50	2,42	2,30 %
4	10	1,50	1,50	2,42	2,29 %	1,50	2,42	2,30 %
5	20	1,50	1,50	2,42	2,30 %	1,50	2,43	2,32 %
6	20	1,50	1,50	2,42	2,29 %	1,50	2,42	2,30 %
7	20	1,50	1,50	2,42	2,28 %	1,50	2,42	2,29 %
8	20	1,50	1,50	2,43	2,31 %	1,50	2,42	2,30 %
9	40	4,00	4,00	6,49	6,21 %	4,00	6,54	6,35 %
10	40	4,00	4,01	6,47	6,16 %	4,00	6,54	6,33 %
11	40	4,00	4,00	6,48	6,20 %	4,01	6,57	6,40 %
12	40	4,00	4,00	6,48	6,18 %	4,00	6,53	6,33 %
13	60	4,00	4,00	6,32	5,80 %	4,01	6,20	5,48 %
14	60	4,00	4,00	6,29	5,71 %	4,01	6,13	5,29 %
15	60	4,00	4,00	6,26	5,64 %	4,01	6,06	5,13 %
16	60	4,00	4,00	6,31	5,75 %	4,01	6,17	5,40 %
17	80	4,00	4,01	5,73	4,32 %	4,00	5,75	4,37 %
18	80	4,00	4,01	5,71	4,26 %	4,00	5,59	3,97 %
19	80	4,00	4,00	5,74	4,35 %	4,01	5,61	4,00 %
20	80	4,00	4,01	5,62	4,05 %	4,00	5,65	4,11 %
	Total	60,00	60,07	93,26	82,99 %	60,09	92,71	81,57 %

Tabla 4.12: PETER-RIO y PETER-DQ: Throughput total y de paquetes *in* para 20 fuentes de tráfico TCP y una carga de la red del 60 % (RTT es 10, 10, 10, 10, 20, 20, 20, 20, 40, 40, 40, 40, 60, 60, 60, 60, 80, 80, 80, 80 ms de S_1 a S_{20})

Src	RTT (*ms*)	Contrato (*Mbps*)	Caudal PETER-RIO			Caudal PETER-DQ		
			in (*Mbps*)	Total (*Mbps*)	Exceso (%)	*in* (*Mbps*)	Total (*Mbps*)	Exceso (%)
1	80	1,50	1,50	2,47	2,41 %	1,50	2,35	2,12 %
2	80	1,50	1,50	2,46	2,40 %	1,50	2,35	2,12 %
3	80	1,50	1,50	2,46	2,40 %	1,50	2,35	2,13 %
4	80	1,50	1,50	2,47	2,41 %	1,50	2,37	2,18 %
5	60	1,50	1,50	2,47	2,41 %	1,50	2,36	2,15 %
6	60	1,50	1,50	2,47	2,40 %	1,50	2,37	2,17 %
7	60	1,50	1,50	2,46	2,40 %	1,50	2,36	2,15 %
8	60	1,50	1,50	2,46	2,39 %	1,50	2,37	2,17 %
9	40	4,00	4,00	6,57	6,41 %	4,00	6,16	5,38 %
10	40	4,00	4,00	6,56	6,40 %	4,01	6,06	5,12 %
11	40	4,00	4,01	6,57	6,41 %	4,01	5,90	4,74 %
12	40	4,00	4,01	6,55	6,35 %	4,00	6,12	5,29 %
13	20	4,00	4,01	6,57	6,41 %	4,00	6,34	5,84 %
14	20	4,00	4,00	6,57	6,41 %	4,00	6,30	5,74 %
15	20	4,00	4,01	6,57	6,40 %	4,01	6,34	5,85 %
16	20	4,00	4,00	6,57	6,42 %	4,00	6,35	5,86 %
17	10	4,00	4,01	6,55	6,37 %	4,01	6,34	5,83 %
18	10	4,00	4,00	6,56	6,40 %	4,00	6,35	5,87 %
19	10	4,00	4,00	6,55	6,37 %	4,00	6,35	5,86 %
20	10	4,00	4,01	6,57	6,42 %	4,00	6,32	5,78 %
	Total	60,00	60,07	98,47	85,98 %	60,07	93,82	84,36 %

Tabla 4.13: PETER-RIO y PETER-DQ: Throughput total y de paquetes *in* para 20 fuentes de tráfico TCP y una carga de la red del 60 % (RTT es 80, 80, 80, 80, 60, 60, 60, 60,, 40, 40, 40, 40, 20, 20, 20, 20 10, 10, 10, 10 ms de S_1 a S_{20})

4.5. Conclusiones

En este capítulo se han analizado las prestaciones de la propuesta de acondicionador de tráfico presentado en [8], para el servicio AF PHB. En la sección 4.1 se desarrolla el funcionamiento detallado de la función policía PETER, cuya función principal es la de lograr un reparto proporcional, al contrato de las fuentes, del ancho de banda excedente. Para ello, deben marcarse previamente los paquetes, utilizando en este caso dos niveles de precedencia posibles (*in* / *out*), para posteriormente aplicar la función policía PETER en el acondicionador de tráfico. Esta función se encarga de descartar los paquetes marcados como *out* cuando el acondicionador detecte que la fuente está consumiendo una porción total de ancho de banda (el contratado más el exceso de modo proporcional) superior a la que le corresponde. Para ello el acondicionador calcula el ratio α_m^i y comprueba que éste no

excede el valor del ratio α_{ideal} calculado en el nodo que multiplexa las fuentes de tráfico.

Por otra parte, en la sección 4.2.1 se comprueba que el acondicionador cubre el primer objetivo del Servicio Asegurado (AF), esto es, asegurar un caudal mínimo o CIR (normalmente el ancho de banda contratado). En este sentido, de este estudio se desprende, que el caudal de paquetes *in* en el acondicionador de tráfico garantiza de modo estricto los contratos con apenas variaciones del 1 %.

Finalmente, en la sección 4.2.2 se analizan las prestaciones del acondicionador con respecto al segundo objetivo del servicio AF, es decir, el ancho de banda sobrante debe repartirse de modo justo, entendiéndose en este caso por justicia un reparto proporcional a los contratos de cada fuente. A partir de los resultados obtenidos, se demuestra que el descarte de paquetes en el acondicionador de tráfico se utiliza para que las diferentes fuentes se adapten a las condiciones de la red a partir de la información proporcionada por α_{ideal} y α_m^i, y conseguir de esta forma, un reparto proporcional a los contratos de las distintas fuentes de tráfico, incluso a pesar de la heterogeneidad de los escenarios (diversidad de contratos y retardos).

Por otra parte, se analiza la interación entre la función policía PETER y un mecanismo de gestión de buffer diferente a la ofrecida por el esquema RIO tradicional. La idea subyacente es eliminar la interferencia que se produce entre paquetes *in* y *out* por el hecho de compartir una misma cola. En este sentido, el esquema de gestión escogido es el denominado Dual Queuing Buffer (DQ). A partir de la comparación entre PETER-RIO y PETER-DQ, se concluye que es posible obtener resultados similares empleado un esquema de gestión de colas más simple como es DQ, donde solamente es necesario especificar el número máximo de paquetes que se aceptan en cada cola, puesto que dentro de cada una de ellas se sigue un régimen FIFO.

Finalmente y con el propósito de evaluar la escalabilidad de este acondicionador (ya se utilizando un esquema RIO o DQ), se realizan simulaciones en las que se incrementa el número de fuentes de tráfico presentes en la red, y en las que se observa que las prestaciones de PETER se mantienen en todos los aspectos.

Capítulo 5

Conclusiones

Con el aumento del volumen del tráfico ofrecido y la diversidad de servicios con distintos requerimientos, ha surgido la necesidad de soportar una cierta calidad de servicio (QoS). Una de las propuestas del *IETF*, para dotar a las redes IP de QoS, es la arquitectura de servicios diferenciados (*Diffserv*), cuyo objetivo es crear un esquema simple que proporcione un rango de niveles de QoS trasladando la complejidad al borde de la red, intentando que los mecanismos empleados en el interior de la misma sean lo más sencillos posible. De esta forma, se consigue una solución caracterizada por su gran nivel de escalabilidad y sencillez de implementación.

La implementación de DiffServ se basa en el uso del byte DSCP (DiffServ Code Point) de la cabecera IP [21]. En los nodos frontera o en la propia fuente de tráfico, los paquetes se marcarán, clasificarán y acondicionarán antes de entrar en la red con el fin de recibir un tratamiento particular en los nodos que atraviesen a lo largo de su camino. Este tratamiento que reciben los paquetes en los nodos interiores se conoce como Per-Hop Behavior (PHB) [23]. Entre las PHB estandarizadas por el IETF, se ha considerado interesante realizar un estudio sobre Assured Forwarding (AF) PHB. Los objetivos del servicio AF son asegurar un caudal (throughput) mínimo a cada fuente, que normalmente es la tasa contratada, también denominada CIR (Committed Information Rate), y además, permitir a las fuentes consumir más ancho de banda del contratado si la carga de la red es baja. Llegados a este punto, conviene resaltar, que el reparto del ancho de banda en exceso entre las diferentes fuentes se ha de realizar de modo justo, encontrándose dos definiciones distintas para el término justicia. En la primera definición, se asume que el reparto del ancho de banda sobrante se ha de realizar de modo equitativo, mientras que en la segunda, se determina que reparto justo es aquel que es proporcional al contrato de cada una de las fuentes de tráfico. En este trabajo, y al igual que en la mayoría de estudios realizados sobre la materia, se considerará como reparto justo empleando la primera definición, dejando para un futuro trabajo la evaluación de una función policía que consiga un reparto de ancho de banda excedente de forma proporcional a los contratos de cada fuente de tráfico.

Por una parte, la introducción de RIO (RED (Random Early Detection) In y Out) [30] supuso un paso importante en el desarrollo de DiffServ. Este mecanismo que se usa para

implementar el AF PHB, utiliza solamente dos niveles de precedencia de descarte dentro de cada clase AF. Los paquetes que se consideran dentro del perfil de tráfico de una fuente se marcarán como *in* y los que están fuera del perfil como *out*. Una vez un paquete queda marcado, el agregado de tráfico llega al router donde se aplica RIO. RIO es la combinación de dos algoritmos RED [19] con diferentes curvas de probabilidad de descarte, de tal forma que los paquetes *out* tienen una probabilidad de descarte más alta. RIO utiliza una única cola FIFO (First In First Out) para servir ambos tipos de paquetes. La probabilidad de descartar un paquete *out* depende del número total de paquetes de la cola, mientras que la probabilidad de eliminar un paquete *in* depende exclusivamente de la ocupación de la cola con paquetes *in*.

Por otra parte, uno de los elementos esenciales en la arquitectura de Servicios Diferenciados es el acondicionador de tráfico. Como ha podido comprobarse, resulta evidente la relación que existe el trabajo que desempeña y la calidad de servicio percibida por el usuario final. En este trabajo se ha analizado mediante simulación las prestaciones de diversas propuestas de acondicionadores de tráfico. Para llevar a cabo el análisis de prestaciones se empleó el simulador de redes *ns-2*, cuyas características mas reseñables son: la ventajas de ser un simulador de software libre, que permite la inclusión de nuevos módulos que se adapten a las necesidades del escenario a simular; su programación está orientada a objetos, lo que flexibiliza y simplifica la tarea de programación, tanto de su código fuente como de su interfaz y la disponibilidad de un amplio soporte para simulación de calidad de servicio en redes IP.

El análisis de prestaciones se centra en determinar la capacidad que tiene cada uno de ellos para cubrir los dos objetivos del servicio AF: asegurar un caudal mínimo a cada fuente de tráfico, que normalmente es la tasa contratada CIR y además, permitir a las fuentes consumir más ancho de banda del contratado si la carga de la red es baja. En este sentido, las distintas fuentes que componen el agregado debe repartise de la manera más justa posible el ancho de banda en exceso, existiendo en este caso, cierta discrepancia sobre qué se entiende por justicia. Por una parte, algunos autores sostienen que un reparto justo del ancho de banda en exceso, es aquel que se realiza de modo equitativo entre las diferentes fuentes que componen el agregado. Por otra parte, otros autores defienden que un reparto justo es aquel que se realiza de modo proporcional a los contratos de cada fuente. En este estudio, se analizará cuál de ellos es más adecuado en función de los dos enfoques mencionados.

En las tablas 5.1, 5.2 y 5.3 se recogen los resultados de las simulaciones para los escenarios A, B y C para los distintos acondicionadores de tráfico analizados en este documento y con una carga de la red del 60 %. A fin de poder establecer una comparativa, en estas tablas se recoge el caudal de paquetes *in*, el caudal total y el porcentaje del ancho de banda en exceso que consigue cada una de las fuentes.

En la primera parte del trabajo, se analizan las prestaciones de los denominados acondicionadores de tráfico clásicos: *Time Sliding Window* [31] y el acondicionador de

tráfico basado en contadores (*Counters Based*) [8]. Por una parte, tal y como se refleja en la Tabla 5.3, el algoritmo TSW no es capaz de asegurar los contratos con el caudal de paquetes *in*. Lo que puede convertirse en un problema en topologías más complejas. Por otra parte, de este estudio se desprende, que con el caudal de paquetes *in* en el acondicionador de tráfico *Counters Based*, se garantiza de modo estricto los contratos con apenas variaciones del 1 %. Esta característica y su sencillez de configuración, no dispone de parámetros configurables, fue lo que lo hizo adecuado en [8] para el desarrollo de nuevas técnicas de acondicionado de tráfico que fuesen capaces de repartir de modo más equitativo el ancho de banda excedente, puesto que ninguno de ellos es capaz de realizar un reparto equitativo del ancho de banda excedente.

En este contexto, una propuesta en [8] orientada a conseguir un reparto justo, es el acondicionador denominado CBM (*Counters Based Modified*). Este acondicionador de tráfico consigue un reparto equitativo a través del uso de una función policía que descarta de manera probabilística paquetes que están calificados como *out*. Empleando este mecanismo, es posible mantener la complejidad en los nodos frontera, utilizando exclusivamente RIO para implementar el PHB. La probabilidad de descarte de un paquete *out* se determina asumiendo que el acondicionador de tráfico conoce la cantidad de ancho de banda sobrante y una aproximación del RTT medio de las conexiones. Como puede apreciarse, es posible asegurar una distribución justa del ancho de banda en exceso cuando se utiliza el acondicionador de tráfico CBM. Particularmente, la interacción acondicionador de tráfico (TSW, LB o CB)-RIO en un escenario heterogéneo, diferentes RTTs y diferentes CIR, origina una distribución injusta del ancho de banda sobrante. Por esta razón, el uso de un descarte probabilístico consigue unos resultados mucho mejores.

Los acondicionadores analizados en el capítulo 2 y en el capítulo 3: TSW, TB, CB y CBM, han considerado únicamente un reparto justo como un reparto equitativo, llegando a buenos resultados en los estudios realizados al respecto a este último. No obstante, como ya se ha mencionado anteriormente, existe otro enfoque en lo relativo al concepto de justicia. En la última parte de este documento, se analiza la función policía denominada PETER, presentada en [8], con el objetivo de ofrecer un reparto proporcional del ancho de banda en exceso dentro del servicio asegurado AF. En esencia, la función PETER ajusta el throughput de las fuentes de tráfico, adaptándolas a las condiciones de la red mediante descarte de paquetes marcados como *out* si fuera necesario, para evitar de esta forma que una fuente determinada consuma un mayor ancho de banda del que le corresponde de manera proporcional. Para realizar esta tarea es necesario utilizar señalización entre el nodo frontera y los acondicionadores de tráfico. Sin embargo, como el nodo frontera es el más cercano al usuario final, esto no supone un problema para llevar a cabo esta señalización, dado que se corresponde con el bucle de abonado (distancias cortas). De este estudio se desprende, por una parte, que el caudal de paquetes *in* en el acondicionador de tráfico garantiza de modo estricto los contratos. Por otra parte, los resultados de simulación muestran un buen nivel de justicia para redes con una carga entre el 10 % y el 70 %, obteniéndose valores superiores a 0.9 pese a la heterogeneidad del escenario de simulación.

Src	RTT (*ms*)	Contrato (*Mbps*)	Caudal TSW			Caudal CB			Caudal CBM			Caudal PETER		
			in (*Mbps*)	Total (*Mbps*)	Exceso (%)	*in* (*Mbps*)	Total (*Mbps*)	Exceso (%)	*in* (*Mbps*)	Total (*Mbps*)	Exceso (%)	*in* (*Mbps*)	Total (*Mbps*)	Exceso (%)
1	20	6,00	6,07	9,89	9,53 %	6,00	9,88	9,70 %	6,01	9,85	9,60 %	6,00	9,88	9,70 %
2	20	6,00	6,08	9,92	9,58 %	6,00	9,81	9,53 %	6,00	9,82	9,55 %	6,00	9,88	9,69 %
3	20	6,00	6,11	9,82	9,29 %	6,00	9,87	9,67 %	6,00	9,82	9,55 %	6,00	9,87	9,68 %
4	20	6,00	6,07	9,85	9,45 %	6,00	9,70	9,24 %	6,00	9,80	9,49 %	6,00	9,88	9,69 %
5	20	6,00	6,11	9,90	9,46 %	6,00	9,84	9,61 %	6,00	9,92	9,81 %	6,00	9,87	9,68 %
6	20	6,00	6,09	9,90	9,53 %	6,00	9,90	9,73 %	6,00	9,83	9,56 %	6,00	9,88	9,69 %
7	20	6,00	6,09	9,87	9,46 %	6,00	9,85	9,62 %	6,00	9,82	9,55 %	6,00	9,88	9,69 %
8	20	6,00	6,09	9,86	9,43 %	6,00	9,89	9,72 %	6,00	10,00	9,99 %	6,01	9,89	9,71 %
9	20	6,00	6,08	9,79	9,26 %	6,00	9,82	9,54 %	6,00	9,85	9,63 %	6,00	9,88	9,69 %
10	20	6,00	6,07	10,01	9,85 %	6,00	9,85	9,63 %	6,00	9,97	9,92 %	6,00	9,88	9,70 %
	Total	60,00	60,87	98,80	94,84 %	60,01	98,41	95,95 %	60,03	98,69	96,65 %	60,03	98,80	96,92 %

Tabla 5.1: Throughput total y de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (El contrato es 6 Mbps para todas las conexiones. RTT es 20 ms para todas las conexiones)

Src	RTT (*ms*)	Contrato (*Mbps*)	Caudal TSW			Caudal CB			Caudal CBM			Caudal PETER		
			in (*Mbps*)	Total (*Mbps*)	Exceso (%)	*in* (*Mbps*)	Total (*Mbps*)	Exceso (%)	*in* (*Mbps*)	Total (*Mbps*)	Exceso (%)	*in* (*Mbps*)	Total (*Mbps*)	Exceso (%)
1	20	2,00	2,09	5,96	9,67 %	2,00	6,25	10,63 %	2,00	5,25	8,12 %	2,00	3,30	3,25 %
2	20	2,00	2,08	5,95	9,67 %	2,00	6,34	10,85 %	2,00	5,21	8,02 %	2,00	3,30	3,24 %
3	20	4,00	4,10	8,14	10,10 %	4,00	8,03	10,07 %	4,00	7,39	8,47 %	4,01	6,60	6,49 %
4	20	4,00	4,14	7,97	9,57 %	4,00	7,92	9,80 %	4,00	7,24	8,10 %	4,00	6,59	6,48 %
5	20	6,00	6,08	9,96	9,71 %	6,00	9,74	9,34 %	6,00	9,45	8,62 %	6,01	9,90	9,74 %
6	20	6,00	6,08	9,93	9,61 %	6,01	9,85	9,60 %	6,00	9,38	8,45 %	6,01	9,89	9,72 %
7	20	8,00	8,15	11,88	9,32 %	8,00	11,52	8,81 %	8,00	11,57	8,92 %	8,00	13,20	12,98 %
8	20	8,00	8,17	11,83	9,15 %	8,00	11,52	8,80 %	8,00	11,68	9,19 %	8,01	13,18	12,94 %
9	20	10,00	10,05	13,60	8,89 %	10,00	13,42	8,54 %	10,00	13,48	8,71 %	10,01	16,50	16,22 %
10	20	10,00	10,07	13,54	8,68 %	10,00	13,37	8,42 %	10,00	13,69	9,23 %	10,01	16,49	16,20 %
	Total	60,00	61,01	98,76	94,37 %	60,02	97,95	94,84 %	60,01	94,34	85,83 %	60,05	98,96	97,27 %

Tabla 5.2: Throughput total y de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 20 ms para todas las conexiones)

Src	RTT (*ms*)	Contrato (*Mbps*)	Caudal TSW			Caudal CB			Caudal CBM			Caudal PETER		
			in (*Mbps*)	Total (*Mbps*)	Exceso (%)	*in* (*Mbps*)	Total (*Mbps*)	Exceso (%)	*in* (*Mbps*)	Total (*Mbps*)	Exceso (%)	*in* (*Mbps*)	Total (*Mbps*)	Exceso (%)
1	10	2,00	1,86	8,38	16,31 %	2,00	9,26	18,15 %	2,00	4,97	7,43 %	2,00	3,26	3,14 %
2	10	2,00	1,88	8,13	15,63 %	2,00	9,49	18,73 %	2,00	4,92	7,30 %	2,00	3,26	3,15 %
3	20	4,00	4,35	8,95	11,51 %	4,00	9,45	13,62 %	4,00	8,47	11,18 %	4,00	6,53	6,31 %
4	20	4,00	4,36	8,88	11,30 %	4,01	9,35	13,36 %	4,00	8,34	10,85 %	4,00	6,52	6,30 %
5	40	6,00	6,38	8,69	5,79 %	6,00	8,55	6,37 %	6,00	10,29	10,73 %	6,00	9,75	9,39 %
6	40	6,00	6,31	8,64	5,83 %	6,00	8,61	6,51 %	6,00	10,42	11,05 %	6,00	9,75	9,37 %
7	60	8,00	8,25	9,63	3,44 %	8,01	9,40	3,48 %	8,00	11,11	7,77 %	8,00	12,87	12,15 %
8	60	8,00	8,18	9,48	3,26 %	8,00	9,38	3,45 %	8,00	11,30	8,24 %	8,00	12,67	11,68 %
9	80	10,00	12,15	12,83	1,71 %	9,99	10,63	1,61 %	10,00	12,36	5,92 %	10,01	16,08	15,19 %
10	80	10,00	12,07	12,75	1,69 %	9,99	10,76	1,90 %	10,00	12,57	6,41 %	9,99	16,00	15,01 %
	Total	60,00	65,79	96,37	76,46 %	60,01	94,88	87,19 %	60,01	94,77	86,88 %	60,01	96,69	91,70 %

Tabla 5.3: Throughput total y de paquetes *in* para 10 fuentes de tráfico TCP y una carga de la red del 60 % (Los contratos son 2, 2, 4, 4, 6, 6, 10, 10, 12, 12 Mbps. RTT es 10, 10, 20, 20, 40, 40, 60, 60, 80, 80 ms)

Apéndice A

Herramienta de Simulación. Simulador de Redes *ns-2*

El simulador de redes Network Simulator o *ns* [2] (también conocido popularmente como *ns-2* en referencia a la versión 2) es un simulador de eventos discretos y orientado a objetos que tiene sus inicios como una variante del simulador REAL Network Simulator en 1989 y ha evolucionado substancialmente durante los últimos años. A partir de 1995 su desarrollo lo llevaba acabo la agencia de proyectos de investigación avanzada para la defensa de Estados Unidos (DARPA) a través del proyecto VINT (Virtual InterNetwork Testbed), en el que intervienen Xerox PARC, UCB y USC/ISI. Posteriormente ha sido mantenido por diversos proyectos de investigación de la Universidad del Sur de California, en la actualidad el proyecto CONSER. En Julio de 2006 se inició el desarrollo de la versión 3 del simulador [35].

Llegados a este punto, conviene señalar que *ns-3* no es una mejora del simulador *ns-2*, si no que se trata de una versión nueva del mismo. Entre las principales causas que motivaron a su desarrollo, fueron los altos requerimientos de memoria, velocidad del simulador, falta de herramientas gráficas y de análisis de resultados. Sin embargo, algunos módulos no se encuentran disponibles aún, o no se encuentran lo suficientemente testeados (por ejemplo el módulo de Servicios Diferenciados, a diferencia que *ns-2* que dispone de un módulo incluído en su núcleo). Por esta razón, *ns-2* fue y en algunos aspectos sigue siéndolo un estándar de facto entre los simuladores libres tanto en el mundo académico como en la industria, por lo que dispone de una gran cantidad de código y documentación disponible en comparación con el resto de simuladores.

Los motivos que han llevado a la elección de esta herramienta y no otra entre la amplia variedad de simuladores disponibles se detallaron en la sección 1.3. El objetivo de este capítulo es la de ofrecer una breve introducción sobre la estructura interna del simulador, a fin de comprender mejor su funcionamiento y que será de especial utilidad a la hora de abordar algunos aspectos como puedan ser la modificación del código fuente del mismo. En este capítulo se prestará además, especial atención en el módulo de servicios diferenciados que proporciona esta herramienta, así como la modificaciones necesarias para conseguir el

funcionamiento deseado.

A.1. Estructura Interna del Simulador

El núcleo de *ns-2* está desarrollado en C++, sin embargo, proporciona una interfaz de simulación a través de OTcl, un dialecto orientado a objetos de Tcl (Object Tool command language, lenguaje orientado a objetos de tipo intérprete). La razón principal de emplear dos lenguajes diferentes se debe principalmente a que *ns-2* lleva a cabo dos tareas diferentes, cada una de ellas con distintos requisitos. En primer lugar, debe implementar todos los detalles de los protocolos de comunicaciones, para ello requiere un lenguaje que implemente algoritmos de manera eficiente y con relativa facilidad, además de ser capaz de manipular bytes, cabeceras de paquetes y en definitiva grandes volúmenes de datos, por ello el simulador emplea para este propósito el lenguaje C++, pues la velocidad de procesamiento es un punto clave. Por otra parte, se requiere que el lenguaje aporte cierta flexibilidad y rapidez a la hora de realizar configuraciones y variaciones en los parámetros para poder explorar los resultados de las simulaciones, sin necesidad de tener que recompilar el código (lenguaje interpretado), por esta razón, *ns-2* emplea el lenguaje OTcl.

Tal y como se señala en [7] el simulador debe soportar una jerarquía de clases en C++, o jerarquía compilada, y una jerarquía de clases equivalente en OTCL, o jerarquía interpretada (ver figura A.1). Ambas jerarquías se encuentran estrechamente relacionadas entre sí, de modo que, desde la perspectiva del usuario, hay una correspondencia uno a uno entre una clase de la jerarquía interpretada y otra de la compilada. La raíz de esta jerarquía es una clase que se llama TclObject. Cuando el usuario crea un objeto simulador desde el intérprete, cosa con la que generalmente se comienza un script, este objeto es instanciado dentro del intérprete y se crea una estrecha relación con otro objeto idéntico, pero dentro de la jerarquía compilada. La jerarquía de clases interpretada se establece automáticamente a través de métodos definidos dentro de la clase llamada TclClass. Además, es posible crear otras jerarquías personalizadas en los scripts, escritas en OTcl, y que no estén desdobladas en la jerarquía compilada. Es muy importante hacer notar que, para trabajar muy seriamente y a fondo con *ns-2*, el usuario deberá construirse sus propias clases, así como sus propios protocolos y modificar según sus necesidades los ya existentes. Todo ello conlleva hacerlo en el programa fuente del propio simulador, escrito en C++, y luego volver a compilar todos los módulos para crear un nuevo ejecutable de *ns-2*. No obstante, también se puede ampliar la funcionalidad de *ns-2* a través de una segunda posibilidad: mediante OTcl, dentro de los scripts. Dependiendo de los requisitos, será más apropiado ampliar la funcionalidad del simulador modificando su código fuente, o bien en OTcl, considerando en todo momento el volumen de datos a manejar y operaciones a realizar.

La figura A.2 ofrece una visión simplificada de *ns-2* desde el punto de vista del usuario, que percibe el simulador como un interprete de OTcl y que cuenta con:

- Un planificador de eventos.

- Librerías de objetos con componentes de red: nodos, aplicaciones y protocolos, todos ellos productores y/o consumidores de paquetes.

- Librerías de configuración de red que permiten crear enlaces entre los componentes de red (denominado fontanería).

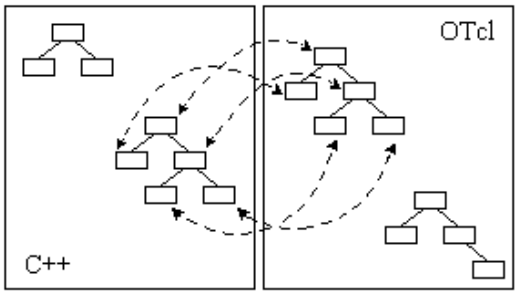

Figura A.1: Dualidad C++ y OTcl en *ns-2*

De manera simplificada se puede observar en la figura A.2, que a la hora de abordar los estudios mediante el simulador *ns-2*, es necesario seguir el siguiente proceso:

1. Implemetación de protocolos o módulos necesarios para realizar el estudio mediante la incorporación de código C++ y OTcl dentro del núcleo del simulador.

2. Descripción de la simulación mediante un script OTcl en la que se detalla el escenario a simular, esta descripción incluye las siguientes acciones: iniciación del planificador de eventos, configuración de la topología de la red usando las librerías de componentes de red y configuración de red, especificación del comienzo y fin de transmisión de cada fuente de tráfico a través del planificador de eventos, entre otras. Nótese que este script es codificado por el usuario y es la entrada que se proporciona al simulador.

3. Análisis de resultados. El simulador dispone de una herramienta denominada NAM (Network AniMator) que permite la visualización del comportamiento de los terminales de la red. A partir de ella, es posible visualizar por ejemplo, como los distintos paquetes de datos se va encaminando a través de los distintos nodos, pérdida de paquetes cuando la red entre entra en congestión, etc. También es posible extraer métricas cuantitativas como resultados de las simulaciones a partir de los ficheros de traza que aporta la simulación, y dada la complejidad, tamaño, y no homogeneidad del fichero de trazas, será preciso en este caso, realizar un post-procesado, utilizando para ello los lenguajes PERL, awk y Java entre otros. Finalmente, tras procesar los datos estos puede representarse de manera gráfica empleando el programa Matlab o gnuplot, disponible tanto para S.O. Windows como para Linux.

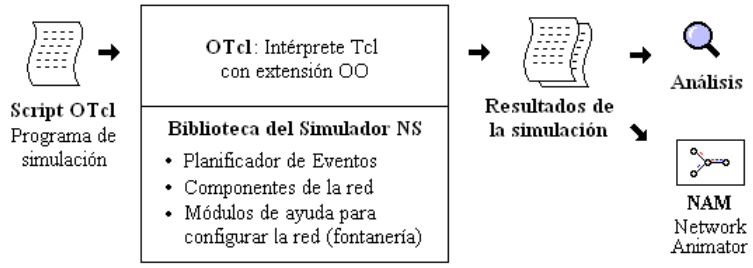

Figura A.2: Estructura simplificada del simulador *ns-2*

A.2. Módulo de Servicios Diferenciados

En esta sección se describe el módulo Diffserv que originalmente fue implementado por del el grupo Advanced IP Networks en Nortel Networks [3] y que se incorporó al simulador en la versión ns-2.1b8.

Tal y como se comentó, la arquitectura diffserv proporciona calidad de servicios dividiendo el tráfico en diferentes categorías, marcando cada paquete con un DSCP determinado que indica la categoría a la que pertenece y despachando el paquete, en consecuencia, de acuerdo a su categoría. El módulo diffserv de *ns* puede soportar hasta cuatro clases de tráfico, cada una de ellas con tres niveles de precedencia, permitiendo de esta forma el tratamiento diferenciado del tráfico en una sola clase.

El módulo Diffserv de *ns* tiene tres componentes. Las principales características de cada uno de estos componentes son las que se reseñan a continuación:

- Políticas de provisión de recursos: Crea las políticas especificadas por el administrador de la red y las distribuye a los routers diffserv. Una política determina cual es el nivel de servicio que se le proporciona a cada uno de los paquetes en la red. Esta asignación puede depender del comportamiento de la fuente (por ejemplo, la tasa media y el tamaño de las ráfagas de datos). En las simulaciones de *ns*, la política de provisión de recursos se determina completamente en el script Tcl.

- Router Frontera: Su responsabilidad es la de asignar a cada paquete el DSCP correspondiente, de acuerdo con la política especificada por el administrador de la red. Para realizar su cometido debe medir, para cada flujo de datos, los parámetros del tráfico entrante.

- Router Interiores (core router): La funcionalidad principal de estos elementos es la de examinar el DSCP de cada paquete y retransmitirlo en consecuencia.

A.2.1. Detalles de Implementación

A fin de diseñar e implementar la arquitectura de servicios diferenciados en *ns*, fue necesario añadir cuatro módulos a la jerarquía de clases del simulador: el primero de ellos

proporciona la funcionalidad básica de un router diffserv (`dsRED`), el siguiente para la definición de los routers interiores (`dsCore`), otro para los routers frontera (`dsEdge`) y finalmente una cuarta clase que se emplea para la administración de las políticas de provisión de recursos.

Técnicamente, cada uno de estos módulos define una nueva clase en *ns*, y cuyos detalles más transcendentales serán objeto de estudio en las siguientes secciones.

A.2.1.1. Cola RED en el Módulo Diffserv

Este módulo se encuentra definido en la clase dsREDQueue, y cuya posición dentro de la jerarquía de clases del simulador puede observarse en la figura A.3. La función de este módulo es la de implementar las funcionalidades básicas de un router diffserv (ver `dsred.{h,cc}`). Por ello, la clase deRedQueue dispone de las siguientes capacidades:

- Proporcionar varias colas físicas RED en un único enlace.
- Proporcionar varias colas virtuales por cada cola física, con un conjunto de parámetros RED individuales para cada una de ellas.
- Determinar la cola física y virtual en la que debe almacenar un paquete entrante de acuerdo con su DSCP.
- Determinar el paquete que debe servirse acorde al algoritmo de planificación de colas seleccionado.

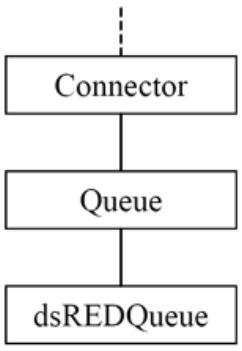

Figura A.3: Posición de la clase dsREDQueue en la jerarquía de clases

De manera más específica, señalar que la clase `dsREDQueue` consta de cuatro colas físicas RED, cada una de ellas con tres colas virtuales. El número de colas físicas y virtuales se especifican en los parámetros `numPrec` y `numQueues_`, donde cada cola física está asociado a una determinada clase de tráfico, y las colas virtuales representan los niveles de precedencia dentro de cada clase. Por tanto, cada combinación cola física − virtual se asocia a un DSCP concreto, que proporciona un nivel de servicio específico para los

paquetes que alberga.

La cola física se define en la clase `redQueue`, que permite la diferenciación de servicios mediante la definición de colas virtuales con parámetros de configuración y estado independientes. Por ejemplo, la longitud de cada una de las colas virtuales se calcula considerando únicamente aquellos paquetes que tienen asignada dicha cola virtual. Por lo tanto, las decisiones de descartar un paquete de una cola virtual determinada puede estar codicionada únicamente por el estado de ocupación y los parámetros de configuración de la cola virtual a la que se encuentra asociado. Debe considerarse, que la clase redQueue no mantiene relación alguna con la clase `REDQueue` (cola RED), que ya se encontraba anteriormente presente en el simulador. No obstante, sí que se trata de una versión modificada de la anterior, en la que se incluye el concepto de cola virtual para realizar el tratamiento diferenciado de los distintos flujos de tráfico. Para realizar la configuración de esta estructura se emplearán los comandos que proporciona la clase `dsREDQueue`.

Además mencionar, que la clase `dsRedQueue` contiene una estructura de datos para definir el comportamiento de salto (PHB) (ver figura A.4). En la arquitectura de servicios diferenciados, tal y como se comentó en el capítulo 1, los routers frontera marcan los paquetes con un DSCP determinado y los routers interiores actúan acorde a este DSCP. Por tanto, se deduce, que ambos emplean esta información para establecer la cola física y virtual a la que está asociado un paquete con un DSCP concreto.

```
struct phbParam{
   int codePt_;    // DSCP del paquete
   int queue_;     // Cola física
   int prec_;      // Cola virtual
}
```

Figura A.4: Parámetros PHB

A.2.1.2. Routers Interior y Frontera

Los routers interiores y frontera se encuentran definidos en las clases `dsCore` y `dsEdge` respectivamente (ver ficheros fuente `dsCore, dsEdge{.h,.cc}`), y cuya posición en la jerarquía de clases de *ns* puede visualizarse en la figura A.5.

El marcado de paquetes se implementa en la clase `edgeQueue`. Un paquete se marca con un determinado DSCP, para que posteriormente pueda situarse en la cola física y virtual correspondiente. Esta clase mantiene una instancia de la clase `PolicyClassifier` que contiene las políticas para el marcado de paquetes.

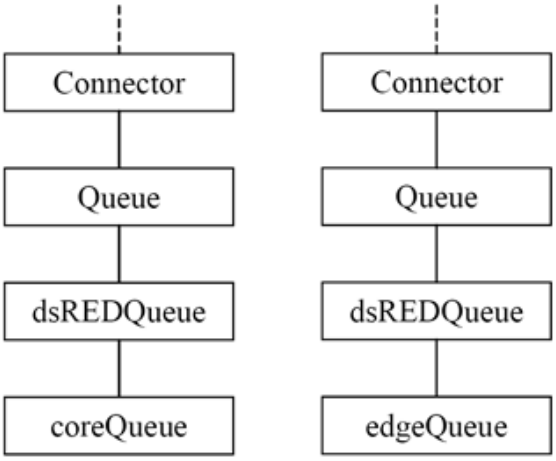

Figura A.5: Posición de las clases dsCore y dsEdge en la jerarquía de clases

A.2.1.3. Módulo de Políticas de Provisión de Recursos

Todos los flujos que tiene los mismos valores dirección origen-destino están sujetas a una política de provisión de recursos común. Esta política define el tipo de función policía empleada, el contrato, y otros parámetros específicos de la función policía empleada. Cada una de ellas requiere por los menos de dos DSCP distintos, uno que será el empleado para marcar aquellos paquetes que se encuentran dentro del perfil y otro para los que están fuera. La elección de uno u otro para llevar a cabo el marcado del paquete, dependerá de la comparación entre el contrato establecido por el cliente y la tasa a la que se encuentra transmitiendo, y posiblemente de algunos parámetros específicos (como por ejemplo el tamaño máximo de ráfaga permitido). En este contexto, se debe especificar cual es el tipo de medidor empleado para obtener las características concretas del tráfico de entrada, es decir, para cada paquete entrante a un router frontera, el medidor debe actualizar las variables internas correspondientes a ese flujo determinado, y marcar el paquete de acuerdo a la política especificada para ese agregado.

Para cada flujo determinado, la información anterior ha de ser suministrada a la tabla de provisión de recursos, en la que cada registro contiene, entre otra, la siguiente información:

- Nodo origen.
- Nodo destino.
- Función policía empleada.
- Tipo de medidor empleado.
- DSCP inicial (DSCP empleado para aquellos paquetes que se encuentren dentro del perfil).
- CIR (*Commited Information Rate*). Tasa de información contratada.

- CBS (*Commited Burst Size*). Tamaño máximo de ráfaga permitido.
- Tiempo de llegada del último paquete.
- Tamaño de ventana empleado en el algoritmo TSW.
- Tasa actual a la que se encuentra transmitiendo la fuente.

Nótese que algunos de los valores anteriores son específicos para una técnica de acondicionado de tráfico concreta, mientras que otros son comunes para todas ellas.

Las funciones policía que se encuentran actualmente implementadas en el simulador son las siguientes:

- TSW2CM (TSW2CMPolicer).
- TSW3CM (TSW3CMPolicer).
- Token Bucket (tokenBucketPolicer).
- CB (CBPolicer).
- CBM (CBMPolicer).
- Single Rate Three Color Marker (srTCMPolicer).
- Two Rate Three Color Marker (trTCMPolicer).
- Null. Ninguno.

A.2.2. Ejemplos de Configuración

El tratamiento diferenciado se implementa en cada enlace de salida mediante las clases dsRED/edge y dsRED/core, que derivan de la clase dsREDQueue:

```
$ns simplex-link $n1 $n2 5Mb 10ms dsRED/edge ;# n1 nodo frontera
$ns simplex-link $n3 $n4 5Mb 10ms dsRED/core ;# n3 nodo interior
```

Capacidad del buffer de la cola de espera y tamaño medio de los paquetes que se almacenan en la cola de espera:

```
$ns queue-limit $n1 $n2 20 ;# capacidad del buffer de 20 paquetes
set q [[$ns link $n1 $n2] queue] ;# cola de espera enlace n1->n2
$q meanPktSize 300 ;# tamaño medio del paquete de 300 octetos
```

Para especificar el número de colas físicas (máximo 4) y de colas virtuales o precedencias (máximo 3) por cada cola física (obteniendo un máximo de doce comportamientos diferenciados, PHB), se emplea:

```
$q set numQueues_ 1 ;# una cola física (número 0)
$q setNumPrec 2 ;# dos colas virtuales (números 0 y 1)
```

Configuración del medidor y marcador (sólo para los nodos frontera), para los paquetes que pertenecen a un flujo determinado. Algunos ejemplos de configuración podrían ser:

```
# Todos los paquetes se marcan con el mismo DSCP (p.ej el 10)
$q addPolicyEntry [$src id] [$dst id] Null 10
$q addPolicerEntry Null 10
$q addPolicyEntry [$src id] [$dst id] TSW2CM 10 $CIR $winLen $beta
# Los paquetes en exceso se marcan con otro DSCP (p.ej. el 11)
$q addPolicerEntry TSW2CM 10 11
```

Los paquetes se distribuyen por las diferentes colas en función de su DSCP (por ejemplo, los paquetes con un DSCP de valor 10 van a la cola virtual 0 de la cola física 0 y los que tienen DSCP de valor 11 a la cola virtual 1 de la cola física 0).

```
$q addPHBEntry <DSCP><queueNum><virtualQueueNum>
$q addPHBEntry 10 0 0
$q addPHBEntry 11 0 1
```

Generalmente, cada cola virtual se configura con unos parámetros RED (*Random Early Discard*) independientes:

```
$q configQ <queueNum><virtualQueueNum><minTh><maxTh><maxP>
$q configQ 0 0 20 40 0.02
$q configQ 0 1 10 20 0.10
```

A fin de comprobar que la configuración de todos los parámetros ha sido correcta, se pueden visualizar las tablas:

```
$q printPolicyTable
$q printPolicerTable
```

Si se dispone de varias colas físicas en un enlace de salida, se debe especificar una disciplina de planificación entre ellas:

```
$q setSchedularMode PRI ;# Prioridad estricta
$q setSchedularMode RR ;# Round Robin
$q setSchedularMode WRR ;# Weighted Round Robin
$q addQueueWeights 0 6 ;# 60% de la capacidad
$q addQueueWeights 1 4 ;# 40% de la capacidad
```

En un instante de tiempo determinado, es posible obtener un resumen de los paquetes que se han transmitido y descartado para cada uno de los niveles de precedencia, para ello se emplea:

```
$ns at 80.0 "$q printCoreStats"
```

```
Packets Statistics
==================================
CP   TotPkts  TxPkts  ldrops  edrops
--   -------  ------  ------  ------
All  37494    37494   0       0
10   15009    15009   0       0
11   22485    22485   0       0
```

A.3. Creación de nuevos módulos

En esta sección se describe el proceso que debe seguirse cuando se desea crear y añadir una nueva clase a la jerarquía de clases del simulador. De manera general, este proceso consta de tres pasos principales: En primer lugar debe determinarse los requisitos que debe cumplir el nuevo módulo a implementar (funcionalidad, características de diseño, ...), porteriormente debe identificarse la posición que debe ocupar en la jerarquía de clases y finalmente se implementa la clase en cuestión, considerando en todo momento, que deben proporcionarse los métodos necesarios para que el usuario pueda interaccionar con el nuevo módulo. Para ilustrar la creación de nuevos módulos se supondrá que la clase que se desea agregar en la jerarquía es la mostrada en la figura A.3.

Generalmente, uno de las fases más complicadas para el programador suele ser la implementación del módulo en cuestión. Dependiendo de la clase, este proceso consta de tres o cuatro puntos principales, que son los que se reseñan a continuación:

1. Creación del archivo de cabecera (`.h`): En este archivo se deben incluir las especificaciones de la clases, así como las definiciones necesarias para la implementación de la misma.

2. Creación del fichero principal de la clase (`.cc`): Este fichero incluye la implementación de todos los métodos que contenga la clase. Para incorporar la nueva clase y hacerla accesible a través de los scripts Tcl, la clase debe ser vinculada a la jerarquía de clases del simulador. Para la clase tomada como referencia, el fichero debe contener el código mostrado en la figura A.6.

3. Modificación del fichero "*Makefile*". En este tercer paso se debe añadir la referencia del módulo en "*Makefile*", de esta forma al invocar el commando make el compilador genera una versión binaria del nuevo código y la incluye en la compilación del simulador: `dsred.o`.

4. Especificación del valor por defecto que tendrán los parámetros que no se inicializan en el constructor de la clase. Este valor por defecto debe especificarse en el fichero de configuración `/ns-2/tcl/lib/ns-default.tcl`. Para que el enlazado de los parámetros especificados en el fichero anterior y los de la clase escrita en C++ sea

```
static class dsREDClass : public TclClass {
public:
    dsREDClass() : TclClass("Queue/dsRED"){}

    TclObject* create(int, const char* const*){
       return (new dsREDQueue);
    }
} class_dsred;
```

Figura A.6: Adición de la clase dsREDClass a la jerarquía de clases

posible, debe añadirse en el constructor el siguiente código:

```
bind("numQueues_", &numQueues_);
```

y en el fichero `/ns-2/tcl/lib/ns-default.tcl` se especifica el valor por defecto de la forma:

```
Queue/dsRED set numQueues_ 4
```

Una vez completado el proceso descrito y recompilar el simulador con el comando make, este nuevo módulo se encontrará disponible en los scripts Tcl.

Apéndice B

Extracción de Resultados

El uso del simulador Ns-2 es bastante simple, primero se crea un archivo de configuración que describe los parámetros de la simulación (topología, patrones de generación de tráfico, configuración de dispositivos, etc.). Tal y como se comentó en el capítulo A, este código de configuración es un script escrito en OTcl que se pasa como parámetro al simulador de redes. A medida que se está simulando la topología, el simulador genera una traza en la que se registran todos los eventos que se han generado durante la simulación. Básicamente, la salida de la simulación consta de dos archivos de texto (uno con extensión .tr y otro con extensión .nam). Ambos ficheros, aunque contienen la misma información, tienen distintos formatos. Al observar estos archivos se distinguen todos los eventos registrados durante el tiempo de simulación línea por línea.

Por una parte, el archivo .nam se puede visualizar con la aplicación de *ns-2* conocida como NAM, programa que tiene como función interpretar estos valores y simularlos en una interfaz gráfica bastante amigable. Mientras que el archivo .tr es necesario procesarlo con un lenguaje de scripts con reconocimiento de patrones y manejo de expresiones regulares, para obtener resultados cuantitativos del escenario simulado. Eso se puede lograr desde la línea de comandos en Linux, y específicamente con lenguajes como awk o Perl.

En la primera parte del capítulo se describirá la estructura del fichero de trazas que proporciona el simulador. La segunda parte se centrará en la descripción de los procedimientos empleados para obtener los resultados obtenidos, así como las herramientas que ha sido necesario desarrollar para facilitar esta labor.

B.1. Estructura del Fichero de Trazas

Cuando en un script se especifica que se va a realizar el trazado de la simulación, el simulador inserta cuatro objetos en el enlace que une dos nodos determinados: *EnqT*, *DeqT*, *RecvT* y *DrpT*, tal y como se observa en la figura B.1. Donde *Enqt* registra la información concerniente a la llegada de un paquete. El paquete en cuestión es almacenado en la cola de entrada del enlace. Si a su llegada la cola se encontrase llena, y el paquete debe descartarse, la información es manejada en este caso por el módulo *DrpT*. El módulo *DeqT* registra el

instante de tiempo en el que el paquete se extrae de la cola. Finalmente, el módulo *RecvT* informa que el paquete ha sido recibido en la salida del enlace.

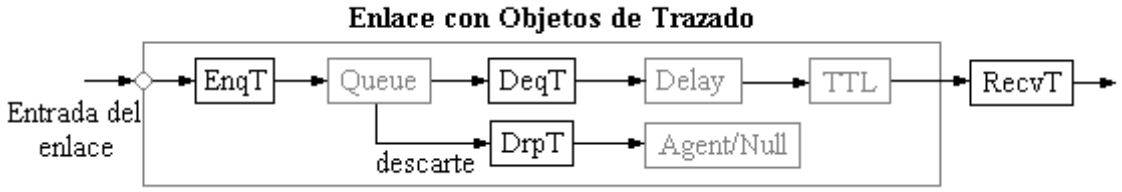

Figura B.1: Estructura del enlace con objetos de trazado

Llegados a este punto, resulta conveniente resaltar, que los componentes de la red se comunican entre sí a través del paso paquetes, sin embargo, estas operaciones no consume tiempo de simulación real. Todos los componentes de red que necesitan consumir cierto tiempo de simulación como consecuencia del manipulando de un paquete (como pueda ser, por ejemplo, un retardo en el enlace), emplean el planificador de eventos atribuyendo un evento al paquete y esperando a que éste sea disparado por sí mismo antes de llevar a cabo cualquier acción de manipulación.

El resultado de realizar la traza de la red, se almacena en un fichero ascii con extesión `tr`, que organiza la información en 12 columnas, tal y como puede observarse en la figura B.2.

Evento	Tiempo	Del Nodo	Al Nodo	Tipo Paquete	Tamaño Paquete	Flags	Fid	Dir. Origen	Dir. Destino	Num Secuenc	Id Paquete

Figura B.2: Estructura del fichero de Trazas

donde:

1. Tipo de evento: Este valor se proporciona a través de cuatro símbolos distintos: $+$, $-$, r y d que significan almacenamiento en cola, extracción de la cola, recepción del paquete a la salida del enlace y descarte respectivamente.

2. Instante de tiempo en el que tiene lugar el evento.

3. Nodo que es el origen del evento en el enlace donde tiene lugar el evento.

4. Nodo que es el destino del evento en el enlace donde tiene lugar el evento.

5. Tipo de paquete (por ejemplo CBR, TCP, etc..). El valor que aparecerá en esta etiqueta se corresponderá con el nombre que recibe la aplicación en el simulador, por ejemplo, para los paquetes de datos TCP, el valor que aparecerá en esta etiqueta será "*tcp*".

6. Tamaño del paquete.
7. Flags.
8. Identificador del flujo de datos (*fid*) de IPv6. Este valor se puede especificar por el usuario en el script OTcl. Este valor puede ser empleado, por NAM, para colear de forma distinta los flujos de datos de cada fuente de datos.
9. Dirección del emisor del paquete en el formato "*idenficador_nodo.puerto*".
10. Dirección destino del paquete en mismo formato que el anterior.
11. Número de secuencia del paquete. A pesar que algunos protocolos, como es el caso de UDP, no emplean números de secuencia en sus cabeceras, el simulador registra este valor en el fichero a fin de facilitar el análisis de los datos obtenidos en la simulación .
12. Identificador único del paquete.

```
- 0.72766 0 15 ack 40 ------- 6 7.0 15.0 10 242
- 0.727687 0 1 tcp 9188 ------- 3 12.0 4.0 11 144
r 0.728215 0 16 ack 40 ------- 7 8.0 16.0 8 236
+ 0.728215 16 0 tcp 9188 ------- 7 16.0 8.0 17 255
- 0.728215 16 0 tcp 9188 ------- 7 16.0 8.0 17 255
+ 0.728215 16 0 tcp 9188 ------- 7 16.0 8.0 18 256
- 0.728215 15 0 tcp 9188 ------- 6 15.0 7.0 20 253
r 0.728232 0 1 tcp 9188 ------- 8 17.0 9.0 10 136
+ 0.728232 1 9 tcp 9188 ------- 8 17.0 9.0 10 136
- 0.728232 1 9 tcp 9188 ------- 8 17.0 9.0 10 136
r 0.72876 16 0 tcp 9188 ------- 7 16.0 8.0 15 246
+ 0.72876 0 1 tcp 9188 ------- 7 16.0 8.0 15 246
r 0.72876 15 0 tcp 9188 ------- 6 15.0 7.0 18 244
+ 0.72876 0 1 tcp 9188 ------- 6 15.0 7.0 18 244
d 0.72876 0 1 tcp 9188 ------- 6 15.0 7.0 18 244
r 0.728777 1 9 tcp 9188 ------- 8 17.0 9.0 9 135
+ 0.728777 9 1 ack 40 ------- 8 9.0 17.0 9 257
- 0.728777 9 1 ack 40 ------- 8 9.0 17.0 9 257
r 0.729332 2 1 ack 40 ------- 1 2.0 10.0 11 251
+ 0.729332 1 0 ack 40 ------- 1 2.0 10.0 11 251
- 0.729332 1 0 ack 40 ------- 1 2.0 10.0 11 251
r 0.729887 1 0 ack 40 ------- 7 8.0 16.0 9 245
```

Figura B.3: Ejemplo de fichero de trazas

Para realizar el análisis de las prestaciones de cada uno de los acondicionares de tráfico evaluados, uno de los valores que se deseaba determinar era la cantidad de paquetes que se marcaban con IN y como OUT a fin de obtener el caudal de paquetes IN y el ancho de banda excedente que conseguía cada una de las fuentes de tráfico. Como puede observarse en la figura B.2, el fichero de trazas no refleja en ningún momento cuál es el DSCP con el que se

ha marcado un paquete determinado. Por esta razón, se hizo necesario, modificar la clase que genera el fichero (`TraceClass`) para que refleje el valor del DSCP con el que va marcado un paquete en todo momento, es decir, las trazas con las que se ha trabajado en este proyecto incluyen realmente 13 campos para cada registro, donde el último valor que se incluye en cada uno de ellos es el valor del DSCP con el que se ha marcado el paquete.

B.2. Monitores

En la sección anterior, se comentó que los la traza de la simulación registra los datos para cada llegada de paquete, salida o descarte. Aunque un usuario consiga la información suficiente a partir del fichero de traza generado, en muchas ocasiones, resulta conveniente conocer los parámetros específicos y su evolución a lo largo del tiempo de algún elemento de la topología.

El simulador, además del fichero comentado, ofrece la posibilidad de utilizar los denominados monitores. El objetivo principal de un monitor de NS-2 es la recogida de datos de un determinado elemento de la topología, para ello implementa una serie de contadores de algunos parámetros específicos (número total de paquetes o bytes recibidos, descartados, etc..), siendo especialmente útiles cuando se desea conocer información básica de la dinámica de la simulación.

En muchas ocasiones, los objetos que implementan los elementos que forman parte de la topología de una red proporcionan una serie de funciones orientadas a la monitorización de ciertos parámetros de estado del elemento. A modo ilustrativo, supóngase que se desea monitorizar la ocupación media e instántanea de una cola RED situada en un enlace que une dos routers (*r1* y *r2*). En este caso el codigo OTcl necesario para llevar a cabo la monitorizarión de estas dos propiedades de la cola RED es el mostrado en la figura B.4.

```
;# Cola del enlace r1 -> r2
set redq [[$ns link $r1 $r2] queue]

;# Fichero donde se volcará la monitorización
set tchan_ [open all.q w]

;# Configuración del monitor
$redq trace curq_ ;# Ocupación instántanea
$redq trace ave_ ;# Ocupación media de la cola
$redq attach $tchan_
```

Figura B.4: Monitorización de una cola RED

El fichero que se obtiene al hacer uso de este monitor (all.q), para este caso particular de monitor, tiene el siguiente formato mostrado en la figura B.5.

```
Q  20.6326  25
a  20.6348  65.0988
a  20.637   64.9425
a  20.6393  64.8647
```

Figura B.5: Fichero monitor resultante

Donde la primera columna especifica el tipo de valor recogido (a = valor medio, Q = valor instántaneo), la segunda el instante de tiempo y la tercera el valor. Una vez procesado el fichero, los resultados obtenidos pueden representarse gráficamente (ver figura B.6).

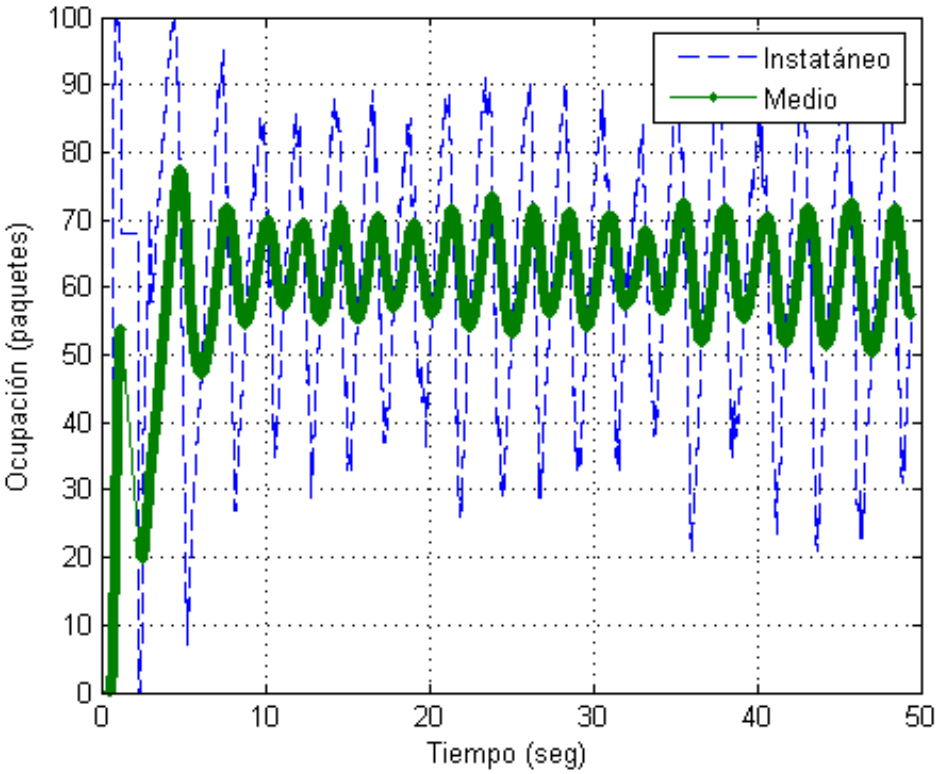

Figura B.6: Evolución de la ocupación media e instántanea de una cola RED

Si la clase en cuestión, no ofrece soporte para la configurción de monitores, es posible configurarlos en OTcl haciendo uso del planificador de eventos, sin necesidad de modificar el código fuente del simulador. La base sobre la que se sustentan este tipo de monitores es muy simple, básicamente consiste en planificar un evento para un instante de tiempo determinado. Este evento consitirá en ejecutar un procedimiento que extraerá los datos de interés considerando en todo momento, que este procedimiento debe planificar el instante de tiempo en el que tendrá lugar la siguiente extracción de datos.

B.3. Herramientas Desarrolladas

Tal y como se comentó anteriormente, uno de los principales inconvenientes que presenta el fichero de trazas es que no proporciona de manera directa resultados cuantativos relativos a la simulación realizada. Por esta razón, se hace necesario realizar un post-procesado para extraer determinadas características. El lenguaje empleado en este caso para realizar este procesado de los ficheros de traza es *awk*.

Awk es un lenguaje de propósito general diseñado para procesar archivos estructurados y con patrones de texto. Dispone de características internas para descomponer líneas de entrada en campos y comparar estos campos con patrones que se especifiquen. Debido a estas posibilidades, resulta particularmente apropiado para trabajar con archivos que contienen información estructurada en campos. La función básica de *awk* es buscar líneas en ficheros (u otras unidades de texto) que contengan ciertos patrones. Cuando en una línea se encuentra un patrón determinada, realiza sobre la línea las acciones especificadas para dicho patrón. *Awk* sigue realizando el procesamiento de las líneas de entrada de esta forma hasta que se llega al final del fichero.

Dada la estructura de los ficheros generados por la simulación, *awk* es especialmente útil para realizar el procesado de estos archivos. Permite obtener resultados, tanto en filtrado de datos como en resumen resultados, de una forma relativamente acelerada, en relación a la considerable envergadura de los archivos y la información a tratar. Los programas *awk* empleados para tal fin son `avgStats.awk` e `instantThroughput.awk`, cuyas características más relevantes son las que se reseñan a continuación:

- **`avgStats.awk`**: Este programa calcula para un determinado flujo de datos los valores medios del retardo, el throughput total y throughput alcanzado para los paquetes que se encuentran dentro del perfil.

  Uso: `'awk -f avgStats.awk <values><tracefile>'`

`<values>:`	`src`	Dirección origen del flujo de datos.
	`dst`	Dirección destino del flujo de datos.
	`flow`	Identificador del flujo de datos.
	`pkt`	Tamaño en bytes de los paquetes del flujo.

  `<tracefile>:` Fichero que se desea analizar.

  A modo ilustrativo, si desde del intérprete de comandos se ejecuta el comando:

  ```
  awk -f avgStats.awk src=10.0 dst=2.0 flow=1 pkt=9188 out.tr
  ```

  se obtiene la información mostrada en la figura B.7.

- **`instantThroughput.awk`**: Proporciona la evolución temporal del throughput de los paquetes IN. Esta información puede representarse posteriormente de manera gráfica.

  Uso: `'awk instantThroughput.awk <values><tracefile>'`

`<values>:`	`tic`	Granulidad de los datos representados.
	`src`	Dirección origen del flujo de datos.
	`dst`	Dirección destino del flujo de datos.
	`flow`	Identificador del flujo de datos.
	`pkt`	Tamaño en bytes de los paquetes del flujo.

`<tracefile>:` Fichero que se desea analizar.

A modo ilustrativo, si desde del intérprete de comandos se ejecuta el comando:

```
awk -f instantThroughput.awk tic=1.0 src=10.0 dst=2.0
flow=1 pkt=9188 out.tr
```

se obtiene la información mostrada en la figura B.8.

```
          flowID:  1
        flowType:  tcp
         srcNode:  10
         dstNode:  2
       startTime:  0
        stopTime:  99
    receivedPkts:  4202
   avgTput[kbps]:  3099.31
 avgTputIn[kbps]:  925.661
    avgDelay[ms]:  55.298
```

Figura B.7: Ejemplo de salida de awgStats.awk

```
flow  src  dst    time    throughput
  1    10    2   1.20020    11464.4
  1    10    2   2.51461    7828.83
  1    10    2   3.52374    2039.56
  1    10    2   4.55725    4836.21
 ...   ...  ...     ...        ...
```

Figura B.8: Ejemplo de salida de instantThroughput.awk

Uno de los objetivos de este documento era el análisis de las prestaciones de diversos acondicionadores de tráfico propuestos. Por tanto, resulta obvio, que es necesario analizar el impacto que tiene la configuración de ciertos parámetros en el funcionamiento de cada uno ellos. Por esta razón se ha desarrollado un programa en Java que permite automatizar la simulación de los diversos escenarios variando los parámetros a analizar. Este programa lanza 10 simulaciones para cada conjunto de valores, posteriormente empleando los

programas *awk* comentados anteriormente, extrae los resultados del fichero de traza generado. Una vez finalizadas la simulaciones, los resultados se almacenan en un fichero que contiene la información mostrada en la figura B.9, donde D(Tput) y D(TputIn) son respectivamente la desviación típica estándar del caudal total y la desviación típica estándar del caudal de paquetes IN.

RTT	src	TputIn(kbps)	D(TputIn)	Tput(kbps)	D(Tput)	BW(exceso)	Porc
80	10	1001.03	0.64	2200.85	91.39	1199.82	9.23
70	11	1000.98	0.42	2447.93	83.48	1446.95	11.13
60	12	2001.15	2.12	3416.81	106.84	1415.66	10.89
50	13	2003.02	0.16	3519.42	73.23	1516.40	11.66
40	14	3003.48	1.15	4323.48	151.42	1320.00	10.15
30	15	3003.13	2.09	4764.11	66.22	1760.98	13.55
20	16	4001.90	2.41	5781.36	54.51	1779.46	13.69
10	17	4003.89	2.67	6254.26	73.28	2250.37	17.31

Figura B.9: Automatización de las simulaciones

Para la representación gráfica de los resultados obtenidos pueden emplearse diversos programas. Por una parte, se dispone de Xgraph, una herramienta proporcionada por *ns-2* que permite crear archivos postscript, Tgif, entre otros. El programa recibe como argumento uno o más archivos de tipo ASCII que contiene los datos en forma de pares ordenados x-y en cada línea. Puede ser llamada en el script OTcl para representar los datos de la simulación una vez que ésta haya finalizado.

Otra opción disponible es la herramienta *gnuplot*. Se trata de un programa en línea de comandos que permite representar funciones en 2 y 3 dimensiones a través de las fórmulas que las definen. También puede dibujar gráficos usando una tabla de coordenadas (en formato sólo texto) creadas con cualquier programa. Además de representar la gráfica en pantalla, permite almacenarla en multitud de formatos entre los que se encuentran los usuales, como jpg, png, pdf, svg; y otros, menos usuales, pero muy interesantes para los usuarios LaTeX como metafont, eps, pstricks, picture, etc.. El software tiene copyright pero se distribuye libremente y está disponible para UNIX, Linux, IBM OS/2, MS Windows, MSDOS, Macintosh, VMS, Atari y muchas otras plataformas.

A fin de facilitar la generación de gráficas de resultados, se ha desarrollado una interfaz gráfica de usuario en Java para el manejo de ficheros los proporcionados por los objetos monitor de *ns-2* o los resultados obtenidos en el post-procesado con *awk* (ver figuras B.10 y B.11). El uso de esta aplicación bastante simple, básicamente debe cargarse el fichero que desea analizarse. Durante este proceso es posible filtrar ciertos valores del fichero, por ejemplo, se puede especificar que se muestren los valores referentes a una determinado dirección origen y/o destino. De esta forma únicamente se mostraran los registros que cumplan la condición especificada en el filtro. A partir de los datos es posible realizar una

representación gráfica utilizando la herramienta que proporciona para tal fin. Uno de los principales inconvenientes que presenta esta utilidad, es que las representaciones gráficas no suele ser muy precisas. Sin embargo, su verdadera potencia radica en la capacidad que tiene para generar de manera sencilla los scripts de *gnuplot* o Matlab necesarios para representar los datos.

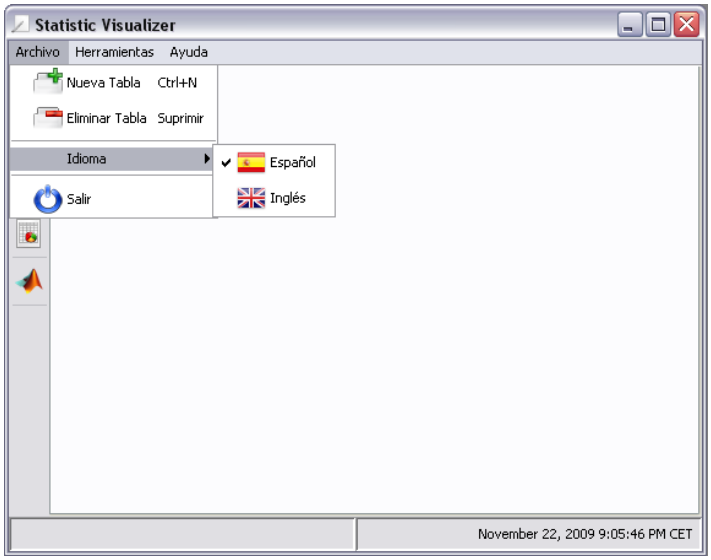

Figura B.10: Herramienta de visualización de resultados. Vista general

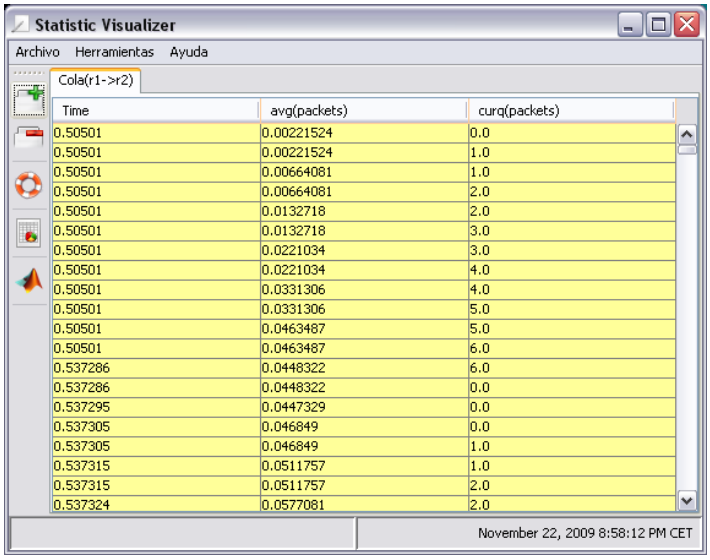

Figura B.11: Herramienta de visualización de resultados. Análisis de datos

Lista de Acrónimos

A

AF PHB	PHB de encaminamiento asegurado (*Assure Forwading PHB*)
AS	Servicio Asegurado (*Assure Forwading*)
ATM	Modo de transferencia asíncrono (*Asynchronous Transfer Mode*)

B

BA	Comportamiento agregado (*Behavior Aggregated*)
BE	Mejor Esfuerzo (*Best Effort*)

C

CB	Acondicionador de tráfico basado en contadores (*Counters Based*)
CBDQ	CB empleado junto con una gestión de buffer de doble cola (*CB Dual Queuing*)
CBM	Acondicionador de tráfico basado en contadores modificado (*Counters Based Modified*)
CIR	Tasa de información contratada (*Commited Information Rate*)

D

Diffserv	Servicios diferenciados (*Differenciated Services*)
DS	Servicios diferenciados (*Differenciated Services*)
DSCP	Código diffserv (*Diffserv Code Point*)

E

ECN	Notificación de congestión explícita (*Explicit Congestion Notification*)
EF PHB	PHB de encaminamiento expeditivo (*Expedited Forwarding PHB*)
EWMA	Media móvil ponderada exponencialmente (*Exponentially Weighted Moving Average*)

F

FIFO	Primero en entrar, primero en salir (*First In First Out*)
FQ	Encolamiento justo (*Fair Queuing*)

H

HBO	Ocupación alta de cola (*High Buffer Occupancy*)

I

ICMP	Protocolo de mensajes de control de Internet (*Internet Control Message Protocol*)
IETF	Grupo para el desarrollo de Internet (*Internet Engineering Task Force*)
IntServ	Servicios integrados (*Integrated Services*)
IP	Protocolo de Internet (*Internet Protocol*)
ISP	Proveedor de servicios de Internet (*Internet Services Provider*)

L

LAN	Red de área local (*Local Area Network*)

M

MPLS	Conmutación por etiquetas multiProtocolo (*MultiProtocol Label Switching*)
MSS	Tamaño máximo de segmento (*Maximum Segment Size*)

N

NS-2	Simulador de redes versión 2 (*Network Simulator version 2*)

P

PETER	Acondicionador de tráfico con reparto proporcional (*Proportional Excess Traffic conditionER*)
PHB	Comportamiento por salto (*Per Hop Behavior*)
PQ	Planificación por prioridad (*Priority Queuing*)

Q

QoS	Calidad de Servicio (*Quality of Service*)

R

RED	Detección temprana de congestión (*Random Early Detection*)
RIO	RED con bits In y Out (*RED In and Out*)
RR	Planificación por turnos (*Round Robin*)
RTT	Tiempo de ida y vuelta (*Round Trip Time*)
RSVP	Protocolo de reserva de recursos (*Resource reSerVation Protocol*)

S

SLA	Acuerdo de nivel de servicio (*Service Level Agreement*)
SRTCM	Acondicionador de tasa única con marcado de tres colores (*Single Rate Three Color Marker*)

T

TB	Acondicionador de tráfico Token Bucket (*Token Bucket*)

TC	Acondicionador de tráfico (*Traffic Conditioner*)
TCA	Acuerdo de acondicionado de tráfico (*Traffic Conditioning Agreement*)
TCS	Especificación de acondicionado de tráfico (*Traffic Conditioning Specification*)
TCP	Protocolo de transmisión controlada (*Transmission Control Protocol*)
ToS	Tipo de servico (*Type of Service*)
TRTCM	Acondicionador de dos tasas con marcado de tres colores (*Two Rate Three Color Marker*)
TSW	Acondicionador de ventana temporal colores (*Time Sliding Window*)

U

UDP	Protocolo de datagramas de usuario (*User Datagram Protocol*)

V

VoIP	Voz sobre IP (*Voice over IP*)

W

WFQ	Planificación justa por pesos (*Weighted Fair Queuing*)
WRR	Planificación RR por pesos (*Weighted Round Robin*)

Referencias Bibliográficas

[1] The Network Simulator *ns-2*, Home Page. Artículo de Página web, 2009. `http://www.isi.edu/nsnam/ns/`

[2] K. Fall y K. Varadhan, "*ns* Notes and documentation". The VINT Project UC Berkeley, LBL, USC/ISI and Xerox PARC. Enero 2009.

[3] Peter Pieda, Jeremy Ethridge, Mandeep Baines y Farhan Shallwani, "A Network Simulator, Differentiated Services Implementation", Open IP, Nortel Networks. Julio 2000.

[4] Eitan Altman y Tania Jiménez, "Simulator Course for Beginners". Lecture Notes, Univ. de Los Andes, Merida, Venezuela. Septiembre 2002.

[5] Sally Floyd, "Validation Experiences with the NS Simulator". Abril 1999.

[6] Cygwin information and installation, Home Page. Artículo de Página web, 2009. `http://www.cygwin.com/`

[7] Jae Chung y Mark Claypool, "NS by Example", Home Page. Artículo de Página web, 2009. `http://nile.wpi.edu/NS`

[8] María Dolores Cano, "Nuevas técnicas de control y gestión de tráfico en Internet para proporcionar Calidad de Servicio extremo a extremo", Tesis Doctoral - Universidad Politénica de Cartagena, 2004.

[9] María Dolores Cano, Pablo Lopez-Matencio, Juan José Alcaraz, Fernando Cerdán, "El papel de los acondicionadores de tráfico para ofrecer Calidad de Servicio extremo a extremo", Revista II Teleco-Forum, Universidad Politécnica de Cartagena, pp. 80-82, Cartagena, España, Abril 2004.

[10] María Dolores Cano, Juan José Alcaraz, Pablo Lopez-Matencio, Fernando Cerdán, "CBDQ: Garantía de Calidad de Servicio en Internet",Proceedings de XIII Jornadas Telecom. I+D 2003, Madrid, Noviembre 2003.

[11] María Dolores Cano, Fernando Cerdán, Joan García Haro, Josemaria Malgosa Sanahuja, "Análisis de las Prestaciones del Acondicionador de Tráfico Counters Based Modified en un Dominio DiffServ", Proceedings de Jornadas de Ingeniería Telemática JITEL '03, Las Palmas de Gran Canaria, España, Septiembre 2003.

[12] Miguel Ángel Martinez Ramirez, María Dolores Cano, "Evaluación de Técnicas de Acondicionado del Tráfico en ns-2", Proyecto Final de Carrera, Universidad Politécnica de Cartagena, 2009.

[13] Maria-Dolores Cano, Fernando Cerdan, Miguel-Angel Martinez-Ramirez, "Traffic Conditioners for QoS Assurance", Internet Policies and Issues, Nova Science Publishers, Inc., Vol. 8, pp. 1-36, 2011.ISBN: 978-1-61122-840-3

[14] Sally Floyd, "RED (Random Early Detection) Queue Management". Artículo de Página Web, 2009. `http://www.icir.org/floyd/red.html`.

[15] R. Jain, D. CHI and W. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer system", Hudson, MA: Eastern Research Laboratory, Digital Equipment Corporation, 1984

[16] R. Jain, "The Art of Computer Systems Performance Analysis", John Wiley and Sons Inc., 1991

[17] R. Braden, D. Clark y S. Shenker, "Integrated Services in the Internet Architecture: an Overview", RFC 1633, Junio 1994.

[18] R. Braden et. al., "Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification", RFC 2205 (Actualizada en RFC 2750), Septiembre 1997.

[19] B. Braden et. al., "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC2309, Abril 1998.

[20] K. Nichols, S. Blake, F. Baker, D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, Diciembre 1998.

[21] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, "An Architecture for Differentiated Services", RFC 2475, Diciembre 1998.

[22] J. Heinanen, F. Baker, W. Weiss, J. Wroclawski, "Assured Forwarding PHB Group", RFC 2597, Junio 1999.

[23] V. Jacobson et. al., "An Expedited Forwarding PHB", RFC 2598, Junio 1999.

[24] J. Heinanen, Telia Finland y R. Guerin, "A Single Rate Three Color Marker", RFC 2697, Septiembre 1999.

[25] J. Heinanen, Telia Finland y R. Guerin, "A Two Rate Three Color Marker", RFC 2698, Septiembre 1999.

[26] V. Jacobson y K. Nichols y L. Zhang, "A Twobit Differentiated Services Architecture for the Internet", RFC 2638, Julio 1999.

[27] B. Davie, A. Charny, J.C.R. Bennet, K. Benson, J.Y. Le Boudec, W. Courtney, S.Davari, V. Firoiu, D. Stiliadis, "An Expedited Forwarding PHB (Per-Hop Behavior)", RFC 3246, Marzo 2002.

[28] D. Grossman, "New Terminology and Clarifications for DiffServ", RFC 3260, Abril 2002.

[29] S. Floyd y V. Jacobson, "Random Early Detection gateways for congestion Avoidance", ACM/IEEE Transactions on Networking, Vol.1 No.4, pp. 397-413, Agosto 1993.

[30] D. Clark y W. Fang, "Explicit Allocation of Best-Effort Packet Delivery Service", IEEE/ACM Transactions on Networking, Vol. 6 No. 4, pp. 362-373, Agosto 1998.

[31] W. Fang, "Differentiated Services: Architecture, Mechanisms and an evaluation", PhD Dissertation - Princeton University, Noviembre 2000.

[32] J. Ibanez y K. Nichols, "Preliminary Simulation Evaluation of an Assured Service", Internet draft, work in progress, draft-ibanez-diffserv-assured-eval-00.txt, Agosto 1998.

[33] José Manuel Giménez Guzmán y Jorge Martínez Bauset, "Red Experimental DiffServ Utilizando Router-PCs con Sistema Operativo Linux", Poster en Jornadas Técnicas de RedIRIS 2003. Palma de Mallorca, España. Noviembre 2003.

[34] FREE SOFTWARE FOUNDATION, Gnu general public licence, `http://www.gnu.org/copyleft/gpl.html`

[35] "NS-3", The Nsnam Wiki, Information Sciences Institute, Julio 2006 `http://nsnam.isi.edu/nsnam/index.php/Ns-3`

[36] Tcl Developer Xchange, Home Page. Artículo de Página web, 2009. `http://www.tcl.tk/software/tcltk/`

[37] Tcl/Tk Quick Reference Guide, Home Page. Artículo de Página web, 2009. `http://www.slac.stanford.edu/~raines/tkref.html`