UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

DISEÑO DE UNA RED PARA ANALIZAR EL EFECTO DE LOS TIPOS DE COLAS

Alberto Martínez Fernández

Director: Jose Oscar Romero Martínez

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingeniería de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2021-22

Valencia, 7 de enero de 2022

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS Gestión ambiental verificada REG.NO. ES-CV- 000030

Resumen

El presente Trabajo Final de Grado se centra en el estudio de la implementación de QoS, que permite priorizar el tráfico para así garantizar cierto nivel de rendimiento en aplicaciones y flujos de datos de alta importancia a través de la optimización de los recursos disponibles (sin expandir innecesariamente la infraestructura de la red).

Por ello, el objetivo principal que sostiene la investigación es el diseño de una red corporativa y la recopilación -así como el análisis- de las estadísticas obtenidas del mismo para observar el efecto que tienen sobre la red las colas y los condicionantes de tráfico utilizados a la hora de implementar QoS en un entorno controlado empleando Omnet++.

Para analizar las hipótesis y profundizar en los resultados, el trabajo se organiza en tres bloques principales: en primer lugar, el estudio de los fundamentos de QoS necesarios para el proyecto; en segundo lugar, la revisión del funcionamiento del simulador Omnet++ utilizado en las simulaciones de la red analizada (para así lograr una mejor comprensión del diseño y de los resultados obtenidos); y por último, el desarrollo de la propuesta -configuración de la red, configuración de los escenarios y análisis de los resultados-.

Palabras clave: Omnet++; simulador de red; diseño de red; tráfico; colas

Resum

El present Treball Final de Grau se centra en l'estudi de la implementació de Qos, que permet prioritzar el trànsit per a així garantir un cert nivell de rendiment en aplicacions i fluxos de dades d'alta importància a través de l'optimització dels recursos disponibles (sense expandir innecessàriament l'infrastructura de la xarxa).

Per això, l'objectiu principal que sosté la investigació és el disseny d'una xarxa corporativa i la recopilació -així com l'anàlisi- de les estadístiques obtingudes del mateix per a observar l'efecte que tenen sobre la xarxa les cues i els condicionants de trànsit utilitzats a l'hora d'implementar QoS en un entorn controlat emprant Omnet++.

Per a analitzar les hipòtesis i aprofundir en els resultats, el treball s'organitza en tres blocs principals: en primer lloc, l'estudi dels fonaments de Qos necessaris per al projecte; en segon lloc, la revisió del funcionament del simulador Omnet++ utilitzat en les simulacions de la xarxa analitzada (per a així aconseguir una millor comprensió del disseny i dels resultats obtinguts); i finalment, el desenvolupament de la proposada -configuració de la xarxa, configuració dels escenaris i anàlisis dels resultats-.

Abstract

This Final Degree Project focuses on the study of the implementation of QoS, which allows the prioritization of traffic in order to guarantee a certain level of performance in applications and data flows of high importance through the optimization of available resources (without unnecessarily expanding the network infrastructure).

Therefore, the main objective of the research is the design of a corporate network and the collection -as well as the analysis- of the statistics obtained from it in order to observe the effect on the network of the queues and traffic constraints used when implementing QoS in a controlled environment using Omnet++.

In order to analyze the hypotheses and deepen the results, the work is organized in three main blocks: firstly, the study of the QoS fundamentals necessary for the project; secondly, the review of the operation of the Omnet++ simulator used in the simulations of the analyzed network (in order to achieve a better understanding of the design and the results obtained); and finally, the

development of the proposal -network configuration, scenario configuration and analysis of the results-.

INDICE

Índice de figuras

1. Introducción

QoS, acrónimo de Quality of Service, Calidad de servicio en español, es una característica que incluyen los routers y switches que permite priorizar el tráfico de determinados servicios considerados como importantes o prioritarios. Es el caso de videollamadas, una actividad que necesita un determinado ancho de banda para su correcto funcionamiento.

Teniendo en cuenta que en la actualidad conectamos decenas de dispositivos al router, desde televisores inteligentes a sensores, alarmas o dispositivos conectados, pasando por smartphones y tablets, se hace necesario priorizar qué dispositivos o qué tareas obtienen un tráfico de red suficiente para tareas como videollamadas.

1.1 Objetivos

El objetivo principal de este TFG consiste en la simulación de una red de comunicaciones para el análisis de las tecnologías de QoS más comunes y así comprobar su rendimiento y ayudar a comprender su funcionamiento, empleando para ello un entorno de simulación de redes como es Omnet++ el cual permite ver el efecto que tienen sobre diferentes redes los elementos de QoS. Además de esto Omnet++ posee herramientas que permiten recopilación de datos de manera rápida y efectiva lo cual será de gran utilidad a la hora de visualizar el efecto que tienen los mecanismos QoS sobre la red.

1.2 Estructura

Este Trabajo de Fin de Grado se ha organizado en seis capítulos distintos para poder así explicar las aportaciones teóricas, así como el entorno de simulación empleado en el proyecto y el desarrollo del proyecto.

El primer capítulo de introducción sirve como un marco metodológico al proyecto, y a su vez, como entrada al tema en cuestión ya que aquí formularemos las hipótesis a desarrollar a lo largo del proyecto y sus objetivos.

En el segundo capítulo se van a presentar los conceptos introductorios a QoS, así como los modelos empleados para brindar QoS y los indicadores que emplearemos para observar el rendimiento de la red. Además de esto también se introducirán métodos para tratar con la congestión, el principal causante de los problemas en la red.

En el tercer capítulo se estudiará el entorno de simulación empleado en el proyecto para observar tanto sus capacidades como limitaciones. Además, se prestará especial atención al framework INET el cual es de especial interés en la simulación de redes al emplear Omnet++.

A lo largo del cuarto capitulo se detallará el desarrollo llevado a cabo en el proyecto para simular la red sobre la que queremos observar los efectos de los mecanismos QoS. Esto supone explicar tanto la configuración de los módulos empleados para simular la red como las configuraciones de los escenarios de simulación que se han realizado. Además de todo esto se presentan los resultados obtenidos de las simulaciones para observar el rendimiento de la red a lo largo de los distintos escenarios.

En el quinto capitulo se presentará un resumen de los resultados anteriormente obtenidos y las conclusiones a la que los mismos llevan.

Para finalizar, en el sexto capitulo se halla la bibliografía, donde se listan todas las fuentes de las que se ha obtenido la información gracias a las cuales el proyecto ha sido posible.

1.3 Metodología

Para lograr los objetivos descritos previamente se ha empleado la siguiente metodología:

En primer lugar, se ha realizado un estudio de los conceptos básicos de QoS junto con los modelos más usados y sus respectivas limitaciones en cuanto a rendimiento y calidad ofrecida. Además, se ha estudiado el efecto de los algoritmos más usados para tratar con la congestión de la red.

Seguidamente se ha realizado un estudio de la herramienta de simulación empleada (Omnet++) para así lograr una mejor comprensión del diseño y de los resultados obtenidos, así como de las limitaciones de este.

Por último, se han realizado una serie de pruebas de simulación creadas en el entorno Omnet++ para analizar las características de rendimiento de la red y así determinar el impacto de los condicionantes de tráfico QoS y los algoritmos de colas sobre la misma.

2. Introducción a QoS

2.1 Conceptos básicos

La calidad de servicio, del inglés Quality of Service (QoS), es el término que usamos para definir la capacidad que tiene una red para proveer un servicio de manera confiable empleando unos recursos limitados. Los mecanismos de QoS permiten a los administradores de la red asignar niveles de prioridad en el tráfico o niveles de calidad con respecto al ancho de banda o retardo punto a punto. Esto hace QoS especialmente importante en servicios sensibles a retardos como puede ser VoIP (Voz sobre Internet Protocol) [1] [2].

Las herramientas utilizadas para garantizar QoS le otorgan al administrador de la red más control, lo cual le permite facilitar su tarea. Estas herramientas de QoS permiten asegurar el correcto funcionamiento de aplicaciones que requieran de ciertas garantías, así como maximizar el uso de recursos para no malgastarlos y también se puede emplear para responder a cambios en la red ya sea una fluctuación en el tamaño de los flujos de datos como errores en la infraestructura.

Algunas de las tareas que QoS facilita al administrador de la red serían:

- Dar prioridad a ciertas aplicaciones de la red.
- Maximizar el uso de recursos de la red.
- Mejorar el funcionamiento de aplicaciones que requieren de un determinado ancho de banda o de un bajo retardo extremo a extremo.
- Responder al cambio de flujos de datos en la red.

2.2 Indicadores de QoS

Los principales parámetros de calidad que estudiaremos en este trabajo serán:

- Retardo extremo a extremo. Es el retardo total causado entre dos puntos de la red compuesto por cuatro tipos diferentes de retardos: de procesamiento de las cabeceras, de tiempo en cola, de tiempo de transmisión y de tiempo de propagación. De todos estos retardos, el único que podemos mitigar a la hora de aplicar QoS es el retardo de cola. El retardo extremo a extremo es especialmente importante en aplicaciones que requieran retroalimentación a tiempo real, como pueden ser aplicaciones de VoIP.
- Jitter. Se conoce como jitter a la variación en el retardo extremo a extremo entre paquetes. Es un parámetro importante en aplicaciones que ofrezcan servicios real-time como puede ser VoIP. El jitter es generalmente ocasionado por la congestión de la red, por ello trataremos de disminuirlo mediante la aplicación de mecanismos QoS.
- Ratio de pérdida de paquetes. La pérdida de paquetes ocurre cuando un paquete de la red no consigue alcanzar su destino ya sea por errores de transmisión o debido a que la congestión de la red fuerza a los routers o conmutadores a descartar el paquete. En aplicaciones de tiempo real estas pérdidas de paquetes causan una pérdida de información afectando a la experiencia de usuario.
- Throughput. Es la cantidad de paquetes por unidad de tiempo que se reciben de manera exitosa en su destino. Se mide de manera habitual en bps (bits por segundo) y es un factor clave en la calidad de un servicio en la red.

2.3 Modelos de QoS

Los dos modelos principales a la hora de implementar QoS en una red IP (Internet Protocol) son los modelos IntServ (Integrated Services) y DiffServ (Differentiated Services). Un tercer método de servicio es el Best Effort, que consiste en el comportamiento por defecto del dispositivo de red sin aplicar ningún modelo QoS.

2.3.1 Best Effort

Es el modelo más básico de los tres existentes y aquel que se encuentra por defecto en internet y en las redes basadas en IP, que no implementan ningún mecanismo de QoS. Best Effort trata por igual a todos los paquetes de la red sin dar prioridad a ningún tráfico en específico, debido a esto el modelo Best Effort es útil para servicios que no tengan requerimientos de retardo extremo a extremo o pérdida de paquetes.

2.3.2 IntServ

La arquitectura de servicios integrados, definida en la RFC 1633, trata de brindar QoS, requerido en aplicaciones con sensibilidad a retardos como pueden ser las aplicaciones VoIP, en las redes basadas en IP que debido a la naturaleza del protocolo ofrecen un servicio best-effort.

IntServ emplea un sistema de reserva de recursos para brindar recursos a las aplicaciones que los requieran. Este sistema emplea RSVP (Resource ReSerVation Protocol) y reserva recursos a lo largo de todo el recorrido del flujo de datos hasta que una de las partes decide cortar la comunicación. La arquitectura de IntServ está principalmente pensada para evitar el retardo extremo a extremo más que para obtener ancho de banda.

RSVP es usado por un host para solicitar la reserva de los recursos en la red para así asegurar la calidad de servicio en ese flujo de datos específico.

IntServ utiliza RSVP para señalar explícitamente las necesidades de QoS del tráfico de una aplicación a lo largo de los dispositivos (routers y conmutadores) en la ruta extremo a extremo a través de la red. Si todos estos dispositivos pueden reservar el ancho de banda necesario, la aplicación de origen puede comenzar a transmitir.

Además de la señalización de extremo a extremo, IntServ requiere varias prestaciones en los routers y conmutadores:

- **Control de admisión**. Permite determinar si un nuevo flujo puede recibir la QoS solicitada sin afectar a las reservas existentes
- **Clasificación de paquetes**. Permite reconocer los paquetes que necesitan niveles particulares de QoS
- **Policing**. Permite tomar medidas, incluida la posible eliminación de paquetes, cuando el tráfico no se ajusta a las características especificadas
- **Colas y planificación**. Permite reenviar los paquetes de acuerdo con las solicitudes de QoS que se han concedido.

IntServ distingue dos tipos de flujos de datos: aplicaciones con tolerancia y aplicaciones intolerantes. Las aplicaciones con tolerancia no se ven afectadas por retardos, el jitter o incluso eventos en la red que interrumpan el flujo de datos, a diferencia de las intolerantes, que si se ven afectadas por estos factores.

Para otorgar servicio a las aplicaciones intolerantes, IntServ necesita conocer el máximo retardo extremo a extremo posible en la comunicación, para aplicar una compensación de retardo fija

mayor o igual a este. Esto permite, a cambio de una pequeña distorsión en la señal, asegurar que los paquetes llegan de manera constante a su destino, disminuyendo así el efecto del jitter.

Por otro lado, las aplicaciones intolerantes emplean un método llamado "servicio predictivo" gracias al cual pueden fijar una compensación menor al retardo máximo posible, basando la compensación en predicciones sobre el futuro retardo de los paquetes.

Este cálculo de los retardos se puede realizar con predicciones más conservativas, las cuales emplean mayores retardos y consumen más recursos. También es posible utilizar predicciones más optimizadas, que nos permiten reducir el retardo extremo a extremo aplicado, reduciendo así el consumo de recursos, pero arriesgando causar más interrupciones en el servicio.

Las aplicaciones tolerantes consumen menos recursos empleando predicciones más optimizadas, y aunque bien es posible que la fidelidad disminuya, causando en ocasiones interrupciones en el servicio, esto no es un gran problema teniendo en cuenta el aprovechamiento de los recursos que esta predicción nos permite realizar. [3]

Desafortunadamente, IntServ es un modelo con mala escalabilidad y alto consumo de recursos en la red, no solo eso, sino que además es complicado de implementar por lo que no es muy utilizado y por tanto no se usará en este proyecto.

2.3.3 DiffServ

Debido a los problemas de IntServ, surge el modelo de servicios diferenciados (DiffServ), un modelo menos exigente y más barato de emplear. DiffServ define una serie de clases de tráfico donde agrupa diversos flujos, todos los flujos dentro de una clase de tráfico son tratados de igual manera. Las clases se definen según la importancia de los datos y se les trata de manera distinta a lo largo de la red. Los paquetes son clasificados y marcados, idealmente, en los bordes de la red y los routers internos solo se preocupan por el PHB (Per Hop Behavior).

En las redes informáticas, el comportamiento por salto (PHB) es un término utilizado en los servicios diferenciados (DiffServ) o la conmutación de etiquetas multiprotocolo (MPLS). Define la política y la prioridad aplicadas a un paquete cuando atraviesa un salto (como un router) en una red DiffServ.

DiffServ emplea Differentiated Service Code Point o DSCP (RFC2474/2475) para marcar los paquetes. DSCP sustituye al campo ToS (Type of Service) previamente definido en los paquetes IPv4 y el campo Traffic Class en IPv6, de los 8 bits de estas cabeceras DSCP emplea los primeros seis bits para marcar el código y deja los otros dos sin usar como podemos ver en la Figura 1.

El campo DSCP se puede escribir en cualquier punto de la red y puede ser modificado de igual forma en cualquier momento con las penalizaciones al rendimiento correspondientes.

Las clases de código DSCP se pueden separar en cuatro categorías: Express Forwarding, Assured Forwarding, Class Signaling y Default Forwarding.

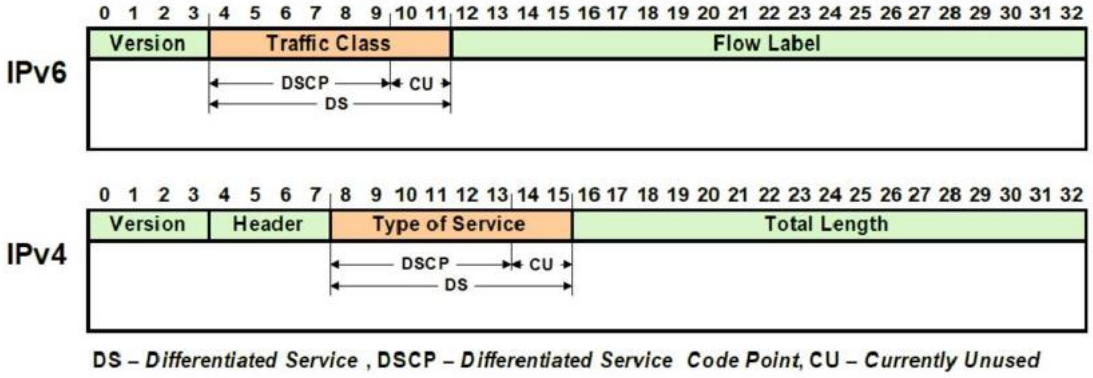

Figura 1. DSCP [4]

Express Forwarding (EF) es el nivel de prioridad más alto posible en una red DiffServ. Es necesario configurar el PHB para priorizar esta clase y evitar así perdidas y retardos perjudiciales a la experiencia del usuario.

Assured Forwarding (AF) es el nivel de prioridad más común en las redes DiffServ. Dentro de AF existen cuatro clases numeradas del 1 al 4 y cada una de estas recibe un segundo número, normalmente llamados grupos olímpicos, recibiendo el nombre oro, plata y bronce, indicando la prioridad de descarte en caso de ser necesario. Los niveles de AF se pueden observar en la Figura 2 (donde oro corresponde a conforming, plata a exceeding y bronce a violating).

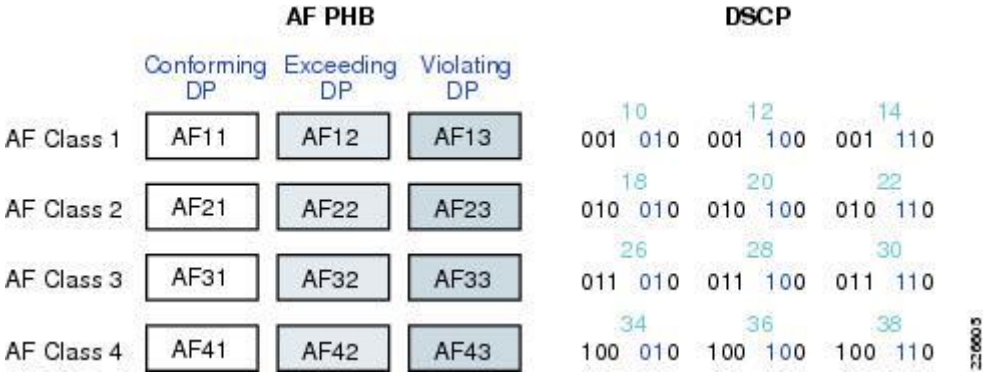

Figura 2. Assured Forwarding [5]

No existe ninguna ventaja inherente entre las cuatro principales clases de AF, depende completamente de cómo se configure la red por lo tanto no hay ningún requisito para pertenecer a estas.

Class Signaling (CS) existe como retrocompatibilidad para las redes que emplean el campo de Type of Service en IP, pero no se van a ver en este proyecto.

Default Forwarding (DF) es la clase que se aplica por defecto a los paquetes y la correspondiente al código 000000 en DSCP y no posee ninguna característica especial.

2.4 Congestión en la red

La red es susceptible de congestión. La congestión ocurre cuando el tráfico en un nodo o en un enlace de la red supera su carga máxima. Esto causa un deterioro de la calidad en la conexión

originando retardos debido a los tiempos en cola, jitter y pérdidas de paquetes. Existen dos maneras de tratar la congestión mediante el uso de mecanismos QoS: a través de "congestion management" y "congestion avoidance". Ambos mecanismos QoS son implementados a través del network scheduler o planificador. El planificador es el encargado de controlar el orden de transmisión de los paquetes (Congestion Management), así como de decidir que paquetes son descartados para evitar la congestión (Congestion Avoidance).

2.4.1 Congestion Management

En redes basadas en IP los paquetes se almacenan en los búferes de memoria hasta que el enlace de salida está libre. Entre los algoritmos más usados para la gestión de cola se incluyen: First In First Out (FIFO), Priority Queuing (PQ), Weighted Fair Queuing (WFQ) y Weighted Round Robin (WRR).

El algoritmo más básico de gestión de colas es FIFO. El principio detrás de FIFO es que todo el tráfico sea tratado de la misma forma ya que no existe manera para ningún paquete de priorizar su paso por la cola. FIFO no proporciona ningún tipo de servicio diferenciado, al aparecer congestión en la cola todo el tráfico se ve afectado por un retardo de colas equivalente a la capacidad de esta, y todos los paquetes que lleguen una vez la cola esté llena serán descartados.

Priority Queuing nos permite priorizar tráfico según la clase a la que este pertenece lo cual podemos usar para brindar cierto nivel de QoS. Si bien esta prioridad es interesante para brindar al tráfico importante de una reducción de retado y jitter, Priority Queuing tiene sus inconvenientes por ejemplo, si el tráfico de alta prioridad es muy elevado, puede provocar que el de menor prioridad sufra grandes pérdidas y retardos.

Weighted Fair Queuing implementa un servicio basado en bits en lugar de paquetes. WFQ sirve a múltiples colas alterando los pesos de estas de manera dinámica lo cual permite proporcionar un servicio completamente "justo" independientemente del tamaño de paquete. Sin embargo, consume muchos recursos y es extremadamente difícil de implementar dada su complejidad.

Weighted Round Robin es un algoritmo que sirve a los diferentes buffers de la cola de manera ordenada y cíclica sirviendo una cantidad de paquetes apropiada para el peso y pasando al siguiente buffer. Es un algoritmo fácil de implementar, pero puede ocasionar retardos y jitter en aplicaciones muy exigentes.

2.4.2 Congestion Avoidance

Existen 2 tipos de algoritmos de gestión para congestion avoidance: pasivos y activos.

Los algoritmos pasivos, aquellos más tradicionales, simplemente otorgan una cantidad máxima de paquetes a cada cola y aceptan todos los paquetes que llegan a la cola hasta que se alcanza el máximo y procede a rechazar todos los que lleguen a continuación hasta que la cola se reduzca en tamaño. Esta técnica se le conoce como "Tail drop".

Por otro lado, los algoritmos activos (AQM) descartan paquetes antes de que la cola se llene con una cierta probabilidad. Emplear AQM causa que menos paquetes sean descartados en caso de que alguna cola se desborde. AQM proporciona mejores resultados en cuanto a jitter y retardo de paquetes. La implementación más básica de algoritmos activos es RED (Random Early Detection) [6].

RED es un algoritmo de gestión de colas para el planificador que elimina de manera preventiva paquetes de la cola. RED monitoriza el tamaño medio de la cola y descarta paquetes basado en probabilidades estadísticas. Si el buffer se encuentra prácticamente vacío se aceptan todos los paquetes y a medida que crece la cantidad de paquetes en cola la probabilidad de descartar algún

paquete aumenta. Cuando la cola se encuentre llena la probabilidad de descartar paquetes habrá tenido que alcanzar 1 y todos los paquetes que lleguen serán descartados.

Existen múltiples algoritmos variantes de RED, la variante más interesante para DiffServ es WRED (Weighted Random Early Detection). WRED permite tener diferentes probabilidades de descarte para diferentes prioridades, mediante DSCP, lo cual facilita la tarea de priorización del tráfico.

2.5 Condicionante de Tráfico

Habitualmente el condicionante de tráfico es un bloque que se encuentra en los extremos de la red DiffServ encargado de clasificar, marcar y modelar el tráfico.

Tal como podemos observar en la Figura 3 los componentes que emplea el condicionante de tráfico para ofrecer QoS se pueden separar en 5 categorías: Classifiers, markers, meter, dropper y shaper.

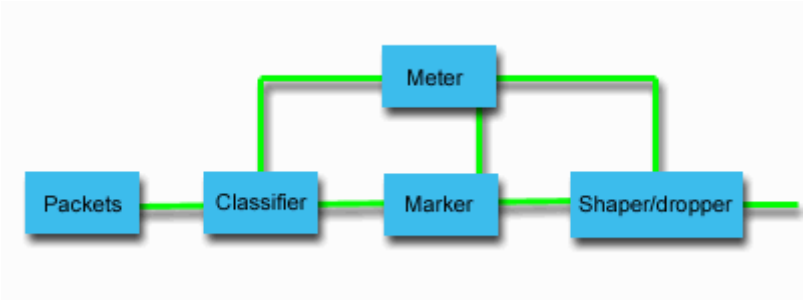

Figura 3 Condicionante de Tráfico [7]

Los classifiers nos permiten separar los paquetes del flujo. Se pueden diferenciar en dos clases: los classifiers BA (Behavior Aggregate) que separan el tráfico en base a su DSCP y los classifiers MF (Multi-Field) que separan el tráfico en función de los campos de la cabecera como pueden ser el origen o destino del paquete o los puertos origen/destino y otro tipo de información como puede ser la interfaz de entrada.

Los clasificadores se utilizan para dirigir los paquetes con unos campos determinados para su posterior procesamiento.

Los markers son los encargados de establecer el campo DS. El marcador puede ser configurado para marcar todos los paquetes que se dirigen a él a un único DS, o puede configurarse para marcar un paquete a uno de un conjunto de DS. Cuando el marcador cambia el DSCP en un paquete se dice que ha "re-marcado" el paquete.

Los meters permiten comprobar si un tráfico se ajusta a una cierta velocidad. En este proyecto se van a observar dos meters distintos: Single Rate Single Bucket Marker (srSBM) y Two Rate Three Color Marker (trTCM).

El Single Rate Single Bucket Marker (srSBM), también conocido como cubo de fichas (Token Bucket), es un mecanismo de control que dicta cuándo se puede transmitir el tráfico, basándose en la presencia de fichas en el cubo, un contenedor abstracto que contiene el tráfico de red agregado que se va a transmitir.

El cubo contiene fichas, cada una de los cuales puede representar una unidad de bytes o un solo paquete de tamaño predeterminado. Mide un flujo de tráfico basándose en dos parámetros: la tasa de información comprometida (CIR) y el tamaño de ráfaga comprometido (CBS).

Los contadores se especifican en términos de cubos de fichas con tamaño CBS y tasa de fichas CIR. Las fichas se generan a la tasa de información comprometida y se añaden al cubo de fichas. Cuando llega un paquete y hay suficientes fichas en el cubo, se considera que el paquete está dentro del perfil y se retira del cubo el número de fichas correspondiente.

Si no hay suficientes fichas en el cubo, el paquete está fuera de perfil. El condicionante de tráfico marca un paquete "verde" cuando está en el perfil y "rojo" cuando está fuera del perfil, tal y como se puede observar en la Figura 4. [8]

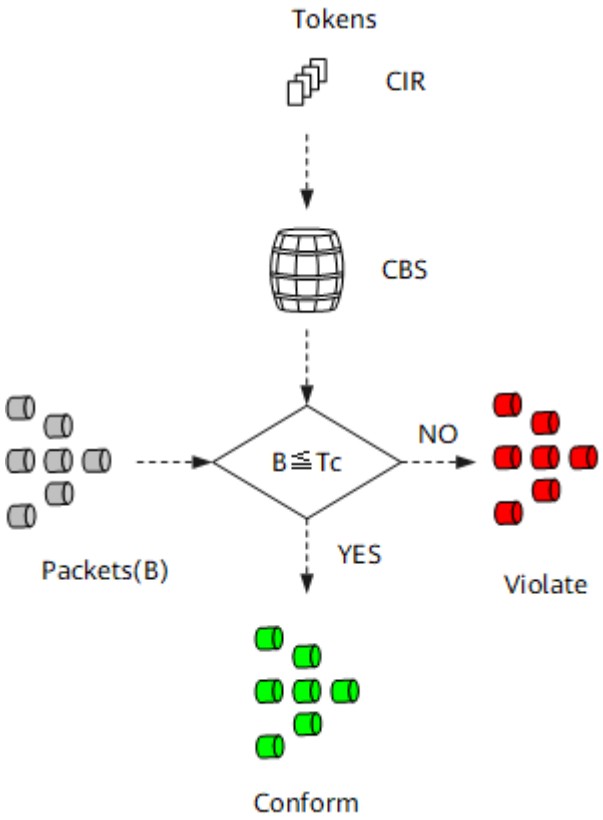

Figura 4. Single Rate Single Bucket Marker [9].

Por otro lado, el Two Rate Three Color Meter emplea cuatro parámetros: tasa de información comprometida (CIR), tamaño de ráfaga comprometida (CBS), tasa de información máxima (PIR) y tamaño de ráfaga máxima (PBS). Estos parámetros corresponden a dos cubos de fichas empleados para realizar la comprobación.

El condicionante de tráfico asigna uno de tres colores (verde, amarillo o rojo) en función de los estados de los dos cubos fichas internos. Un paquete se marca en rojo si supera la PIR. En caso contrario, se marca en amarillo o en verde dependiendo de si supera o no la CIR.

El trTCM es útil en la vigilancia de un servicio, en el que es necesario forzar una tasa de pico por separado de la tasa de información comprometida. El trTCM funciona en uno de los dos modos: modo Color-Blind y modo Color-Aware.

En el modo Color-Blind, el meter predice que el flujo de paquetes no está coloreado. En el modo Color-Aware el meter asume que el flujo de paquetes ha sido precoloreado. El Marcador (re)colorea el paquete según los resultados del meter.

Según, el comportamiento del meter se especifica en términos de dos cubos de fichas, P y C, y su modo con dos tasas PIR y CIR, respectivamente. El tamaño máximo del cubo de fichas P es PBS y el tamaño máximo del cubo de fichas C es CBS.

Los cubos de fichas P y C están inicialmente (en el momento 0) llenos, es decir, el recuento de fichas Tp (0) = PBS y el recuento de fichas Tc (0) = CBS. Posteriormente, el recuento de fichas Tp se incrementa en una PIR por segundo hasta llegar a PBS y el recuento de fichas Tc se incrementa en una CIR por segundo hasta llegar a CBS. Según, cuando llega un paquete de tamaño B bytes en el tiempo t, sucede lo siguiente si el trTCM está configurado para operar en el modo Color-Blind:

- Si 0 > Tp (t) - B, el paquete es rojo.
- Si 0 > Tc (t) - B, el paquete es amarillo y Tp se decrementa en B.
- Si lo anterior no se cumple el paquete es verde y tanto Tp como Tc se decrementan en B.

Cuando llega un paquete de tamaño B bytes en el tiempo t, ocurre lo siguiente si el trTCM está configurado para operar en el modo Color-Aware:

- Si el paquete ha sido precoloreado como rojo o si 0 > Tp (t) - B, el paquete es rojo
- Si el paquete ha sido precoloreado como amarillo o si 0 > Tc (t) - B, el paquete es amarillo y Tp se decrementa en B.
- Si lo anterior no se cumple el paquete es verde y tanto Tp como Tc se decrementan en B.

La implementación real de un meter no necesita ser modelada según la especificación anterior [10]. El funcionamiento de Two Rate Three Color Marker tal y como se ha explicado se puede observar en la Figura 5.

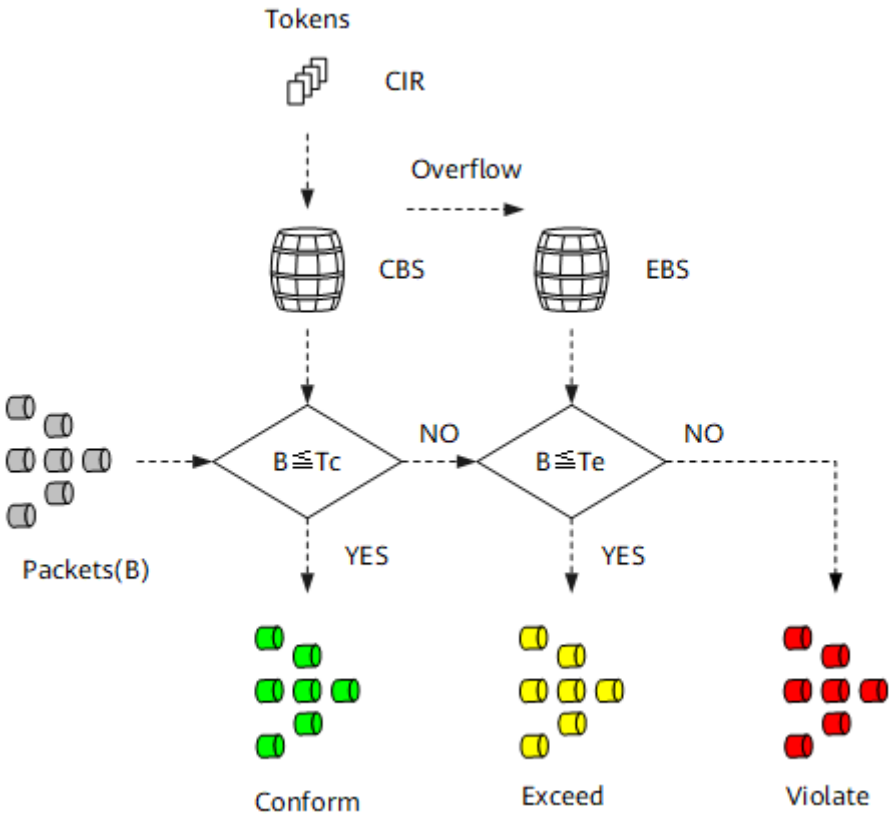

Figura 5 Two Rate Three Color Marker [9].

Lo que diferencia a shappers y droppers es cómo reaccionan ante los paquetes que han sido marcados como "rojos". Los shappers introducen un retardo a los paquetes empleando un buffer de tamaño limitado para evitar que sean destruidos, mientras que los droppers simplemente los eliminan, este segundo caso también es conocido como "policing".

3. Entorno de Simulación

3.1 Omnet++

Omnet++ es un entorno de simulación modular de eventos discretos enfocado a las redes. La creación de modelos en Omnet++ se basa en el uso de módulos reutilizables. Estos módulos se conectan entre si a través de gates, similar a los puertos de cualquier otro sistema, y pueden ser combinados para generar módulos complejos. Los módulos pueden pasar mensajes a lo largo de rutas predefinidas a través de puertas y conexiones, o directamente a su destino (Wireless). Los módulos pueden tener parámetros que pueden utilizarse para personalizar el comportamiento del módulo y/o para parametrizar la topología del modelo. Los módulos del nivel más bajo de la jerarquía de módulos se denominan módulos simples, son programados en C++ y encapsulan el comportamiento del modelo.

Los módulos simples se pueden combinar en módulos compuestos y así sucesivamente. El número de niveles de jerarquía es ilimitado. El modelo completo como podemos verlo en la Figura 6, llamado red en Omnet++, es en sí mismo un módulo compuesto.

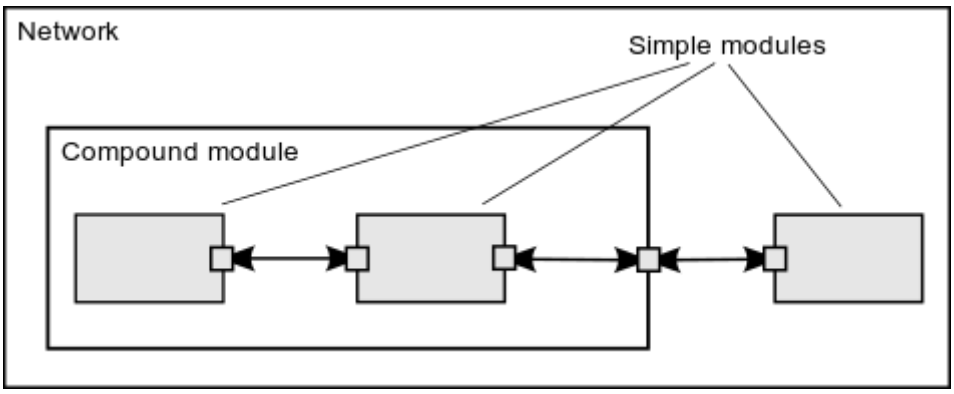

Figura 6 Modelo Omnet++ [11]

La transmisión de mensajes se realiza a través de puertas (gates), las cuales actúan de interfaces de entrada y salida de los módulos. Las conexiones creadas de esta manera pueden ser configuradas para controlar la velocidad del enlace, el retardo y la tasa de error, y de esta manera crear una simulación más realista. Las conexiones se crean dentro de un solo nivel de la jerarquía, no se permiten las conexiones que atraviesan los niveles de la jerarquía, ya que dificultarían la reutilización del modelo. También se pueden definir tipos de conexión con propiedades específicas (denominados canales) y reutilizarlos en varios lugares.

Los parámetros se pueden utilizar también para modificar el comportamiento de los módulos, y para parametrizar la topología del modelo (ej. Numero de hosts parametrizable). Los parámetros con pueden utilizarse para construir topologías flexibles. Dentro de un módulo compuesto, los parámetros pueden definir el número de submódulos, el número de puertas y la forma en que se realizan las conexiones.

El archivo de descripción de red (.ned), es el archivo donde el usuario describe la estructura de la red a modelar en lenguaje NED. NED proviene de las siglas de Network Description (descripción de la red). El NED permite la instanciación de módulos simples de Omnet++ para así formar módulos complejos que más tarde el usuario puede definir como redes. Los archivos NED tienen niveles jerárquicos, lo cual nos permite emplear diseños complejos creados en otro archivo NED y reutilizarlos las veces que veamos necesarias.

El lenguaje NED soporta herencia mediante el uso de **extend** para los elementos del mismo tipo existiendo una única excepción a esta regla y esa es que los módulos simples no pueden hacer **extend** de módulos complejos.

La configuración y los datos de entrada para la simulación se encuentran en un archivo de configuración que suele llamarse omnetpp.ini. Este archivo se agrupa en secciones cada una de ellas pertenecientes a un escenario de simulación, siempre existiendo una sección llamada [General] donde se especifican los parámetros comunes a todas las redes, algunos parámetros de la configuración solo se pueden asignar aquí. Al resto de secciones se les llama [Config <nombre de escenario>], una vez se ejecuta la simulación es posible seleccionar cuál de las secciones se quiere ejecutar, la cual se ejecutará junto a [General]. También es posible especificar si se quiere que otras secciones se ejecuten mediante el uso de **extends = <configname>**, lo cual nos permite reutilizar código. Podemos extender de tantas otras secciones como queramos.

Las simulaciones reciben información a través de los parámetros del módulo, a los cuales se les puede asignar un valor en los archivos NED o en omnetpp.ini, en este orden. Los parámetros asignados en los archivos NED no pueden ser anulados en omnetpp.ini. Por el contrario, es más fácil y flexible mantener la configuración de los parámetros del módulo en omnetpp.ini.

Para evitar ejecutar múltiples veces una simulación con diferentes ajustes de parámetros Omnet++ permite especificar iteraciones sobre ajustes de parámetros empleando ${…} como en el ejemplo que podemos ver en la Figura 7.

```
[Config AlohaStudy]
*.numHosts = ${1, 2, 5, 10..50 step 10}
**.host[*].generationInterval = exponential(${0.2, 0.4, 0.6}s)
```

Figura 7. Iteraciones de parámetros

Este ejemplo ejecutará la simulación un total de 24 veces, una por cada combinación posible de parámetros, sin la necesidad de emplear scripts externos.

También es posible ejecutar iteraciones paralelas empleando una exclamación y otra variable de iteración.

Como podemos ver en el ejemplo de la Figura 8, el único bucle que existe está definido por la primera línea, la variable plan. Las otras dos iteraciones, hosts y load, simplemente la siguen; para el primer valor de plan se seleccionan los primeros valores de host y load, y así sucesivamente.

```
**.plan =     ${plan= "A", "B", "C", "D"}
**.numHosts = ${hosts= 10,  20,  50, 100 ! plan}
**.load =     ${load= 0.2, 0.3, 0.3, 0.4 ! plan}
```

Figura 8. Iteración paralela

Omnet++ tiene soporte para registrar los resultados de las simulaciones empleando vectores y escalares de salida. Los vectores de salida son datos temporales obtenidos de los módulos o canales. Se pueden emplear para obtener valores como el retardo extremo a extremo o el throughput. Los escalares por otro lado son resultados calculados durante la simulación y escritos cuando esta termina. Un escalar puede ser un numero entero o un resumen estadístico, se puede emplear para obtener la tasa de pérdida de paquetes. Omnet++ genera un archivo para cada uno de estos dos tipos de datos. [11]

3.2 INET

INET Framework es una librería open-source para Omnet++. INET soporta redes alámbricas, inalámbricas, móviles, ad hoc y de sensores.

Incluye modelos para la pila de Internet (TCP, UDP, IPv4, OSPF, etc.), protocolos de capa de enlace (Ethernet, PPP, etc.), soporte para la capa física inalámbrica, protocolos de enrutamiento MANET, DiffServ, MPLS con señalización LDP y RSVP-TE, modelos de aplicación y muchos otros protocolos y componentes. También ofrece soporte para la movilidad de nodos, visualización avanzada, emulación de redes y mucho más. [12]

INET contiene una librería que proporciona componentes como colas, generadores de tráfico y condicionantes de tráfico.

La mayoría de los módulos INET tienen configuradas múltiples declaraciones para recoger estadísticas que son automáticamente guardadas al simular. Es posible añadir estadísticas en nuevos módulos derivados de aquellos que importas de INET.

4. Desarrollo

En este proyecto usaremos el programa Omnet++ explicado en el punto 3 para simular el comportamiento de una red simple para analizar las características de rendimiento de la red y así determinar el impacto de los condicionantes de tráfico QoS y los algoritmos de colas sobre la misma.

Para ello primero se configurarán, en la sección 4.1, los módulos que se emplearán en la simulación: módulo de condicionante de tráfico, módulo de la red, módulo de las redes LAN que componen la red y el módulo de la cola que se empleará con la correspondiente configuración del planificador.

Mas adelante, en la sección 4.2, se especificarán los escenarios empleados para simular la red y su configuración.

Por último, en la sección 4.3, se observarán los resultados obtenidos de la simulación y se analizará el efecto que han tenido los mecanismos de QoS sobre la red.

4.1 Configuración de la red

4.1.1 Modulo TC

Este módulo introduce el condicionante de tráfico, marcando y midiendo el tráfico que entra en él y tratando con el tráfico que no se conforme. En este proyecto el condicionante de tráfico emplea policing por lo que descartará todos aquellos paquetes que superen la velocidad asignada, sea esta la de conformidad o la de exceso en caso de que esta exista.

Los módulos routers contenidos en INET poseen slots para condicionante de tráfico en Egress y condicionador de tráfico en Ingress en los interfaces ppp y ethernet. En nuestro caso los routers se encuentran conectados a través de ppp por lo que introduciremos este módulo en la interfaz ppp.

Este módulo se empleará como un condicionante Egress a la salida de las interfaces de los clientes, ya que se tiene un mayor control de QoS al aplicar las medidas de control en el tráfico de salida al disponer de la opción de emplear colas a diferencia del tráfico entrante.

El módulo lo podemos observar a continuación en la Figura 9.

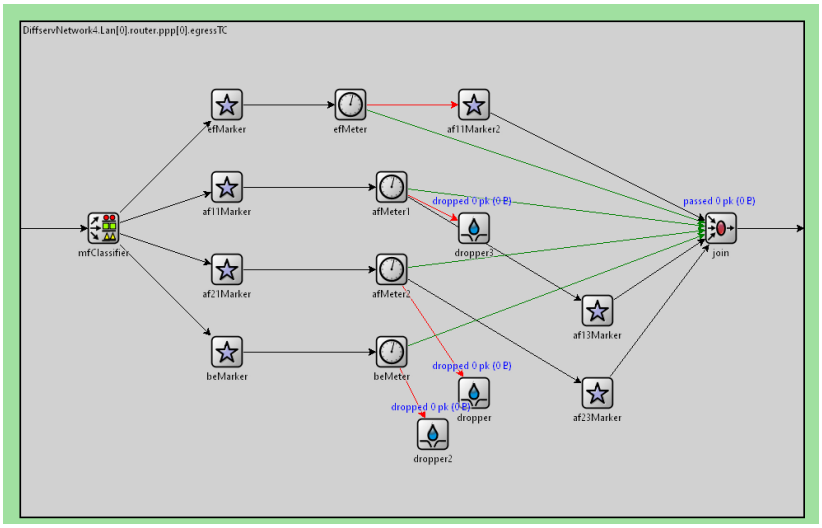

Figura 9. Modulo Traffic Conditioner

INET nos ofrece dos distintos módulos para el componente meter, en nuestro caso emplearemos TokenBucketMeter para tráfico EF y BE y para ambos AF emplearemos TwoRateThreeColorMeter.

El módulo TokenBucketMeter nos permite parametrizar CIR y CBS y enviar todo el tráfico que se conforme a estos parámetros de velocidad por la salida "verde" y todo aquello que no se conforme a los parámetros a la salida "roja" donde se toman las medidas necesarias para tratar con el tráfico, en nuestro caso descartar los paquetes para cualquier clase menos EF donde disminuiremos la prioridad.

Por otro lado, TwoRateThreeColorMeter nos permite definir CIR y CBS como el anterior módulo, así como PIR y PBS, lo cual nos otorga una salida más para emplear en los tráficos AF donde si el tráfico se conforma saldrá por "verde", si se encuentra en exceso, pero no lo supera saldrá por la salida "amarilla" donde disminuiremos su prioridad, y en caso de superar el exceso saldrán por "rojo" donde se descartarán los paquetes.

Los módulos de meters en INET aceptan parámetros de velocidad en función de porcentajes de velocidad de la línea.

Para diferenciar las clases el condicionante de tráfico emplea un filtro en el clasificador cuyos parámetros se encuentran definidos en el archivo filters.xml

En el clasificador aquel tráfico perteneciente a la gate 0 será marcado como EF, el perteneciente a gate 1 como AF11 y el perteneciente a la gate 2 como AF21. Todos aquellos no declarados en el filtro de la Figura 10, se considerarán tráfico perteneciente al Default Forwarding y por tanto se les tratara como BE.

```
<filters>
  <filter destAddress="server" protocol="udp" destPort="5003" gate="0"/>
  <filter destAddress="server" protocol="udp" destPort="5001" gate="1"/>
  <filter destAddress="server" protocol="udp" destPort="5000" gate="2"/>
</filters>
```

Figura 10. Filtro TC

4.1.2 Modulo DiffServQueue

El módulo DiffServQueue es un sistema de colas de ejemplo ofrecida por INET que se puede utilizar en las interfaces de los nodos de una red DiffServ. Este módulo nos permite almacenar los paquetes EF en una cola dedicado y priorizar sus paquetes. Ya que EF tiene prioridad sobre los otros tráficos es necesario limitar su ancho de banda, para ello se ha añadido un TokenBucket a la entrada de cola EF que descarta cualquier paquete que no se conforme a cierta velocidad.

Como podemos observar en la Figura 11, dentro del módulo el tráfico es separado en distintas colas según la clase a la que pertenece, el tráfico EF posee prioridad sobre el resto del tráfico, mientras que el resto de las clases se distribuyen mediante Weighted Round Robin.

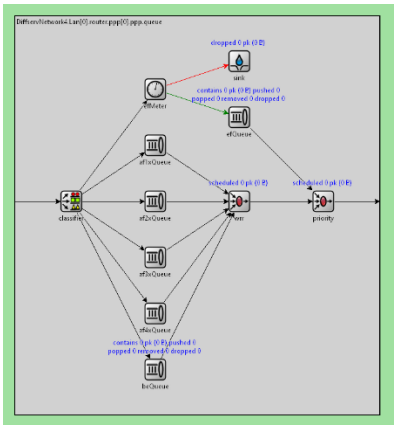

Figura 11. Modulo DiffServQueue

Dentro de cada una de las colas de las clases AF se encuentran caminos separados para diferenciar las clases de prioridad dentro de cada clase, junto con un dropper RED en cada camino, siendo este más estricto para las clases de menor prioridad. Tal y como podemos observar en la Figura 12.

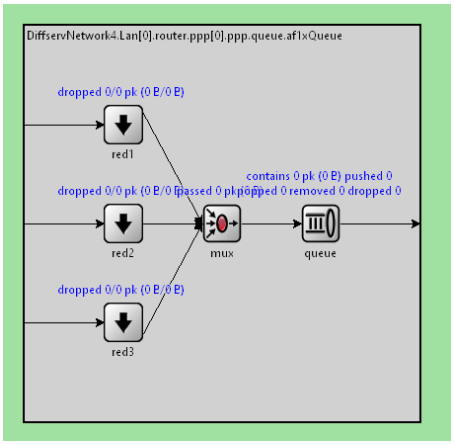

Figura 12. AFxx Queue

Por otro lado, la cola de BE es una cola DropTail sin más, carece de cualquier tipo de QoS.

4.1.3 Modulo LAN

El módulo consiste en un numero parametrizable de clientes, todos ellos conectados a un conmutador ethernet el cual esta a su vez conectado con un router con conexión al exterior del módulo. Las líneas empleadas en el módulo se han creado a base de herencia a partir de líneas DatarateChannel de INET, el cual nos permite definir los parámetros de retardo, velocidad del enlace, y la tasa de error. El ser un canal que usa como base DatarateChannel también implica que las estadísticas definidas en el mismo serán recogidas por los canales que creemos.

Este módulo LAN, observado en la Figura 13, es el módulo que emplearemos para generar el tráfico de red, por tanto, el router que se encuentra dentro del mismo estará encargado de regular tráfico saliente que como ya mencionamos antes es el más útil a la hora de aplicar QoS a nuestra red.

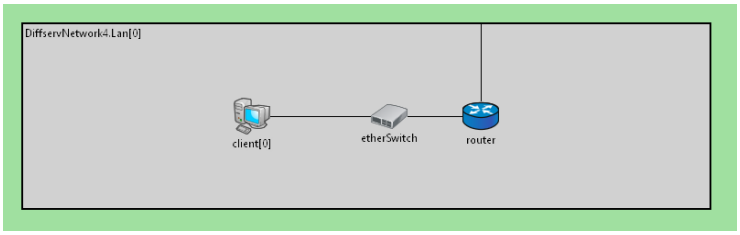

Figura 13. LAN

4.1.4 Modulo RedDiffServ

El módulo principal de este proyecto en el que se describe la red sobre la que se realizan pruebas, en otras palabras, la network o red. El módulo puede ser observado en la Figura 14. Como podemos observar esta red está compuesta por un numero parametrizable de LANs que como hemos visto más arriba están conectadas al exterior mediante un router, este router se conecta a un segundo router al salir de la LAN el cual a su vez conecta con módulo host de INET que emplearemos para recibir la transmisión de paquetes en las simulaciones.

Figura 14. Red

Las líneas una vez más se han creado a base de herencia a partir de líneas DatarateChannel de INET, lo cual nos permite una vez más definir el retardo, la velocidad del enlace y la tasa de error, así como recolectar estadísticas como puede ser la utilización del enlace o el throughput al igual que ya vimos en el módulo LAN.

Las líneas dentro de la LAN se han configurado con velocidades de 100 Mbps, mientras que la línea que une a la LAN con el router y el mismo con “server” poseen 30 Mbps. Esto causa que el cuello de botella suceda en la interfaz de salida de LAN. Esto lo hacemos porque es una mejor opción configurar la QoS de una red en tráficos salientes, el control de tráficos entrantes es posibles, pero mucho más limitado, dado que no se pueden emplear colas o buffers.

4.2 Configuración de escenarios

Como se comentó en el apartado 3 al describir el funcionamiento de Omnet++, el encargado de configurar los parámetros de simulación que emplearemos en este proyecto es el archivo omnetpp.ini, en él nos encontraremos definidos los diferentes escenarios, así como una configuración conocida como “[General]” común a todos los escenarios.

Para comprobar la eficacia de DiffServ se testeará la red en y tres escenarios. El primer escenario sin QoS, un segundo escenario con él condiciona condicionante de tráfico con policing estricto, y un tercer y último escenario donde a parte del condicionante de tráfico también se emplea la cola de DiffServQueue.

4.2.1 General (sin QoS)

En este primer escenario de la configuración del proyecto se especifica la duración de la simulación, el tráfico generado en la red, así como el tamaño de la cola empleada por defecto.

Para el tráfico se emplean cuatro aplicaciones configuradas como generadores de tráfico UDP con dirección de destino "server" y puertos destino diferenciados para emplearlos como parámetros a la hora de diferenciar las clases de tráfico.

En cuanto a la velocidad de las aplicaciones generadas, el tráfico AF1 y AF2 han sido configurado para trabajar a 12 Mbps cada uno, el tráfico EF se ha configurado a 4 Mbps, y el tráfico de baja prioridad BE se encuentra a 24 Mbps. También se detiene el flujo del tráfico AF1 para así evaluar más adelante la flexibilidad de la red mediante el throughput.

Tanto el número de LANs como el de clientes se ha dejado en tan solo 1 ya que no son necesarios más.

Por otro lado, se han configurado cuatro aplicaciones en el servidor que actúan de sumideros para los paquetes enviados desde los clientes en los diferentes puertos.

El modelo host guarda estadísticas sobre los paquetes enviados por sus aplicaciones, por lo que podemos comparar el número de paquetes enviados en las aplicaciones del cliente y compararlos con los de las aplicaciones del "server" para así hallar la tasa de pérdida de paquetes.

También se ha configurado la cola de la interfaz ppp para aumentar su capacidad hasta los 300 paquetes, lo cual será útil más adelante en otros escenarios.

Esta configuración se puede observar a continuación en la Figura 15.

```
[General]
network = RedDiffServ
sim-time-limit = 10s

# Clientes
**.numClients = 1
**.numLans = 1
**.client[*].numApps = 4
**.client[*].app[*].typename = "UdpBasicApp"
**.client[*].app[*].destAddresses = "server"

# App 0 - AF1
**.client[*].app[0].destPort = 5001           # AF1
**.client[*].app[0].packetName = "AF1"
**.client[*].app[0].startTime = 1s
**.client[*].app[0].stopTime = 3s
# 12Mbps
**.client[*].app[0].messageLength = 900B
**.client[*].app[0].sendInterval = normal(600us,10us)

# App 1 - AF2
**.client[*].app[1].destPort = 5000        # AF2
**.client[*].app[1].packetName = "AF2"
**.client[*].app[1].startTime = 1s
# 12 Mbps
**.client[*].app[1].messageLength = 900B
**.client[*].app[1].sendInterval = normal(600us,10us)

# App 2 - Best Effort
**.client[*].app[2].destPort = 5002           # BE
**.client[*].app[2].packetName = "BE"
```

```
**.client[*].app[2].startTime = 1s
# 24 Mbps
**.client[*].app[2].messageLength = 900B
**.client[*].app[2].sendInterval = normal(300us,10us)

# App 3 - Video
**.client[*].app[3].destPort = 5003         # EF
**.client[*].app[3].packetName = "video"
**.client[*].app[3].startTime = 1s
# 4 Mbps
**.client[*].app[3].messageLength = 500B
**.client[*].app[3].sendInterval = normal(1ms,10us)

# Server App
**.server.numApps = 4
**.server.app[*].typename = "UdpSink"
**.server.app[0].localPort = 5001        # AF1
**.server.app[1].localPort = 5000        # AF2
**.server.app[2].localPort = 5002        # BE
**.server.app[3].localPort = 5003        # EF

# Router queue
**.router.ppp[*].ppp.queue.typename = "DropTailQueue"
**.router.ppp[*].ppp.queue.packetCapacity = 300
```

Figura 15. Configuración general

4.2.2 Policing estricto

A la hora de aplicar policing al condicionante de tráfico se ha introducido el módulo TC dentro de la interfaz de salida ppp del router de la LAN. Dentro del módulo se han cambiado los parámetros de CIR y CBS de los meters para garantizar que cualquier tráfico que supere ciertas velocidades sea tratado como debe. Estos cambios se pueden observar en la Figura 16.

```
[Config Policing]
description = "Diffserv traffic conditioning is turned on in the ppp[0]
interface of the router"
**.router.ppp[0].egressTC.typename = "TrafficConditioner"
**.router.ppp[0].ppp.queue.interfaceTableModule = "^.^.^.interfaceTable"

#efMeter
**.efMeter.cir = "30%"
**.efMeter.cbs = 141kB
#afMeter1
**.afMeter1.cir = "30%"
**.afMeter1.cbs = 141kB
**.afMeter1.pir = "30%"
**.afMeter1.pbs = 141kB
#afMeter2
**.afMeter2.cir = "30%"
**.afMeter2.cbs = 141kB
**.afMeter2.pir = "30%"
**.afMeter2.pbs = 141kB
#beMeter
**.beMeter.cir = "10%"
**.beMeter.cbs = 15kB
```

Figura 16. Configuración adicional del segundo escenario

Los meters del módulo han sido configurados con los valores que podemos ver en la Tabla 1 en el escenario de policing estricto.

	CIR	CBS	PIR	PBS
EF	30%	141kB	-	-
AF11	30%	141kB	30%	141kB
AF21	30%	141kB	30%	141kB
BE	10%	15kB	-	-

Tabla 1. Meters estricto

La configuración estricta de estos meters nos permite establecer un límite a las velocidades de los distintos tráficos, gracias a lo cual podemos garantizar que nunca ocurra la congestión en nuestra red. Esto requiere que los paquetes EF no inunden la red, ya que en el condicionante de tráfico les damos tratamiento preferencial y en lugar de eliminarlos les reducimos la prioridad una vez superan su CIR.

4.2.3 Policing y Queueing

El escenario de Policing y Queueing nos sirve para ampliar el anterior escenario de simulación con policing, introduciendo una cola de DiffServ tras el condicionante de tráfico, que divide las colas por clase y le confiere prioridad al paso del tráfico EF para reducir el retardo de cola y prevenir las pérdidas, como podemos observar en la Figura 17.

```
[Config Queueing]
description = "A DiffservQueue is added to the ppp[0] interface of the
router"
**.router.ppp[*].ppp.queue.typename = "DiffservQueue"
**.router.ppp[0].ppp.queue.interfaceTableModule = "^.^.^.interfaceTable"
**.router.ppp[*].ppp.queue.efMeter.cir = "30%" # reserved bandwidth for EF
packets
**.router.ppp[*].ppp.queue.efMeter.cbs = 141kB

**.router.ppp[*].ppp.queue.wrr.weights = "6 6 6 6 1"
```

Figura 17. Configuración adicional del segundo escenario

En la configuración Queueing establecemos los parámetros que empleara la DiffServQueue, la cual se sitúa a la salida del condicionante de tráfico. Los pesos establecidos para el Weighted Round Robin nos permite controlar la distribución del ancho de banda que no emplea el flujo EF.

La configuración del escenario a simular, tal y como se observa en la Figura 18, extiende de las configuraciones descritas anteriormente, heredando sus parámetros y sobrescribe el PIR y PBS para los flujos AF con la intención de evitar el desaproveche de la línea.

```
[Config WithPolicingAndQueueing]
**.afMeter1.pir = "60%"
**.afMeter1.pbs = 282kB
**.afMeter2.pir = "60%"
```

```
**.afMeter2.pbs = 282kB

extends = WithPolicing, WithQueueing
```

Figura 18. Configuración adicional del tercer escenario

Los meters del módulo han sido configurados con los valores que podemos ver en la Tabla 2 para este escenario.

	CIR	CBS	PIR	PBS
EF	30%	141kB	-	-
AF11	30%	141kB	60%	282kB
AF21	30%	141kB	60%	282kB
BE	10%	15kB	-	-

Tabla 2. Meters No estricto

Esta configuración más laxa de los flujos nos permite enviar tráfico AF por encima del 30% asignado inicialmente bajando la prioridad de todo aquel que supere la conformidad, otorgando así una mayor flexibilidad a nuestros flujos.

4.3 Análisis de resultados

Estos resultados se obtienen a partir de los escenarios descritos en el apartado anterior.

Los módulos de INET poseen múltiples parámetros de recopilación de estadísticas preparados para permitirnos analizar la simulación. En este proyecto se observarán los siguientes:

- El retardo extremo a extremo de la red de los flujos. Del mismo podremos obtener el jitter en base a la desviación estándar del retardo. Ambos parámetros importantes para aplicaciones con ciertos requerimientos de calidad.
- El throughput de los flujos a lo largo del tiempo. Con esta estadística podremos observar el aprovechamiento de la red.
- La pérdida de paquetes de los diferentes flujos. Nos permite percibir posibles pérdidas de paquetes dañinas para las aplicaciones.

4.3.1 Sin QoS

En primer lugar, observaremos el comportamiento de una red sin QoS para observar el devastador efecto que tiene la congestión sobre el tráfico. Estas redes suelen generar problemas ya que el tratamiento equitativo de los flujos causa retardos y pérdidas en todos los flujos, lo cual es especialmente dañino en aplicaciones sensibles.

No solo eso, sino que al usar una cola con Tail Drop, las aplicaciones que envíen un mayor flujo de datos tendrán una mayor probabilidad de entrar en ella. Esto causa que los flujos más pequeños, que en el caso de esta simulación es el tráfico sensible sufran una mayor cantidad de pérdidas.

El retardo extremo a extremo que introduce esta situación en nuestro tráfico, como podemos observar en la Figura 19, surge por el tiempo de espera en la cola, y por tanto es directamente proporcional al tamaño de esta.

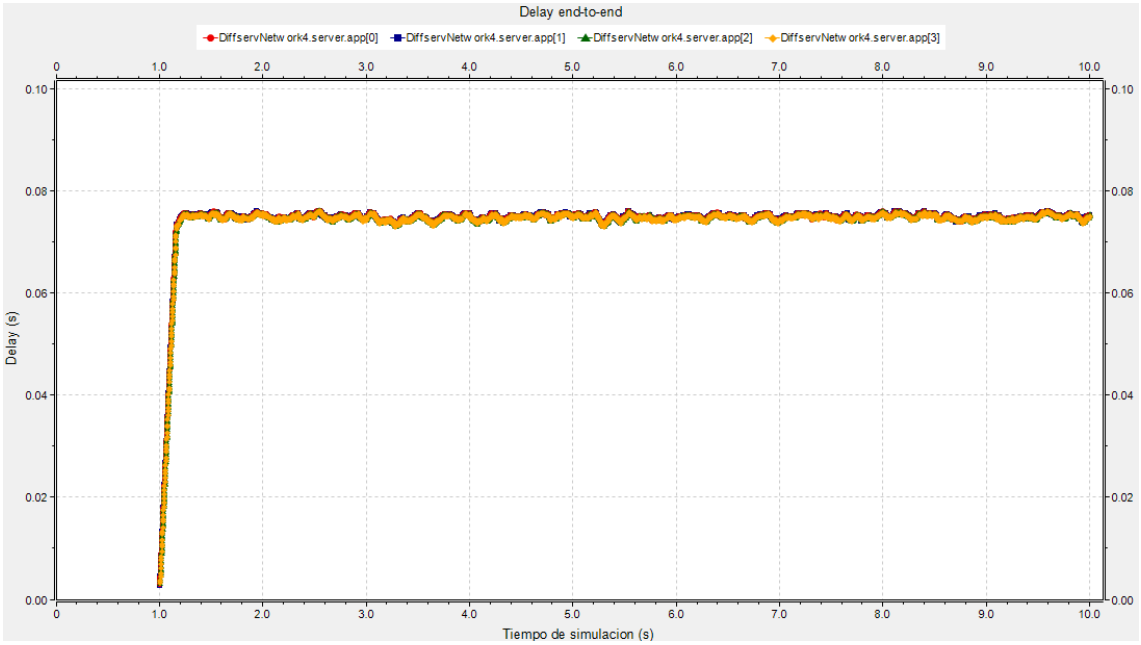

Figura 19. Retardo sin QoS

Ya que esta simulación no posee QoS el tráfico recibe todo el mismo tratamiento así que la aplicación que más datos envía acaba acaparando la gran mayoría de la banda, lo cual puede causar que aplicaciones poco importantes se interpongan en el funcionamiento correcto de aplicaciones que requieran una conexión estable, como podemos observar en la Figura 20. Como se puede ver una vez se detiene el flujo de AF1x los flujos restantes vuelven a acaparar el ancho de banda que pueden sin miramientos por aquellos tráficos que requieren de este ancho de banda.

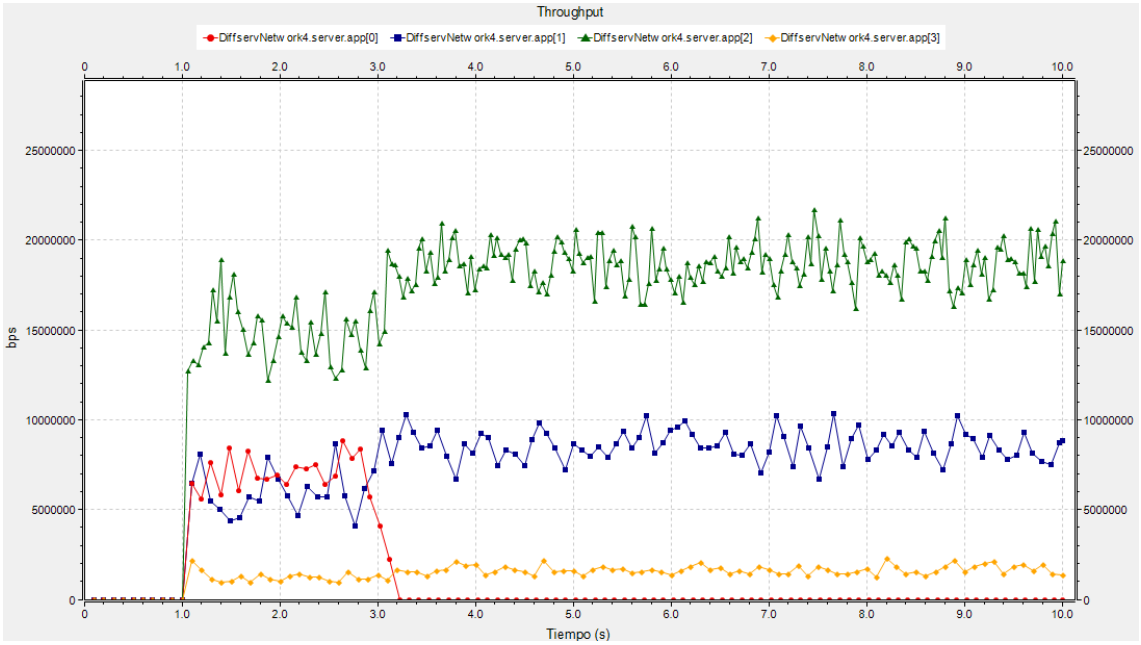

Figura 20. Throughput sin QoS

Como hemos mencionado anteriormente los flujos de mayor volumen, en nuestro caso el tráfico BE consiguen acaparar más ancho de banda, por lo cual sufren menores pérdidas. Una vez más

podemos observar como el tratamiento equitativo puede resultar dañino para nuestras aplicaciones sensibles en la Figura 21.

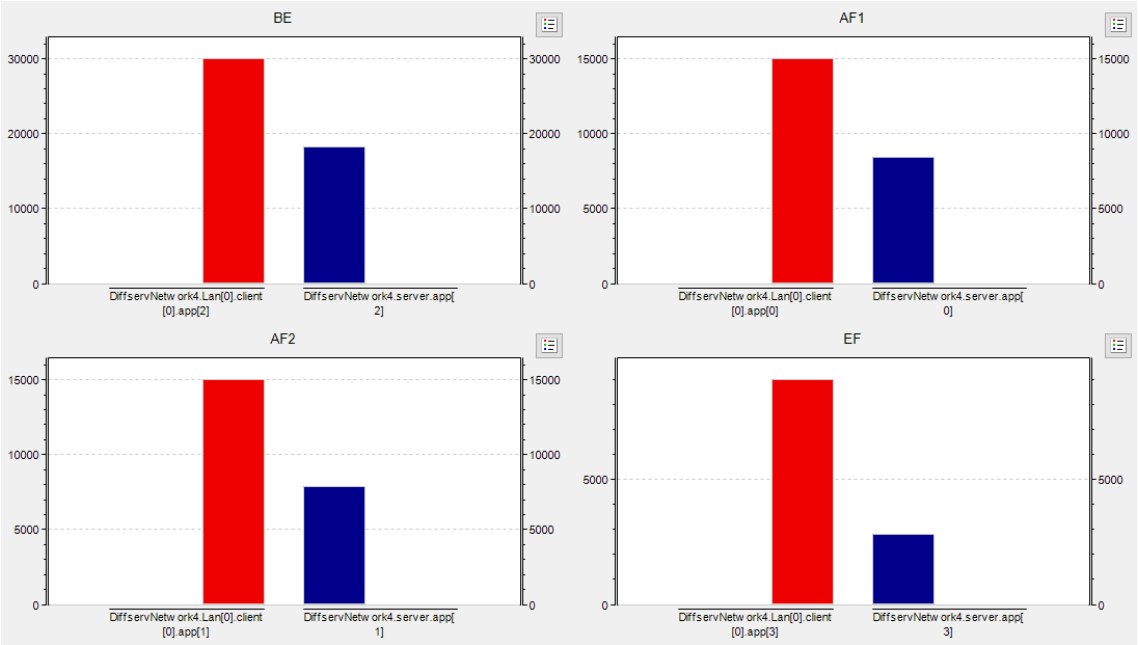

Figura 21. Packet Loss sin QoS

Como se puede observar los flujos con un trafico mas elevado, en este caso BE (24Mbps) y los AF (12Mbps), sufren una menor perdida de paquetes, y obtienen un mayor throughput que el tráfico mas reducido, en este caso EF (4Mbps)

4.3.2 Con Policing estricto

Policing nos permite limitar las velocidades de las diferentes clases, con lo cual podemos asegurar que siempre tenga suficiente espacio el tráfico importante. Sin embargo, para lograr esto es necesario otorgar unos límites de velocidad estrictos. Para ello, necesitamos prescindir de los límites de exceso (EIR) en AF en nuestro caso, para así evitar que la congestión llegue a suceder en la cola a toda costa. Estas velocidades estrictas implican un posible desaprovechamiento de la línea fuera de congestión, ya que no pueden usar ese ancho de banda libre los otros tráficos.

El uso de policing causa que aquellos paquetes que no se ajusten a la velocidad asignada sean eliminados, estos paquetes “no tienen” un retardo pues nunca llegan a su destino tal y como se puede observar en la Figura 22 por lo que no son contabilizados para los cálculos de retardo extremo a extremo. Esto causa que los paquetes que lleguen al otro lado de la red lo hagan con un retardo de colas mínimo. En otras palabras, policing elimina aquellos paquetes que vayan a congestionar la red por lo que nunca llega a estar congestionada y podemos garantizar un retardo de colas mínimo en todos los flujos.

El retardo inicial observado en la Figura 22 durante alrededor medio segundo se debe al efecto del tamaño de ráfaga comprometido (CBS), es posible mitigarlo reduciendo el mismo.

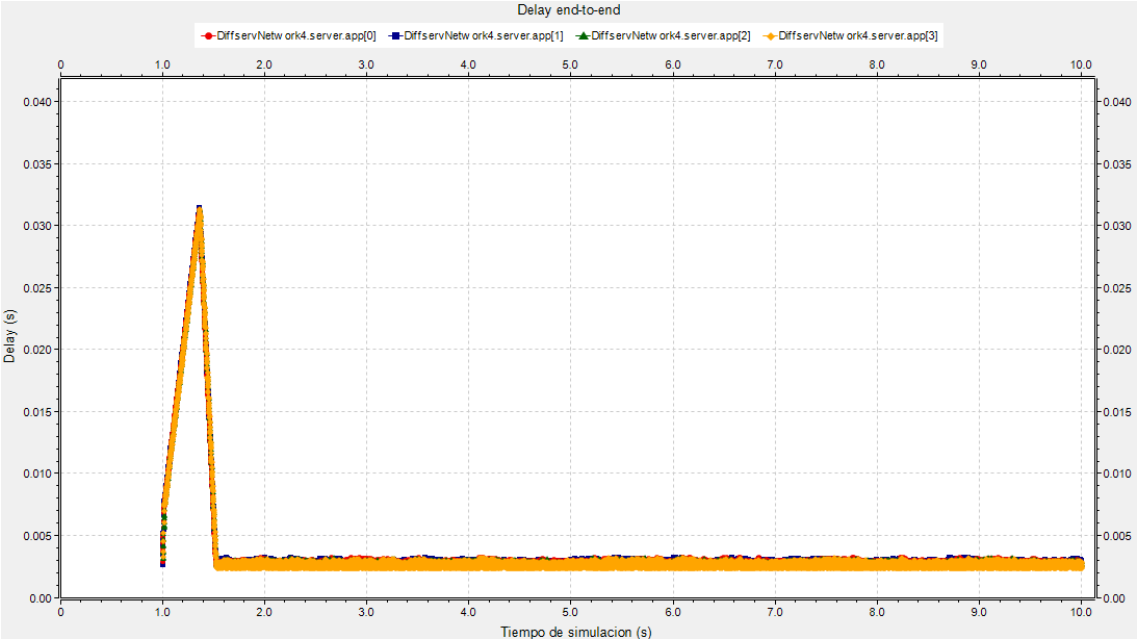

Figura 22. Retardo con policing

El throughput en un caso de policing estricto será en todos los flujos igual o menor a la velocidad establecida en el CIR. Por tanto, la red nunca entrara en congestión evitando como antes hemos podido ver cualquier posible retardo de colas en la red. Sin embargo, esto lo conseguimos a cambio de arrebatar al tráfico su flexibilidad arriesgando desaprovechar la línea como podemos ver una vez detenemos el flujo de paquetes de la clase AF1x en la Figura 23. Una vez se detiene el flujo el resto de los paquetes sigue enviando a la velocidad máxima establecida en su meter ya pues es imposible para estos superarla, aunque las aplicaciones que empleamos están generando paquetes a una tasa superior a su CIR.

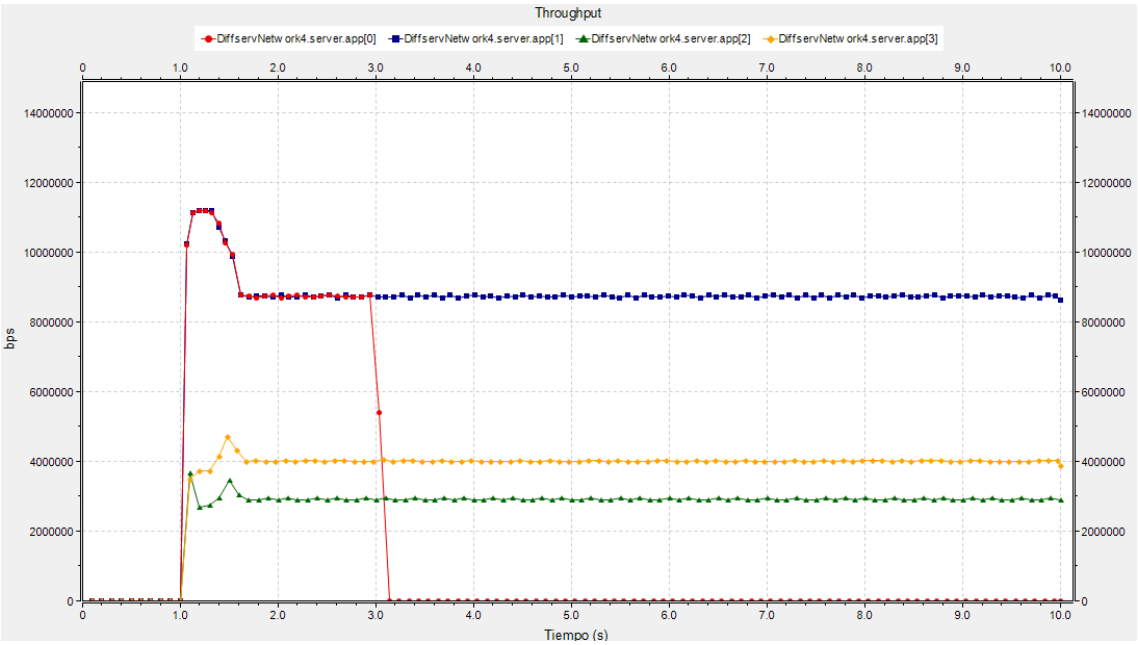

Figura 23. Throughput con Policing

La pérdida de paquetes viene definida únicamente en la diferencia entre la velocidad a la que intenta enviar la aplicación y aquella que tiene asignada por el condicionante de tráfico, siempre y cuando se mantenga por debajo de la velocidad asignada no perderá ningún paquete, tal y como se puede observar en la Figura 24.

El problema como habíamos visto con esto es el desaprovechamiento de la línea, si bien no aparece congestión en nuestra red, es posible para los flujos no importantes perder paquetes cuando existe un ancho de banda libre.

Lo único que podría causar una pérdida de paquetes EF en nuestro modelo sería un aumento por encima de la velocidad designada de la aplicación critica en EF, ya pues nunca descartamos esos paquetes dada su urgencia y en su lugar pasamos la carga a la cola AF11. Si se envía tanto tráfico EF como para causar que la cola entre en congestión, entonces y solo entonces, se perderán paquetes de EF.

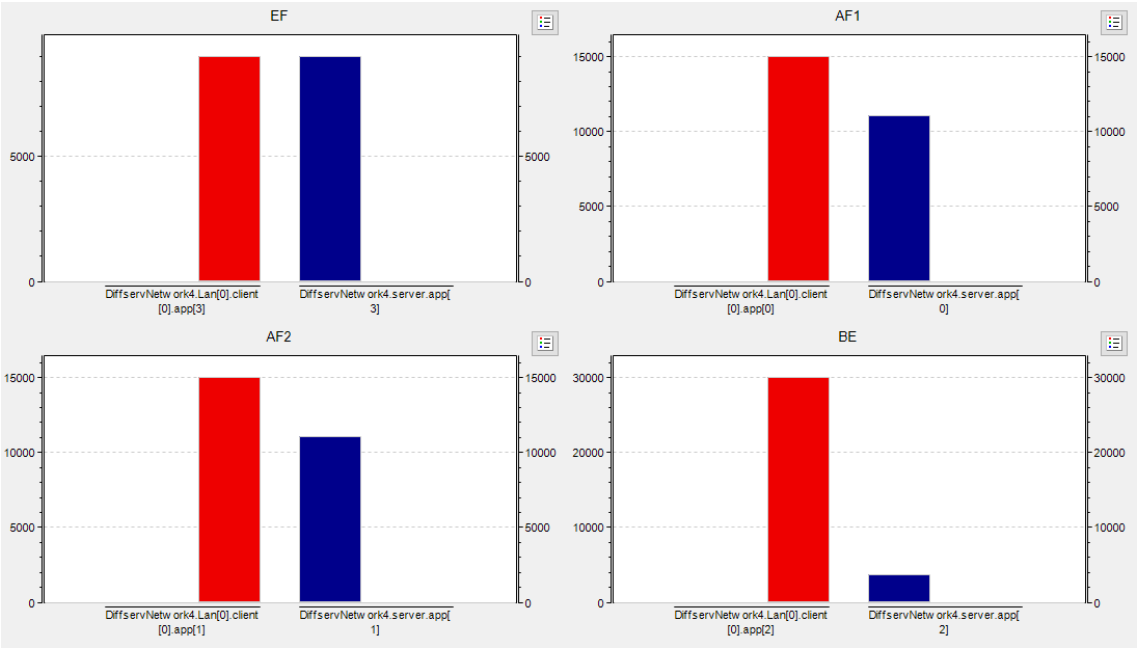

Figura 24. Packet Loss con Policing

Como se puede observar a diferencia de la anterior simulación hay una cantidad nula de perdidas en EF, mientras que BE sufre grandes perdidas dado el poco ancho de banda que tiene asignado en el condicionante de tráfico.

4.3.3 Con Policing y cola

En este escenario podremos observar cómo añadir una cola nos permite ser más flexibles a la hora de usar policing. Gracias al uso de una cola bien configurada nos podemos permitir introducir una tasa de exceso en los tráficos AF, lo cual aumenta la flexibilidad del tráfico reduciendo el desaprovechamiento de la línea.

El retardo extremo a extremo del tráfico más sensible sigue estando garantizado gracias a emplear una cola de prioridad para el tráfico EF dentro de nuestra cola como podemos observar en la Figura 25.

A pesar de que ahora aparece un retardo extremo a extremo más elevado en los flujos de AF, esto es debido a que a diferencia de cuando solo teníamos policing la cantidad de paquetes que se pierden es mucho menor gracias al límite de exceso introducido en este policing adaptado para la cola menos restrictivo. Un 30% extra de los paquetes que anteriormente se hubieran descartado por superar el CIR ahora es recibido por las colas AF con una menor prioridad.

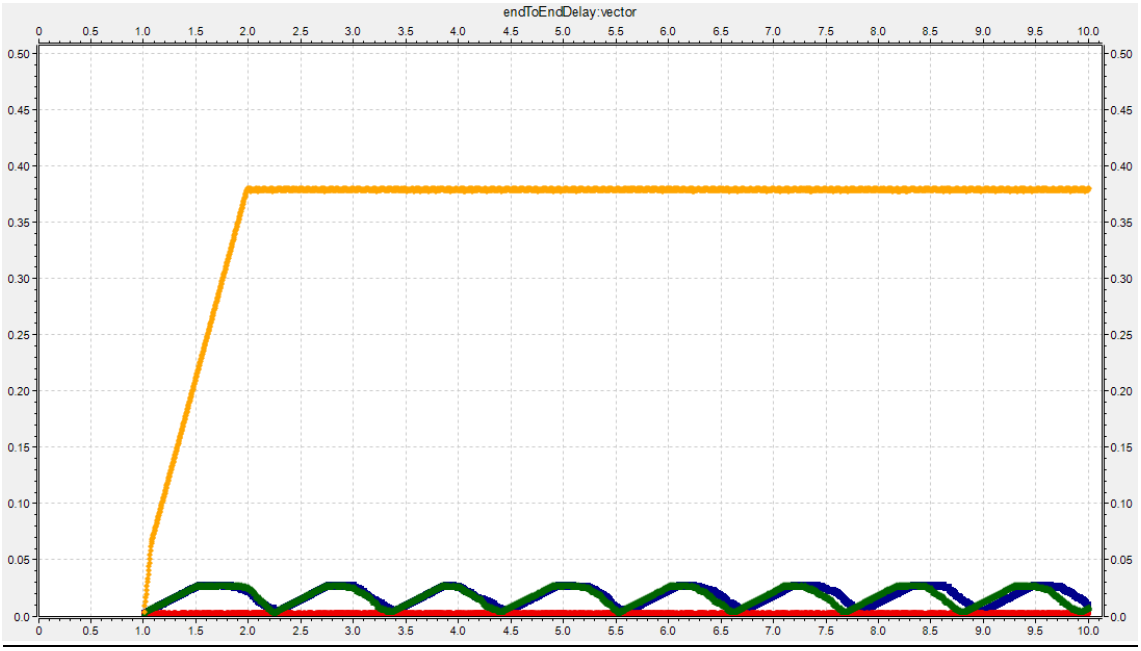

Figura 25. Retardo Final

Añadir la cola tras el policing nos permite introducir un límite de exceso a los flujos por encima del límite de conformidad, para que en caso de que alguno de los flujos cese o disminuya su velocidad podamos aprovechar el ancho de banda que ha sido liberado. Los pesos en WRR de la cola funcionan a modo de límite superior de velocidad para los flujos lo cual otorga más flexibilidad. Como podemos observar en la Figura 26, una vez detenemos el tráfico de AF1 tanto AF2 como BE se reparten el tráfico restante, evitando que la línea se desaproveche.

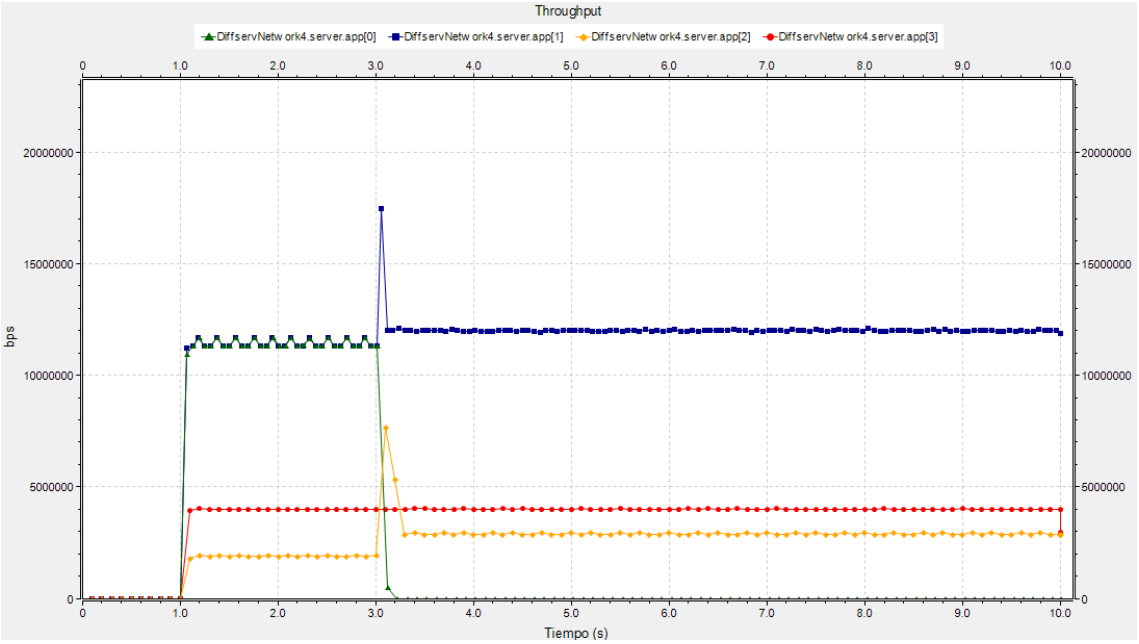

Figura 26. Throughput Final

El uso de una cola trae consigo una disminución de la pérdida de paquetes en las colas AF a cambio de una pérdida de paquetes BE, lo cual es deseable en la mayoría de los casos ya pues BE es un tráfico sin garantías y por el cual no debemos preocuparnos. Sería posible disminuirlo aún

más dándole menos prioridad a el tráfico BE, ya bien haciendo el meter más estricto en el condicionante de tráfico o dándole un peso menor en la cola. A menor prioridad BE más ancho de banda tienen los AF para repartirse y menos paquetes perderán en la congestión, como se puede percibir en la Figura 27.

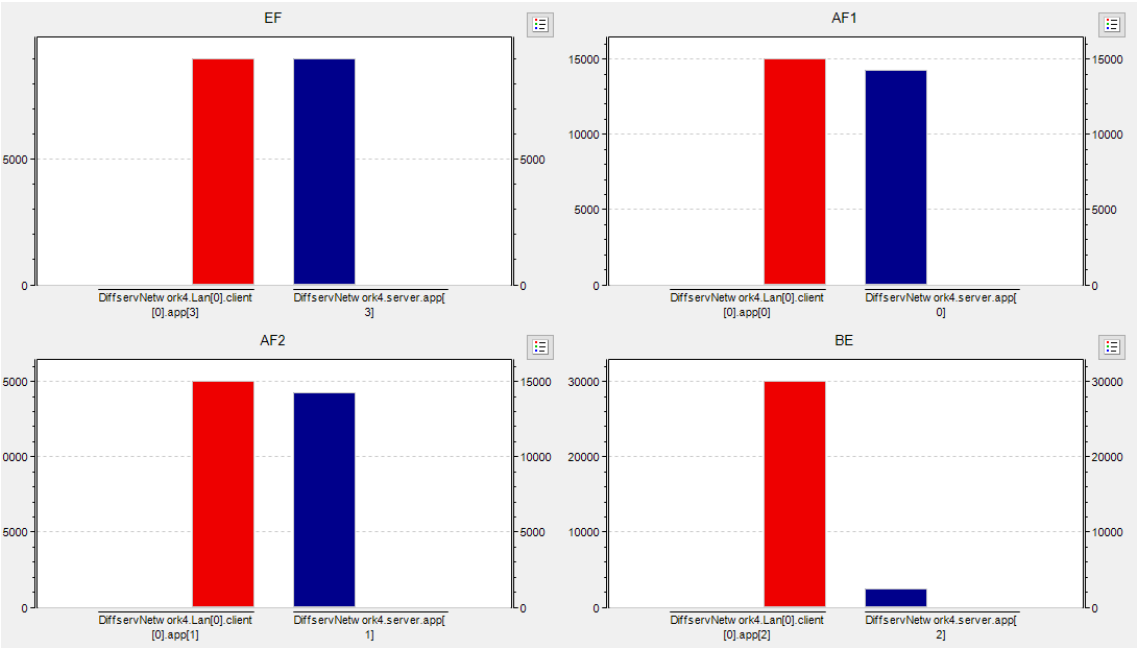

Figura 27. Packet Loss Final

5. Conclusiones

De las pruebas realizadas se puede llegar a ciertas conclusiones acerca de los mecanismos QoS en los escenarios simulados.

Se han simulado tres escenarios diferentes. En primer lugar, sin QoS, en el cual al entrar en estado de congestión todos los flujos llenan la cola y "pelean" por el ancho de banda, por tanto, todos los flujos pierden paquetes, a mayor velocidad el flujo menos pérdidas, y también sufrirán todos de un retardo de colas directamente proporcional al tamaño de la cola, ya que los paquetes que logren atravesar el enlace habrán tenido que esperar toda la cola para llegar.

En la segunda simulación se han observado los efectos del policing estricto, al añadir el condicionante de tráfico sobre la red. Esto nos permite asignar una velocidad máxima a los flujos de menor prioridad, permitiéndonos garantizar un retardo extremo a extremo reducido en todos los flujos y una pérdida de paquetes nula en flujos importantes siempre y cuando hayamos asignado un ancho de banda suficiente para el mismo.

En la tercera simulación se añade la cola que hemos empleado para brindar DiffServ a la red. Esta cola nos permite otorgar velocidades de exceso a los flujos para aumentar la utilización del enlace sin tener que sacrificar retardo extremo a extremo en los flujos más importantes. En su lugar sufriendo ese retardo extremo a extremo los tráficos de menor importancia.

Como podemos observar en las gráficas obtenidas durante la simulación, el condicionante de tráfico presenta un impacto positivo en la red, garantizando un retardo de colas y jitter bajos, así como pérdidas nulas en flujos de datos que prioricemos, siempre y cuando configuremos de manera correcta la red, aunque sacrificando los tráficos de menor importancia para garantizarlo.

El uso de policing o shaping puede causar pérdidas en ciertos flujos en las líneas operando por debajo su velocidad máxima. Esto puede suceder en flujos de baja prioridad en el caso de que traten de superar su velocidad pico (PIR). Esto es causado principalmente si el perfil de tráfico se sale de las expectativas de la red. En una situación normal los parámetros del QoS deberían estar configurados de manera personalizada para el tráfico esperado en esa interfaz.

Utilizar policing de manera estricta o shaping, en caso de redes dadas a ráfagas cortas, requiere tener un perfil de tráfico muy bien definido lo cual de manera habitual no es factible y por lo tanto requerimos de colas como el Weighted Round Robin que empleamos en este proyecto.

Ya que QoS es algo usado principalmente debido a la baja credibilidad de paquetes marcados por agentes externos a nuestra red QoS se emplea principalmente en redes privadas y, por tanto, a pesar de que existe un estándar de recomendaciones para QoS, el uso de este es altamente dependiente de cada red y el uso que se le quiera dar.

En definitiva, el presente Trabajo Final de Grado ha pretendido llevar a cabo un estudio teórico-práctico sobre las ventajas de QoS y las posibilidades de su utilización concreta.

6. Bibliografía

[1] Palo Alto Networks, «What is Quality of Service (QoS),» [En línea]. Available: https://www.paloaltonetworks.com/cyberpedia/what-is-quality-of-service-qos. [Último acceso: 23 12 2021].

[2] M. E. Flannagan, Administering cisco QoS in IP networks, 2001.

[3] C. &. S. Braden, «Integrated Services in the Internet Architecture: an Overview,» Junio 1994. [En línea]. Available: https://datatracker.ietf.org/doc/html/rfc1633. [Último acceso: 23 12 2021].

[4] *Omiotek, Zbigniew. (2012). A method for QoS differentiation in DiffServ networks based on the long-term properties of a video stream. Annales UMCS, Informatica. 12. 10.2478/v10065-012-0007-1.*.

[5] D. Matthews, «RFC 2597 Assured Forwarding Per-Hop Behaviour Group,» 19 4 2011. [En línea]. Available: http://darenmatthews.com/blog/?p=743. [Último acceso: 23 12 2021].

[6] S. a. J. V. Floyd. [En línea]. Available: http://www.icir.org/floyd/red.html. [Último acceso: 23 12 2021].

[7] IBM, «Traffic conditioners,» [En línea]. Available: https://www.ibm.com/docs/en/i/7.2?topic=service-traffic-conditioners. [Último acceso: 23 12 2021].

[8] Huawei, «NE40E-F V800R011C10 Feature Description - QoS 02,» [En línea]. Available: https://support.huawei.com/enterprise/es/doc/EDOC1100125532/c4ea13f2/token-bucket. [Último acceso: 23 12 2021].

[9] Huawei, «Traffic Metering and Token Bucket Mechanism,» [En línea]. Available: https://support.huawei.com/enterprise/en/doc/EDOC1000178317/f6e567c8/traffic-metering-and-token-bucket-mechanism. [Último acceso: 23 12 2021].

[10] J. Heinanen, T. Finland y R. Guerin, «A Two Rate Three Color Marker (RFC 2698),» [En línea]. Available: https://datatracker.ietf.org/doc/html/rfc2698. [Último acceso: 23 12 2021].

[11] Omnet++, [En línea]. Available: https://doc.omnetpp.org/omnetpp/manual/. [Último acceso: 23 12 2021].

[12] Omnet++, «INET User Guide,» [En línea]. Available: https://inet.omnetpp.org/docs/users-guide/index.html. [Último acceso: 23 12 2021].

[13] Alibaba Cloud, «CIR,CBS,EBS,PIR,PBS noun Explain token bucket application,» [En línea]. Available: https://topic.alibabacloud.com/a/circbsebspirpbs-noun-explain-token-bucket-application_8_8_31065920.html. [Último acceso: 23 12 2021].

[14] Cisco, «Comparing Traffic Policing and Traffic Shaping for Bandwidth Limiting,» [En línea]. Available: https://www.cisco.com/c/en/us/support/docs/quality-of-service-qos/qos-policing/19645-policevsshape.html. [Último acceso: 23 12 2021].

[15] Cisco, «DiffServ -- The Scalable End-to-End QoS Model,» [En línea]. Available: https://www.cisco.com/en/US/technologies/tk543/tk766/technologies_white_paper09186a00800a3e2f.html. [Último acceso: 23 12 2021].

[16] Cisco, «Fundamentos de QoS,» [En línea]. Available: https://community.cisco.com/t5/blogs-routing-y-switching/fundamentos-de-qos-calidad-de-servicio-en-capa-2-y-capa-3/ba-p/3103715. [Último acceso: 23 12 2021].

[17] NetCraftsmen, «INBOUND QOS,» [En línea]. Available: https://netcraftsmen.com/inbound-qos/. [Último acceso: 23 12 2021].

[18] Hillstone Networks, «Introduction to QoS,» [En línea]. Available: https://www.hillstonenet.com/support/4.5/en/config_qos_intro.html. [Último acceso: 23 12 2021].

[19] Huawei, [En línea]. Available: https://support.huawei.com/enterprise/es/doc/EDOC1100143232/21bf3548/dscp. [Último acceso: 23 12 2021].

[20] Techtarget, «quality of service (QoS),» [En línea]. Available: https://searchunifiedcommunications.techtarget.com/definition/QoS-Quality-of-Service. [Último acceso: 23 12 2021].

[21] Juniper Networks, «Único algoritmo de cubo de tokens,» [En línea]. Available: https://www.juniper.net/documentation/mx/es/software/junos/routing-policy/topics/concept/policer-algorithm-single-token-bucket.html. [Último acceso: 23 12 2021].

[22] Omnet++, «Omnet++ Documentation,» [En línea]. Available: https://doc.omnetpp.org/inet/api-current/neddoc/index.html. [Último acceso: 23 12 2021].