IMPACTO DE LOS MECANISMOS DE CONTROL DE CONTENCIÓN EN REDES OBS DISTRIBUIDAS

Kevin Javier Guevara Ortiz
Yimi Rodrigo Díaz Erazo

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Grupo I+D Nuevas Tecnologías en Telecomunicaciones - GNTT
Departamento de Telecomunicaciones
Popayán, 2015

IMPACTO DE LOS MECANISMOS DE CONTROL DE CONTENCIÓN EN REDES OBS DISTRIBUIDAS

Kevin Javier Guevara Ortiz
Yimi Rodrigo Díaz Erazo

Trabajo de Grado presentado como requisito para obtener el título de Ingeniero en Electrónica y Telecomunicaciones

Director
Ing. José Giovanny López Perafán. PhD.

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Grupo I+D Nuevas Tecnologías en Telecomunicaciones - GNTT
Departamento de Telecomunicaciones
Popayán, 2015

Tabla de contenido

LISTA DE FIGURAS

LISTA DE TABLAS

Lista de Acrónimos

ATM *Asynchronous Transfer Mode,* Modo de Transferencia Asíncrona.

CWDM *Coarse Wavelength Division Multiplexing,* Multiplexación por División de Longitudes de onda Espaciadas.

DWDM *Dense Wavelength Division Multiplexing,* Multiplexación por División de Longitud de onda Densa.

EDFA *Erbium Doped Fiber Amplifier,* Amplificador Óptico de Fibra Dopada con Erbio.

FDL *Fiber Delay Line,* Línea de Fibra de Retardo.

GMPLS *General Multiprotocol Label Switching,* Conmutación por Etiquetas Multiprotocolo Generalizado.

IP *Internet Protocol,* Protocolo de Internet.

JET *Just End Time,* Justo el Tiempo Suficiente.

OBS *Optical Burst Switching,* Conmutación Óptica de Ráfagas.

OPS *Optical Packet Switching,* Conmutación Óptica de Paquetes.

OCS *Optical Circuit Switching,* Conmutación Óptica de Circuitos.

OLT *Optical Line Termination,* Terminales Ópticos de Línea.

OXC *Optical Cross-Connect,* Matrices de Conmutación Óptica.

OADM *Optical add-drop Multiplexer,* Multiplexores Ópticos de Adición-Extracción de longitud de onda.

PDU *Protocol Data Unit,* Unidad de Datos de Protocolo.

RAM *Random Access Memory,* Memoria de Acceso Aleatorio.

SCU *Switching Control Unit,* Unidad de Control de Conmutación.

SDH *Synchronous Digital Hierarchy,* Jerarquía Digital Síncrona.

WDM *Wavelength Division Multiplexing,* Multiplexacíon por División de Longitud de onda.

INTRODUCCIÓN

Debido al crecimiento exponencial del tráfico que circula a través de Internet, es necesario desarrollar nuevas arquitecturas que permitan utilizar eficientemente los recursos, proporcionando una asignación flexible y dinámica al tráfico de información. La conmutación óptica de ráfagas (OBS, *Optical Burst Switching*) se ha propuesto como una forma eficiente para satisfacer las necesidades futuras de ancho de banda en las redes ópticas.

Las redes OBS utilizan al máximo el ancho de banda ofrecido por la fibra óptica, para lo cual ha sido necesario deshacerse del procesamiento eléctrico, que es la causa de la generación de los cuellos de botella en las redes ópticas, pero a cambio de ello se deben implementar nuevos mecanismos, llamados mecanismos de control de contención. Sin procesamiento eléctrico los paquetes no pueden ser guardados, dado que en la actualidad no existen memorias ópticas o están en proceso de desarrollo, se crean por lo tanto tales mecanismos para controlar el acceso de una ráfaga a ocupar un canal óptico, cuando ésta contiende con otra por el mismo recurso.

En el capítulo uno de este trabajo de grado se hace un breve recorrido por la historia de las redes ópticas: su evolución y desarrollo desde los primeros experimentos de transmisión con luz hasta las redes en la actualidad, donde se tienen redes completamente ópticas de muy buenas prestaciones, incluso sobrepasando a las demás tecnologías.

En el capítulo dos se presenta la tecnología de conmutación óptica de ráfagas, se analizan los procesos que se llevan a cabo en la red para el envío y recepción de paquetes y ráfagas, y los parámetros que usa cada nodo para su configuración. También se estudia el proceso de encaminamiento como una de la funciones más importantes de las redes de comunicaciones, y los protocolos de planificación del enlace que determinaran la mejor organización de los canales y tiempos de conmutación para obtener la máxima eficiencia posible, que en el caso de la red OBS diseñada para este trabajo de grado, se utiliza JET como algoritmo de planificación.

Igualmente, las metaheurísticas proporcionan un aporte esencial para el desarrollo de este trabajo, debido a que permitirán solucionar de manera eficiente las dificultades de enrutamiento y contención necesarias para el funcionamiento de la red.

En el capítulo tres se describe la metodología empleada para la realización de este proyecto. Se inicia el proceso con el diseño de una red de transporte de conmutación de ráfagas, en la cual se simula el flujo de los mensajes, se hace luego una comparación del comportamiento de la red al aplicarse a los nodos centrales tres mecanismos de control de contención diferentes.

Se presenta por último, el análisis de los tres mecanismos de control de contención empleados, para este fin se utiliza la herramienta de simulación

OMNet++, en ésta se programan todos los módulos necesarios para el correcto funcionamiento de la red OBS y mediante la variación de algunos parámetros se obtienen los resultados para luego compararlos y dar las respectivas conclusiones.

CAPÍTULO 1

EVOLUCIÓN DE LAS REDES ÓPTICAS

Los sistemas de telecomunicaciones soportados en redes ópticas crecen a un ritmo acelerado gracias a la explosión de la internet, que cuentan con mayor capacidad y velocidad de transmisión, estos sistemas son movidos por la gran demanda de servicios de alta velocidad que exigen más ancho de banda y capacidad. Es de destacar la importancia de la técnica de multiplexación por división de longitud de onda (WDM, *Wavelength Division Multiplexing*) en la cual se utilizan diferentes longitudes de onda para la transmisión de información mediante fibra óptica, así mismo, se han desarrollado los amplificadores ópticos que han permitido el rápido avance de la tecnología de la fibra óptica [1].

Desde la aparición de la fibra óptica se ha dado un cambio importante en el mundo de las comunicaciones, debido a las grandes ventajas que trae en comparación con los otros medios físicos ya existentes tales como: el cable coaxial, par trenzado o la transmisión por radiofrecuencias. Todo este desarrollo se ha dado en etapas y se consideran en cinco grandes grupos de acuerdo con su evolución, redes de primera, segunda, tercera, cuarta y quinta generación.

1.1 Redes ópticas de primera generación

En las redes ópticas de primera generación, todo el tráfico proveniente de cualquier nodo transmitido a través de enlaces WDM era procesada en el dominio eléctrico, esto es, debía cada nodo tener convertidores óptico–eléctrico para todo el tráfico de entrada y convertidores eléctrico-óptico para todo el tráfico de salida del nodo. El costo de estos dispositivos era demasiado alto y su complejidad elevada debido a la cantidad de señales eléctricas a procesar.

En estos sistemas, aunque se utilizaba la fibra óptica como medio de transmisión de alta capacidad, todo el procesado, enrutamiento y conmutación era realizado en el dominio eléctrico.

Este tipo de redes no eran eficientes en cuanto a que no aprovechaban el gran ancho de banda que ofrece la fibra óptica, por tal motivo se introducen en la red dispositivos de extracción de información de control para cada paquete, dejando pasar el resto de tráfico sin ninguna intervención eléctrica, de esta manera se evita el procesamiento excesivo e innecesario de señales eléctricas. Esta nueva técnica permite el comienzo de las redes ópticas de segunda generación [1].

1.2 Redes ópticas de segunda generación

En la segunda generación de las redes ópticas se realizan funciones adicionales en el dominio óptico que anteriormente se limitaba a la simple transmisión punto a punto. Para esta generación ya se cuenta con nuevas técnicas de enrutamiento, conmutación y de más funciones para el control y gestión de la red, haciendo de ésta manera una red mucho más económica y de más capacidad debido al considerable ahorro de equipos y procesamiento electrónico en cada uno de los nodos. Es en esta etapa donde se presenta el concepto de capa óptica dentro del modelo de capas.

Los multiplexores de adición-extracción de longitud de onda completamente ópticos (OADM, *Optical add-drop Multiplexer*), terminales ópticos de línea (OLT, *Optical Line Termination*) y matrices de conmutación óptica (OXC, *Optical Cross-Connect*) son la base de las redes de la segunda generación. Los multiplexores permiten el paso de una mayor cantidad de tráfico evitando procesamiento y reduciendo costos en la red. Los OXC pueden conmutar longitudes de onda desde cualquier puerto de entrada hasta la salida del nodo y los OLTs que son los elementos que permiten la conexión de los multiplexores con los OXCs.

En la figura 1.1, se muestran los principales elementos hardware y los caminos ópticos en un esquema de una red de segunda generación [1].

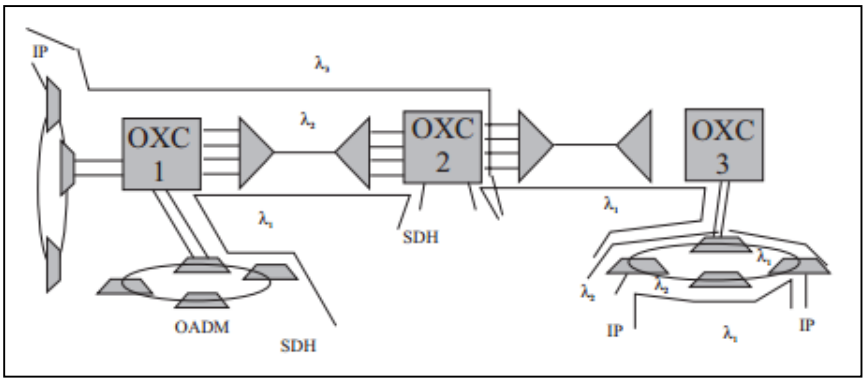

Figura 1.1. Red óptica de segunda generación [1].

El objetivo del desarrollo de ésta tecnología es poder trasladar el tráfico de las capas ATM y SDH al dominio óptico, puesto que es el tráfico IP el que predomina, entonces el propósito es crear los protocolos que den más facilidad para su transporte.

1.3 Redes ópticas de tercera generación

Hay dos tecnologías prometedoras para cubrir la gran cantidad de tráfico que genera la creciente demanda de usuarios sobre las redes ópticas WDM. La conmutación de paquetes ópticos y la conmutación de ráfagas ópticas.

La conmutación de ráfagas ópticas se refiere a paquetes que son ensamblados dentro de unidades de transporte definidas como ráfagas y que se encaminan por

una red completamente óptica. Se considera esta técnica como un paso intermedio entre la conmutación de circuitos y la conmutación de paquetes ópticos. Este tipo de redes constan de nodos de frontera, donde se ensamblan las ráfagas y de nodos centrales que se encargan del encaminamiento de las mismas. Además, los nodos frontera crean los paquetes de control, uno por cada ráfaga de datos, este paquete contiene la cabecera de su ráfaga asociada, razón por la cual se envía por un canal independiente llamado canal de control.

Para este tipo de redes no se necesitan buffers o memorias para el enrutamiento de las ráfagas.

En las redes OPS, los paquetes están formados por la cabecera y los datos que se multiplexan en la capa óptica (DWDM), éstos paquetes son completamente ópticos. En esta tecnología, se pretende llevar las operaciones que realizaba en el dominio eléctrico pero esta implementación resulta muy compleja. En el dominio óptico las longitudes de ondas que forman cada enlace se comparten por todos los paquetes entre dos nodos de la red desde el origen hasta el destino.

Los conmutadores solo pueden realizar procesamiento en el dominio eléctrico, por lo que es necesario pasar las cabeceras a éste dominio para hacer las acciones requeridas de encaminamiento.

Para los paquetes que necesiten ser almacenados para su respectiva conmutación debería usarse memorias RAM ópticas que aún se encuentran en desarrollo, o en otro caso utilizar líneas de retardo que no resultan muy viables debido a que introducen retardos [1].

Dado que implementar una red OPS resulta complicado, OBS se convierte en la solución con mejores perspectivas para la realización de redes totalmente ópticas. En la figura 1.2 se muestran los esquemas de red OPS y OBS.

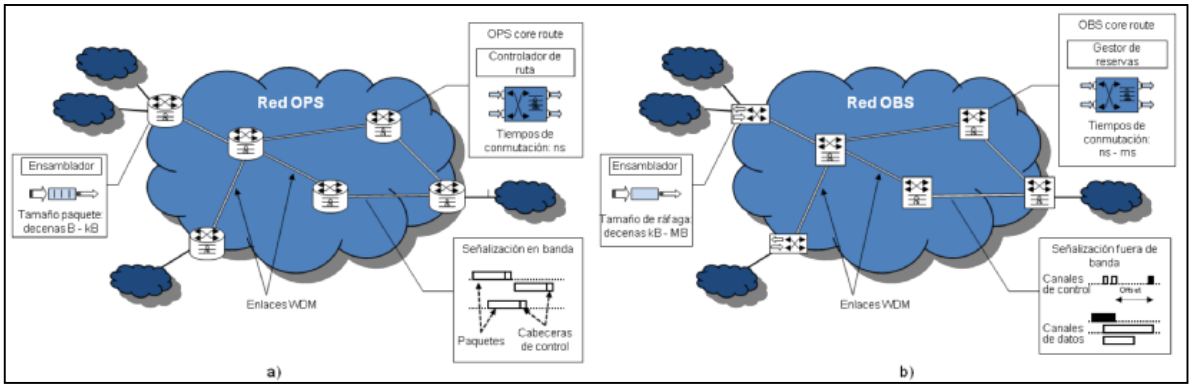

Figura 1.2. Redes de Conmutación de Tercera Generación [2].

1.4 Redes ópticas de cuarta generación

Las redes de transporte óptico continúan su desarrollo con el empeño de ser cada vez más eficientes, avanzando con la confianza de llegar a ser redes totalmente ópticas.

Para que el tráfico de información pueda viajar totalmente en el dominio óptico ha sido necesario el desarrollo de sistemas tales como: conmutadores micro electromecánicos, amplificadores semiconductores y multiplexores ópticos.

Las redes de cuarta generación o también llamadas redes totalmente ópticas (OTN, *Optical Transport Network*) combinan la flexibilidad y el manejo de SDH y con las ventajas de WDM, puede combinar múltiples redes y servicios tales como SDH/SONET tradicional, video, Ethernet y protocolos de almacenamiento sobre una infraestructura común, reduciendo significativamente los gastos en capital y de operación. En la figura 1.3 se muestra la infraestructura de la red OTN de cuarta generación.

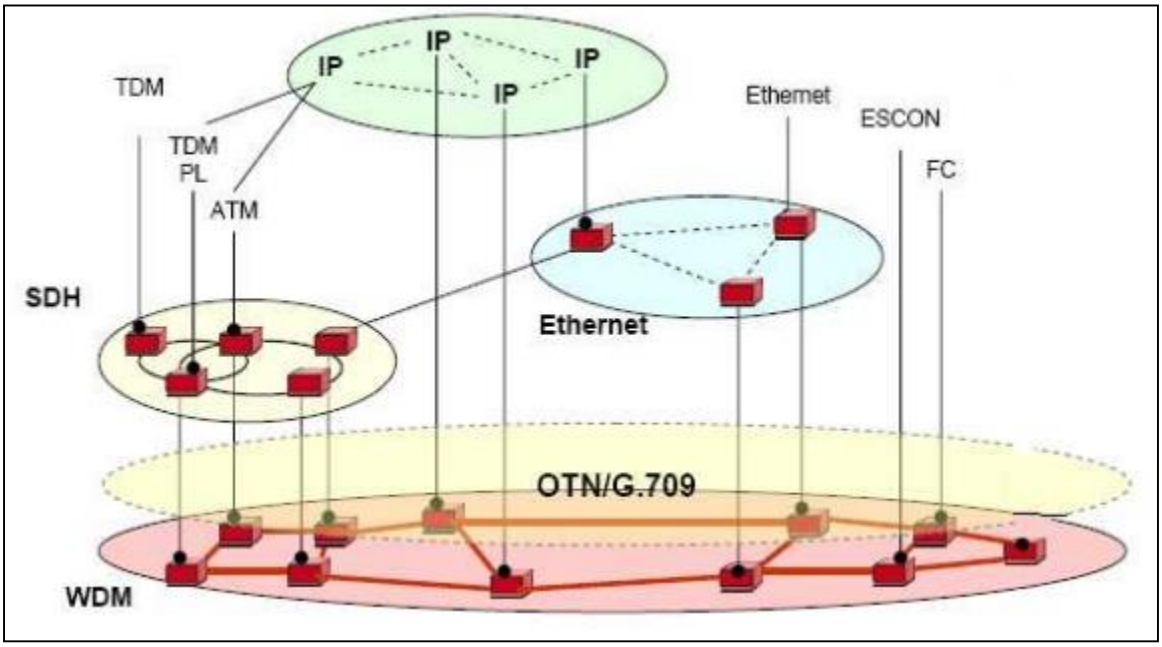

Figura 1.3. Redes de Conmutación óptica de cuarta Generación [2].

1.5 Redes ópticas de quinta generación

La arquitectura de las redes de quinta generación se determina para las Redes Definidas por Software (SDN, *Software Define Networks*). Estas redes son capaces de transportar un alto y cambiante volumen de tráfico [3].

La empresa HUAWEI lanza la arquitectura de red Flex Optical Network (Red Óptica Flexible) que puede mejorar de manera efectiva la eficiencia espectral mientras que la capa eléctrica flexible adopta tecnología 100G basada en arquitectura Flex-OTN (OTN Flexible). Esta última permite un servicio flexible de conmutación y multiplexado entre enlaces de gran capacidad con lo cual se pueden alcanzar capacidades de transmisión y distancia bajo demanda [2].

1.6 Tipos de fibra óptica

Los sistemas de comunicación por fibra óptica han revolucionado la industria de las telecomunicaciones y han desempeñado un papel importante en el desarrollo del mundo de la información. Debido a sus ventajas sobre la transmisión eléctrica, la fibra óptica se pone a la delantera como el medio de transmisión para las redes del futuro.

La fibra óptica es un medio de transmisión que se compone de un material como el vidrio o plástico transparente y que se organiza en hilos de vidrio muy delgados por los cuales se envían los pulsos de luz que representan la información a transmitir. La fibra es muy empleada en telecomunicaciones para la transmisión a gran distancia debido a que los datos se propagan a mucha velocidad similar a las ondas de radio, posee gran ancho de banda y además es inmune a las interferencias electromagnéticas.

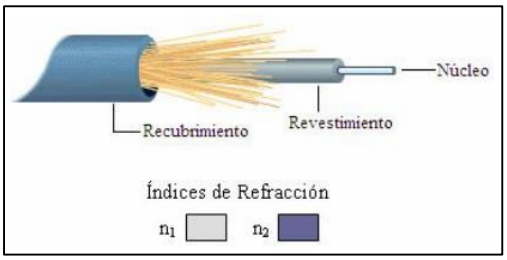

Figura 1.4. Composición Fibra óptica [4].

Una fibra óptica consiste en dos partes, como se puede ver en la figura 1.4, un núcleo interior y un material de envoltura exterior. El núcleo y la envoltura están compuestos de tal manera que hacen que se conserven las señales de luz dentro de la fibra, permitiendo que los pulsos de luz sean transmitidos para largas distancias a gran velocidad [4].

Los diferentes caminos que puede seguir un haz de luz en el interior de una fibra se denominan modos de propagación. Y según el modo de propagación se tienen dos tipos de fibra óptica: multimodo y monomodo.

En la fibra multimodo los rayos de luz pueden propagarse por más de un camino o modo de propagación. Debido a que cada camino tiene una distancia diferente se supone que al final de la fibra los pulsos luminosos deben llegar en tiempos diferentes.

Las fibras multimodo se emplean normalmente para distancia cortas, menores a 2 km, es fácil de diseñar y de bajo costo. En la figura 1.5 se observa el trayecto que sigue el haz de luz [4].

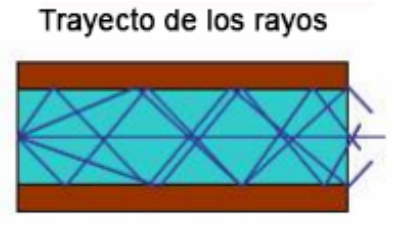

Figura 1.5. Fibra multimodo [4].

En la fibra monomodo el diámetro del núcleo se reduce de 8,3 a 10 micras, éste tamaño obliga a que toda la energía en una señal de luz se propague por un solo camino, el único camino que puede tomar la luz es el paralelo a la fibra. Las fibras monomodo se emplean para aplicaciones de grandes distancias (hasta 400 Km máximo con un láser de alta potencia) y transmitir mucha información. El objeto del desarrollo de éste tipo de fibra es eliminar la dispersión intermodal que tuvo como consecuencia el aumento dramático en la tasa de bits [4]. En la figura 1.6 se observa el trayecto que sigue el haz de luz.

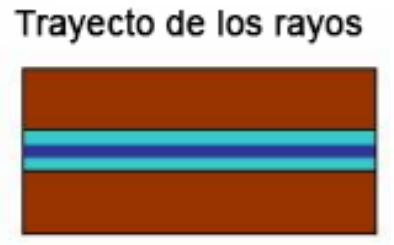

Figura 1.6. Fibra monomodo [4].

1.7 Amplificadores ópticos y WDM

El desarrollo de los amplificadores de fibra dopada con erbio (EDFA, *Erbium Doped Fiber Amplifier*) da comienzo a una nueva etapa en el desarrollo de los sistemas de transmisión de fibra óptica.

Los amplificadores EDFA proporcionaron otra manera de aumentar la capacidad del sistema al ser capaces de amplificar simultáneamente señales en muchas longitudes de onda y utilizar multiplexación por división de longitud de onda

Los primeros amplificadores EDFAs funcionaban en la banda C, hoy en día se tienen EDFAs funcionando en la banda L y a los sistemas WDM igualmente actuando en dichas bandas.

Se han desarrollado también los amplificadores Raman que utilizan el sistema WDM. Estos amplificadores vienen a complementar a los EDFAs dando apertura a nuevas bandas de fibra como la banda S y la banda U. En la figura 1.7 se muestra un enlace que utiliza multiplexores y amplificadores [4].

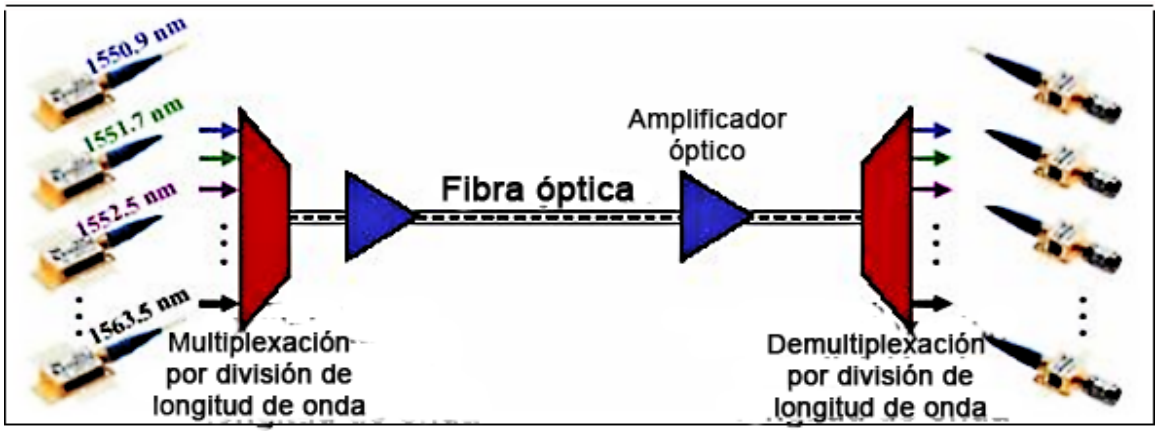

Figura 1.7. Enlace con multiplexación y amplificación [4].

1.8 Enlaces de transmisión y redes

En lo que respecta a las redes, desde que se descubrió la fibra como medio de transmisión, ya se pensaba en nuevas arquitecturas de red para la transmisión de grandes cantidades de información, pero el alto costo de esta tecnología dificultó su aplicación. En la actualidad se continúa con investigaciones sobre redes ópticas de conmutación de paquetes ópticos pero más interés tiene las redes de enrutamiento por longitud de onda dado los beneficios de tener una capa óptica.

Hoy en día ya se encuentran disponibles como productos comerciales multiplexores OADM y matrices OXC a precios asequibles, esto dado el hecho que los sistemas de alta capacidad para la transmisión de información resulta más económico a nivel óptico que en el eléctrico además de otras ventajas como la inmunidad a interferencias electromagnéticas o la facilidad que poseen de incorporar funcionalidades integradas de restauración para hacer frente a los fallos de red [1].

1.9 Multiplexación por División de Longitud de Onda (WDM, *Wavelength Division Multiplexing*)

La multiplexación por división de longitud de onda es una tecnología que se desarrolló para aprovechar mejor las capacidades de la fibra óptica, debido a que en un principio no se utilizaba todo el ancho de banda disponible en una fibra y por lo tanto la implementación de la misma resultaba complicada y sin mucha relevancia para lograr un cambio significativo en la red. Por estas razones la creación de esta tecnología trajo consigo la evolución de las redes ópticas, puesto que fomentó un uso más adecuado de la fibra óptica aumentando de manera exponencial la capacidad de las redes ópticas.

WDM consiste en usar varias longitudes de onda dentro de una fibra para enviar varias señales sobre distintas portadoras ópticas, estas portadoras se encuentran distanciadas entre sí para que no se vaya a generar alguna interferencia que pueda afectar la señal de información. Cada señal es transmitida por una fuente distinta, que después de pasar por un conversor eléctrico a óptico, es combinada

por un multiplexor que combina las señales ópticas generadas por un led o laser, estas señales se transmiten en un mismo haz de luz, sin embargo, las señales no se transponen ya que a cada señal se le asigna una longitud de onda distinta en el haz de luz. Al llegar a su destino un demultiplexor separa las señales ópticas que posteriormente se convierten en eléctricas, como se muestra en la figura 1.8 [5] [6].

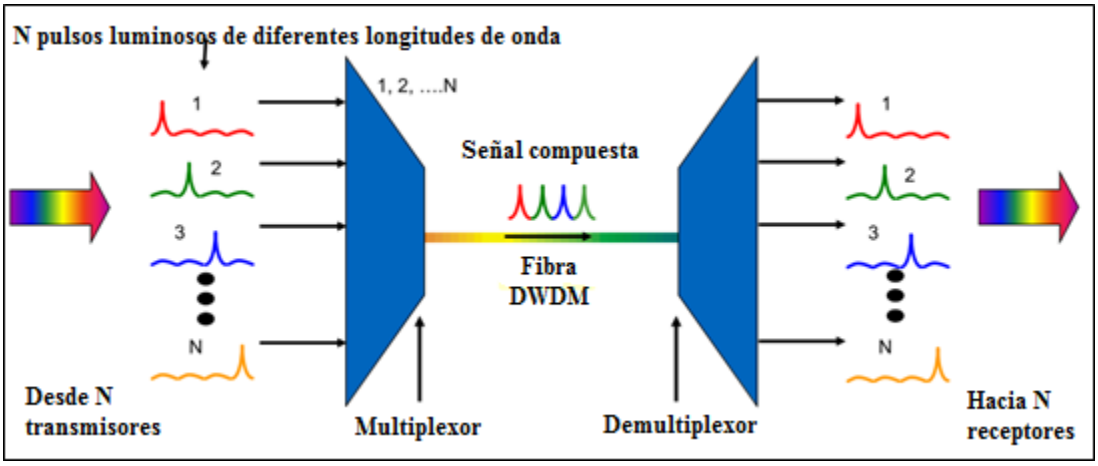

Figura 1.8. Sistema WDM [7].

1.9.1 Multiplexación por División de Longitud de Onda Densa (DWDM, *Dense Wavelength Division Multiplexing*)

Este tipo de sistemas permite multiplexar en una sola fibra muchas portadoras ópticas en distintas longitudes de onda, debido a que se utiliza un espaciamiento mínimo entre los canales ópticos (50Ghz, 100Ghz o 200Ghz), permitiendo transportar desde 40 hasta 160 canales ópticos en el mejor de los casos. Esto se logra gracias al uso de láseres ópticos de precisión y mejor calidad que los comunes, baja dispersión en las fibras utilizadas y módulos que compensan dicha dispersión.

DWDM usa la banda C (1525nm-1565nm) y la banda L (1565nm-1620nm), como se muestra en la figura 1.9, debido a que este rango espectral permite el uso de amplificadores de fibra dopada con erbio (EDFA), lo que es muy útil para enlaces de transmisión a largas distancias, por lo tanto DWDM es uno de los sistemas más utilizados para aplicaciones que manejan grandes cantidades de tráfico [8] [6].

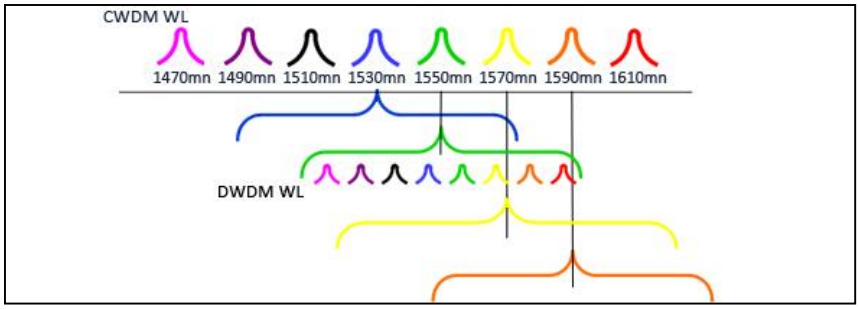

Figura 1.9. Bandas de Frecuencia usadas en DWDM y CWDM [9].

1.9.2 Multiplexación por división de longitudes de onda con mayor espaciamiento (CWDM, *Coarse Wavelength Division Multiplexing*)

Es un tipo de multiplexación que permite el transporte de varias portadoras ópticas en una misma fibra a un menor costo que el sistema DWDM, sin embargo, con una capacidad menor pero que es compensada con la baja complejidad en la implementación de este sistema. Dado que usa separación entre canales de 20nm (2500Ghz) cuyo rango de frecuencias se encuentra entre 1270-1610nm, como se muestra en la figura 1.10, puede transportar hasta 18 longitudes de onda en una fibra monomodo, por lo tanto los láseres ópticos que se usan en este sistema no necesitan ser de alta precisión, incluso hasta puede soportar distorsión o algún tipo de imperfección en los componentes sin que esto afecte de manera significativa las señales.

En este tipo de sistema la señal no se amplifica como se hace en DWDM, es más eficiente y menos costoso regenerar la señal en cada uno de los nodos por los que pasa, entonces cada canal se somete por separado a una conversión óptica-eléctrica-óptica. Por lo tanto la regeneración cumple las funciones de amplificar, reconstruir y temporizar la señal eliminando la dispersión generada en la fibra sin necesidad de utilizar un elemento en la red que compense este tipo de deterioro en la señal, logrando llegar a alcanzar distancias de transmisión de 80km [9] [10].

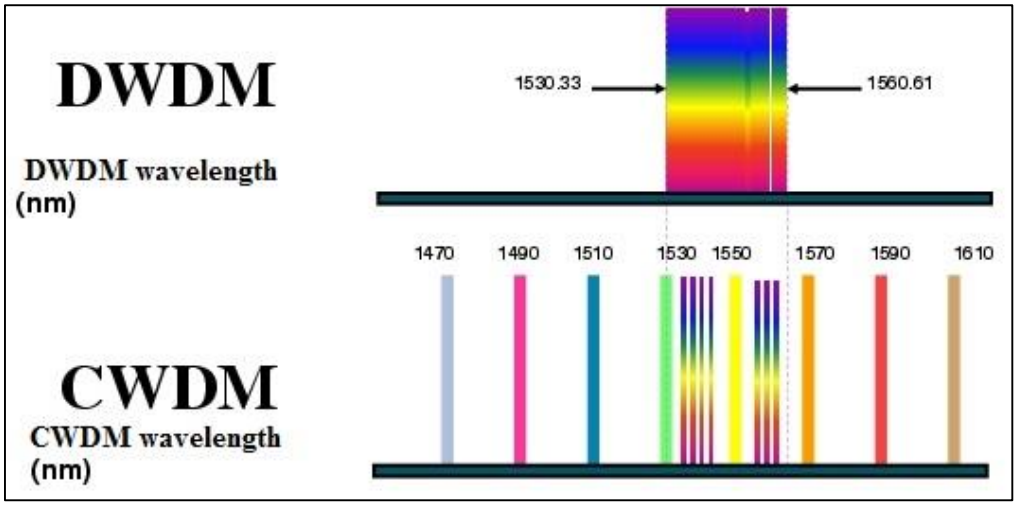

Figura 1.10. Bandas de uso CWDM Y DWDM [11].

1.10 Tipos Conmutación óptica

Los sistemas ópticos presentan distintas maneras de conmutación que se han ido desarrollando desde el uso de este tipo de redes, buscando siempre una mejor eficiencia en el desempeño de las mismas. Cada tipo de conmutación consta de un método único para realizar este proceso, por lo tanto todos constan de ventajas y desventajas significativas entre ellos que al final son la base que se usa para establecer cuál de estos se usará o conviene más para algún tipo de red.

A continuación se muestran los sistemas de conmutación óptica de circuitos, conmutación óptica de ráfagas y conmutación óptica de paquetes, que son los tipos de conmutación más relevantes debido al gran nivel de desarrollo e investigación que se ha hecho sobre ellos. También se hará especial énfasis en sus características específicas para tener un mejor detalle sobre su funcionamiento.

1.10.1 Conmutación Óptica de Circuitos (OCS, *Optical Circuit Switching*)

Consiste en el establecimiento previo de caminos ópticos entre los nodos que se van a comunicar, este tipo de conmutación asigna una longitud de onda única para la comunicación entre nodos, como se puede ver en la figura 1.11, donde se reservan dos longitudes de onda que se usaran para transportar dos paquetes de información desde su nodo transmisor, hasta su nodo receptor. Por medio de un paquete de control se reservan los recursos necesarios para la conexión, en caso de que algún nodo no tenga recursos disponibles se realiza todo el proceso de establecimiento de ruta nuevamente [12].

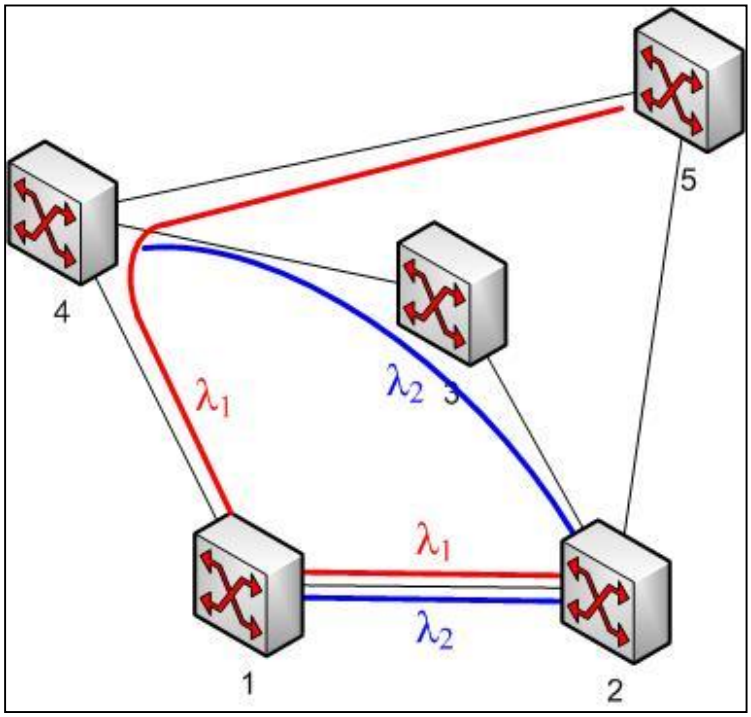

Figura 1.11. Conmutación Óptica de Circuitos [12].

El funcionamiento de OCS presenta ventajas para comunicaciones en las que se vaya a transportar un gran volumen de datos, puesto que desde el establecimiento de la comunicación se conservan los recursos proporcionados para la misma, el inconveniente de esta forma de asignar recursos de manera estática es que pueden presentarse dificultades al establecer longitudes de onda y caminos ópticos cuando en una red hay muchos nodos que quieren comunicarse, debido a que la red se puede congestionar y así finalmente saturarse, lo que no permitiría realizar la asignación de recursos para la comunicación [13].

1.10.2 Conmutación Óptica de Ráfagas (OBS, *Optical Burst Switching*)

Es considerada como el intermedio entre la conmutación óptica de circuitos y la conmutación óptica de paquetes, ya que abarca las ventajas de cada tipo de conmutación y además se puede comportar como estos métodos de conmutación dependiendo del tamaño de la ráfaga que maneje. En OBS, los paquetes con características similares de destino, calidad del servicio, entre otras, son agrupados en ráfagas para posteriormente ser enviados. Inicialmente los datos llegan a un nodo de borde, que se encarga de ensamblar/desensamblar la ráfaga, establecer la conexión y asignar un tiempo de offset entre el paquete de control y la ráfaga como se puede ver en la figura 1.12. Este paquete de control es asignado a cada ráfaga y se encarga de reservar los recursos a través de la red para la ráfaga, el tiempo de offset entre el paquete de control y la ráfaga permite que los nodos tengan un rango de tiempo suficiente para asignar los recursos deseados y encaminar debidamente la ráfaga en la red.

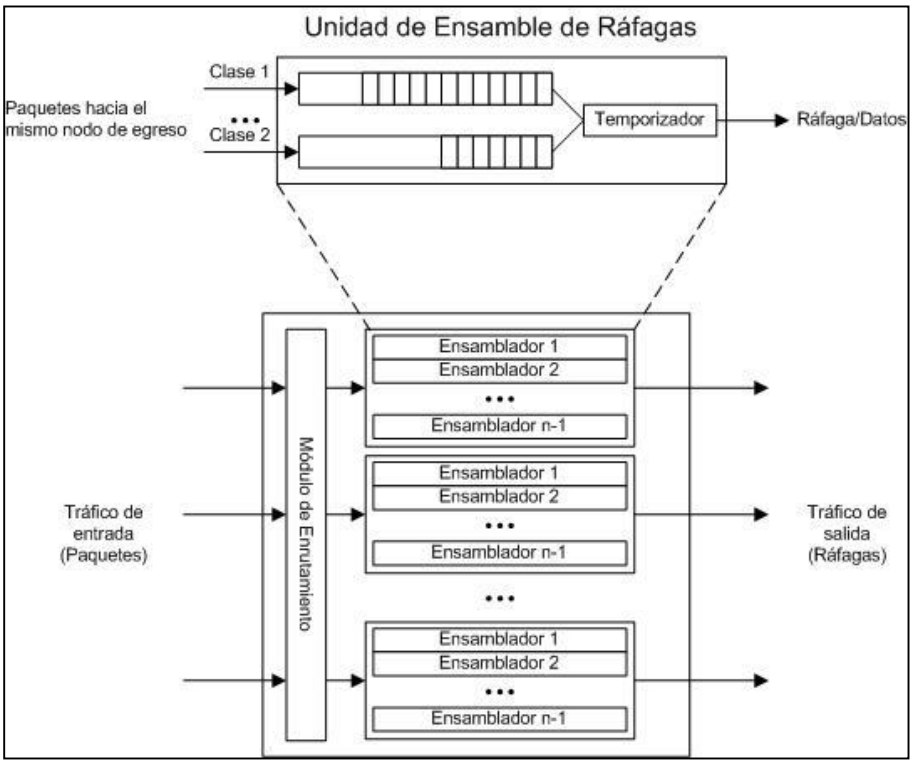

Figura 1.12. Nodo de Borde/Frontera red OBS [12].

Después la ráfaga se dirige a los nodos core o nodos de núcleo que se encargan de conmutar las ráfagas, definir el tipo de contención que se dispondrá, realizar la respectiva señalización para cada ráfaga y asignar los recursos como se muestra en la figura 1.13.

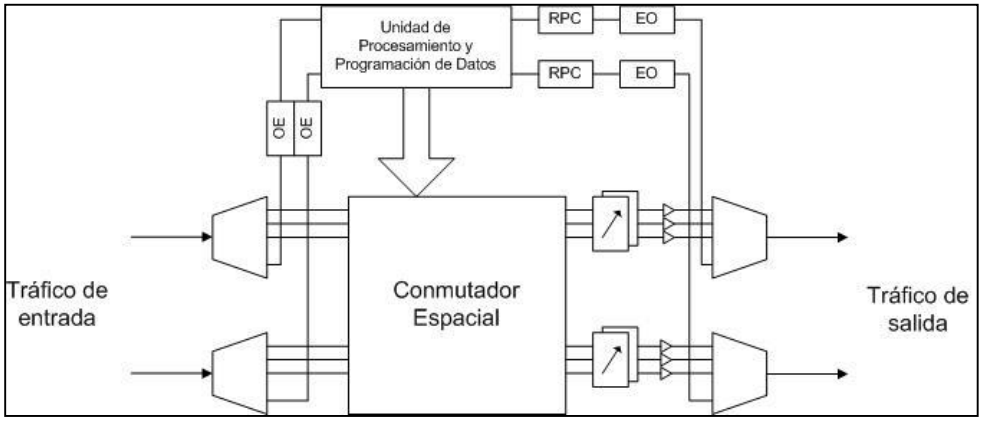

Figura 1.13. Nodo Central/Núcleo red OBS [12].

Finalmente, la ráfaga pasa nuevamente por un nodo de frontera, que desensambla la ráfaga y envía los paquetes a su destino final.

OBS presenta ventajas de flexibilidad y manejo respecto a los otros métodos de conmutación, sin embargo, falta investigación y desarrollo de componentes ópticos que permitirían una mayor eficiencia para la implementación de redes completamente ópticas que puedan usar estas tecnológicas a su máxima capacidad, pero aun así, OBS sigue siendo una tecnología con la que se puede trabajar de manera eficiente en redes ópticas [12] [13].

1.10.3 Conmutación Óptica de Paquetes (OPS, *Optical Packet Switching*)

En OPS la información es segmentada en paquetes, cada paquete está conformado por la carga útil y una cabecera, en la cabecera se transporta toda la información de configuración para el proceso conmutación de los paquetes en cada nodo como se muestra en la figura 1.14. Mientras se realiza el proceso de lectura y procesamiento de configuración en cada nodo, el paquete se guarda en una “memoria” que en este caso se trataría de una línea de retardo. Este tipo de conmutación proporciona una gran ventaja en cuanto a flexibilidad y tratamiento de los recursos ópticos, puesto que la asignación de longitudes de onda para el transporte de la información se hace de manera dinámica al igual que el establecimiento de caminos ópticos [14] [12] [13].

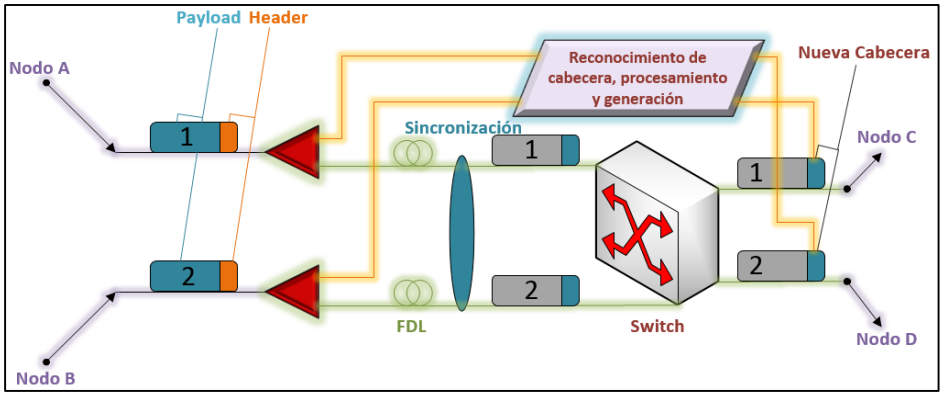

Figura 1.14. Conmutación Óptica de Paquetes.

La gran desventaja de OPS es que hasta ahora no se han desarrollado memorias ópticas que puedan guardar los paquetes mientras se realiza la configuración de transporte en el nodo de red, por lo tanto, es necesario usar líneas de retardo que representan un gran costo y desperdicio de recursos para una red. También debido a que la información no se puede mantener mucho tiempo en líneas de retardo, es necesario que cada nodo este equipado con procesadores más complejos y rápidos de lo normal, lo que también se ve reflejado en costos de la red ya que si se usan procesadores de menos capacidad esto afectaría la estabilidad y el desempeño de la misma.

1.10.4 Comparación y Conclusión

En la tabla 1.1 se muestra la comparación entre los distintos métodos de conmutación óptica mostrados en los apartados anteriores.

Tabla 1.1. Ventajas y desventajas métodos de conmutación.

	Ventajas	Desventajas
OCS	• Manejo eficiente de gran cantidad de tráfico • Fácil de implementar • Bajos costos	• Tendencia a la congestión. • Asignación estática de recursos • Poco eficiente al manejar pequeños volúmenes de tráfico
OBS	• Uso eficiente del ancho de banda • Gran adaptabilidad a cambios en los recursos disponibles y volúmenes de tráfico que maneja	• Necesita de componentes eléctricos en ciertos nodos de la red

OPS	• Uso eficiente de recursos • Gran flexibilidad y manejo de tráfico	• Falta de desarrollo de memorias ópticas • Altos costos para su implementación

Para finalizar es necesario aclarar el uso de OBS para el desarrollo del presente trabajo de grado, debido al gran campo de estudio que abarca y que necesita una investigación más a fondo para un uso más eficiente de redes que usen este tipo de conmutación. Además cabe destacar la importancia que tiene el estudio de la congestión en redes ópticas, ya que dar solución a este inconveniente en la comunicación entre nodos de la red, podría impulsar aún más el desarrollo de esta tecnología, que es el objetivo primordial para su desarrollo.

CAPÍTULO 2

CONMUTACIÓN ÓPTICA POR RÁFAGAS - OBS

2.1 Introducción

OBS es una tecnología que usa conmutación de ráfagas en las redes de comunicación por fibra óptica y su principio es evitar la conversión óptica-eléctrica que se realiza en todos los nodos de la red óptica, todo este proceso de conversión hace que se desaproveche la gran capacidad que tiene la fibra óptica para la transmisión de información debido a los cuellos de botella que se crean en los nodos.

En OBS la transmisión de la información se hace en unidades de transporte llamadas ráfagas ópticas, cada ráfaga está constituida de paquetes que contienen la misma dirección de destino o algún otro parámetro por medio del cual se quiera clasificar la información y realizar el ensamble de la ráfaga. OBS conserva el mismo principio que utiliza la conmutación óptica de paquetes, la diferencia está en que OBS evita demasiado procesamiento al tener que convertir a bits eléctricos solamente un paquete pequeño por ráfaga. La carga útil es transportada en todo su trayecto en el dominio óptico mediante un rápido establecimiento y liberación de las longitudes de onda [4].

Una red OBS está conformada por nodos frontera y nodos centrales, en los nodos fronteras se realiza el ensamblado de los paquetes provenientes de cualquier red que puede ser ATM, Token Ring, Frame Relay, IP, etc., se establecen los canales de transmisión y se forma un paquete de control por ráfaga formada y lista para transmitir.

El paquete de control es enviado por un canal diferente y lleva toda la información de la cabecera de su ráfaga asociada, como es su dirección de destino, dirección de origen, longitud de onda, tamaño y calidad de servicio. En los nodos centrales es recibido, procesado, reconfigurado y transmitido nuevamente hacia el siguiente nodo.

El sistema anteriormente descrito corresponde a una red OBS distribuida, donde todos los nodos se conectan entre sí y cualquier nodo final puede ser a la vez nodo fuente de datos. A diferencia de los sistemas OBS centralizados que cuentan con un único nodo central, de esa forma la información se emite desde un único punto y los nodos finales reciben la información que quieren dar desde ese punto central.

2.2 Nodo Central

2.2.1 Estructura

Los nodos centrales o nodos core se encargan de transmitir los datos en el dominio óptico y procesar los paquetes de control en el dominio eléctrico. Los paquetes de control son los encargados de configurar la matriz de conmutación para que las ráfagas puedan entrar y salir del nodo sin ningún tipo de intervención eléctrica y de este modo permanezcan solamente en el dominio óptico durante todo su viaje.

Un nodo central se compone de los siguientes módulos:

- Una unidad de control de conmutación (SCU*, Switching Control Unit*), ésta crea una tabla de enrutamiento, la cual debe mantener para consultar continuamente los puertos de salida disponibles siempre que llegue un paquete de control, para cuando llegue ésta unidad debe realizar las siguientes acciones:

  - ✓ Interpretar la cabecera.
  - ✓ Planificar el envío de ráfagas.
  - ✓ Detectar y resolver colisiones.
  - ✓ Reescribir la cabecera.
  - ✓ Controlar la conversión de longitudes de onda.
  - ✓ Actualizar la tabla de enrutamiento.

Un conmutador óptico, este módulo sirve únicamente para dar paso a los rayos de luz que llegan a sus puertos de entrada, es responsabilidad de la unidad de control de su configuración y de elegir por cuál de sus puertos de salida saldrá la ráfaga y en caso de que no hayan puertos disponibles, elegir la configuración que depende de las políticas de resolución de contiendas implementado
En la figura 2.1, se muestra la estructura de un nodo central donde los enlaces de entrada son demultiplexados, los datos pasan por la matriz de conmutación y los paquetes de control pasan hacia la unidad de control. Estos paquetes deben primero pasar por una conversión Óptico – Eléctrico, ya en el dominio eléctrico se actualizan, para luego ser sometidos a una conversión Eléctrico – Óptica y seguir su camino hacia otro nodo [15] [16] [17].

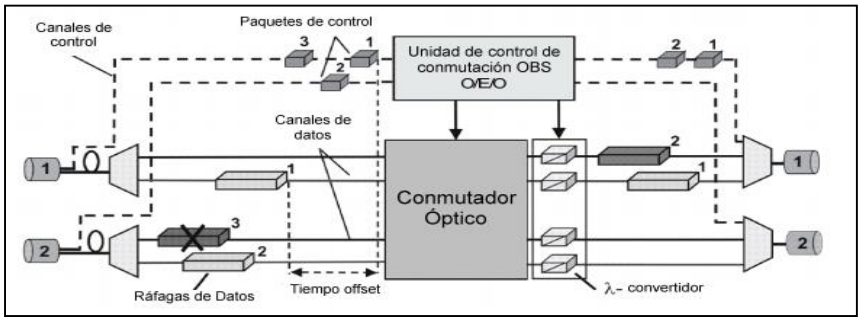

Figura 2.1. Estructura del Nodo Central [15].

2.2.2 Capa Física

A nivel físico se necesitan mecanismos de contención de ráfagas y mecanismos de temporización tanto para las ráfagas de datos como para los paquetes de control.

Es necesario que se especifiquen interfaces para la correcta transmisión de las ráfagas, tanto en los nodos frontera como en los nodos de núcleo. Se deben especificar también los códigos de línea, y especificar la velocidad a la cual se quieren transmitir la información [12] [17].

2.2.3 Capa Enlace de Datos

La capa de enlace de datos se divide en dos partes, una parte de datos y una de control, debido a su principio de funcionamiento donde toda la información de control de la red es transportada fuera de banda por un paquete de control.

Le corresponde al plano de Control, tanto en los nodos de núcleo como de borde la temporización, resolución de contenciones, control de offset, señalización enrutamiento y asignación de longitud de onda. En OBS se debe mantener la información del paquete de control lo más simple posible para evitar demasiado procesamiento del mismo.

En la tabla 2.1 se definen las funciones necesarias en el plano de datos:

Tabla 2.1 Funciones de Nodo de Frontera y Nodo Core [12].

Funciones realizadas en el plano de datos	
Nodo frontera	Encapsulamiento de paquetes en ráfagas
	Separación de las ráfagas en Unidades de Datos de Protocolos (PDU, *Protocolo Data Unit*) de capas superiores.
	Reconocimiento de características como dirección de destino y requerimientos de calidad de servicio.
	Traducción de direcciones de capas superiores (IP) en direcciones OBS
	Asignación del tiempo Offset
	Señalización
Nodo Core	Reserva de recursos
	Configuración de las estructuras de conmutación
	Resolución de contenciones
	Diferenciación del servicio
	Señalización
	Conmutación de ráfagas

2.3 Establecimiento de la ruta

El establecimiento de la ruta o encaminamiento, es el proceso de selección de los nodos intermedios entre el origen y destino por los cuales viaja la ráfaga para poder alcanzar su destino. Por lo tanto el establecimiento de la ruta está ligado al proceso de reserva del camino que utiliza el protocolo adecuado para tal fin en los nodos intermedios. Para que la red pueda alcanzar su objetivo debe ponerse en la tarea de llevar a cabo otros procesos ejecutados por nuevos protocolos, para la obtención de la información topológica y de ingeniería de tráfico y que además cuando haya conocido el estado de la red, pueda obtener la ruta del mejor camino óptico.

El encaminamiento en la red OBS se realiza usando las tablas de enrutamiento donde cada nodo comparte su información topológica de red de forma similar como se hace en las redes de conmutación en el dominio electrónico.

El encaminamiento también podría efectuarse si los nodos de borde obtienen la información de la ruta y la comparten a los nodos core a través del paquete de control, pero esto traería la desventaja de aumento de carga de información a este paquete causando mayor tiempo de procesamiento del mismo.

La forma más eficiente para establecer los caminos para las ráfaga ópticas es mediante el uso de la Conmutación de Etiquetas Multiprotocolo Generalizado (GMPLS, *Generalized Multi-Protocol Label Switching*) que se ajusta perfectamente a las necesidades de OBS ya que su estructura sigue el mismo principio haciendo un uso eficiente del plano de control, además contribuye a la integración con otras redes [12] [16].

2.4 Planificación del canal

Uno de los mayores problemas que enfrentan las redes es la pérdida de información, en OBS, cada nodo debe asignarle a cada ráfaga entrante una longitud de onda apropiada asociada con un puerto de salida para que pueda dirigirse al siguiente nodo y a así poder llegar a su destino. Esta es la planificación que se lleva a cabo en las redes OBS y su principal objetivo es el de minimizar los vacíos en cada canal, un vacío es el espacio que hay entre ráfagas que viajan por un misma fibra con la misma longitud de onda.

Cuando a un nodo llega un paquete de control, se ejecuta un algoritmo de planificación (descritos a continuación) que es el responsable de asignarle a la ráfaga entrante un puerto de salida con un canal de datos [16].

2.4.1 Algoritmos de planificación

- Último Canal Libre sin planificación (LAUC, *Latest Available Unscheduled Channel*)

Este algoritmo lleva un registro del menor tiempo en el cual el canal está libre y listo para poder transmitir una ráfaga de datos, cada vez que se haga una nueva

reserva se actualiza el canal, las longitudes de onda a utilizar no están planificadas, se toma la última que va a estar libre, de esta manera se logra reducir al mínimo los vacíos que quedan entre ráfagas [18] [19].

- LAUC con relleno de vacíos (LAUC-VF, *LAUC void-filling*)

Tiene por objetivo minimizar el vacío existente entre el tiempo de inicio de una nueva reserva y la reserva que le precede. Estos algoritmos tienen que mantener información de los vacíos generados, esta información se obtiene haciendo el cálculo del tiempo de llegada de una ráfaga más su tiempo de duración, si en caso que se deba implementar un mecanismo de contención, se suma el retardo introducido por éste, menos el tiempo de llegada de la siguiente ráfaga [18] [19].

- Espacio Intermedio Minimo (Min-EV, *Minimum Ending Void*)

Este algoritmo se utiliza cuando no se usa fibra de retardo, tiene como objetivo minimizar el vacío entre el tiempo del último bit de la última ráfaga y el inicio de la siguiente, este algoritmo resulta mejor que los anteriores con respecto a la tasa de pérdidas de ráfagas y en el tiempo de programación [18] [19].

2.5 Señalización

Cuando una ráfaga debe ser enviada, se necesita previamente hacer una reserva de recursos en los nodos a donde se dirigirá la ráfaga, para que ésta pueda ser conmutada y nuevamente dirigida al siguiente nodo, a esta reserva anticipada de recursos se conoce como señalización o establecimiento de conexión.

- Justo a Tiempo (JET, *Just Enough Time*)

Es el protocolo de señalización de mayor preferencia para implementar en OBS. Este protocolo únicamente informa que va a llegar una ráfaga a un nodo y establece si ésta se puede encaminar, todo este procedimiento se realiza solo momentos antes de la llegada de la ráfaga sin esperar a que se configuren los conmutadores de longitud de onda [18] [19] [20].

✓ Funcionamiento:

1. En el nodo de frontera se establece la ráfaga y el paquete de control respectivo. El paquete de control lleva la información del tiempo de offset y del tamaño de su ráfaga asociada.

2. Cuando el paquete de control llega al nodo de núcleo éste extrae su contenido; utiliza la información del tiempo de offset y tamaño de la ráfaga para reservar los recursos y configurar la estructura de conmutación. Ahora este nodo, ya conociendo el tiempo en que llegará la ráfaga, reserva los recursos un momento muy pequeño antes de que llegue la ráfaga.

3 El paquete de control se actualiza y se disminuye su tiempo de offset, dependiendo de su tiempo de procesamiento y se envía al nodo siguiente realizando el paso 2. Si el paquete al llegar a un nodo de núcleo no encuentra recursos y éste implementa un mecanismo de contención, entonces el offset debe ser actualizado con un tiempo mayor que depende del tiempo empleado en la contención.

4 Si en el nodo no existen un mecanismo de contención y no se ha podido asignar recursos entonces la ráfaga será descartada y se interrumpe el proceso de señalización, por consiguiente los próximos nodos no reservarán ningún recurso.

5 Con la información de duración o tamaño de la ráfaga, se configura la estructura de conmutación para que libere los recursos después de que la ráfaga ha dejado el nodo, luego de esto continúa con el proceso de asignación de recursos para otra ráfaga.

En la figura 2.2 se muestra el funcionamiento de este protocolo de una forma más didáctica.

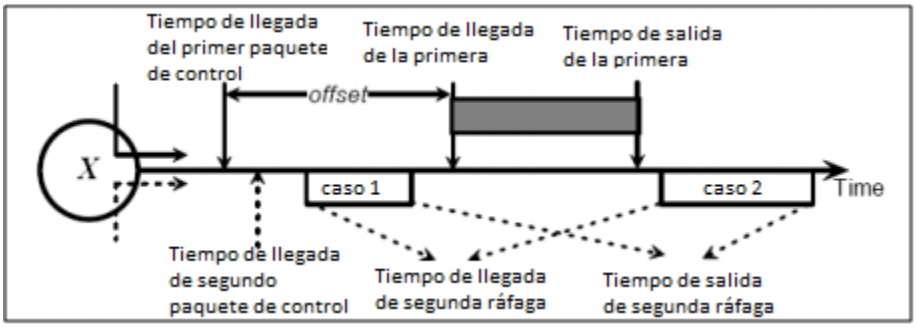

Figura 2.2. Funcionamiento del protocolo JET [12].

2.6 Métodos de Control de Contención

En una red electrónica convencional la mayoría del tiempo se presenta congestión de datos en los nodos, lo que lleva a una saturación en la red y que se ve reflejado en pérdida de paquetes, tiempos de retardo elevados, entre otras dificultades, que finalmente causan una desestabilización de la red. Es por esto que se han desarrollado maneras de evitar este tipo de incidentes y es así como se crean los métodos de control de contención cuya función es mantener a la red y los nodos que la conforman, siempre con un margen de disponibilidad suficiente para atender todo el tráfico que por ella transita. Pues lo mismo ocurre en las redes ópticas, también se presenta congestión en la red, pero el problema es que no se pueden usar los mismos métodos de contención que en redes electrónicas, ya que no existen memorias ópticas que puedan guardar información y por lo tanto imposibilita almacenar información por cierto tiempo mientras la red se descarga

un poco como ocurre normalmente en las redes electrónicas que simplemente guardan la información en memorias por cierto tiempo y luego la envían.

Debido a esto, se han desarrollado otros métodos para resolver los problemas en las redes ópticas que son un poco más complejos que los usados en las redes electrónicas, pero que al final, intentan realizar la misma operación y mantener la red en funcionamiento.

A continuación se presentan los métodos de control de contención en los que más se ha trabajado y que son más relevantes en estas redes.

2.6.1 Líneas de Retardo

En las redes comunes por conmutación de paquetes electrónicos, es posible guardar la información en las memorias que poseen los conmutadores, las cuales pueden almacenar los datos durante un tiempo indeterminado mientras que sea posible realizar el envío de dicho paquete. En el dominio óptico no se puede contar con este tipo de memorias debido a que aún no se desarrolla un buffer capaz de guardar datos en forma de “luz” es por esto que se usan otro tipo de métodos para retardar el envío de estos paquetes cuando sea necesario.

Una línea de retardo de fibra (FDL, *Fiber Delay Line*) es usada como equivalente de una memoria RAM en redes ópticas, sin embargo, no cuenta con las mismas características que proporciona una memoria RAM. La FDL consiste en una simple línea de fibra que se coloca en la red para retardar en cierta medida la señal que se necesita enviar. Cerca de 200km de fibra son necesarios para retardar un paquete de información 1ms [21] [22].

Según su configuración los buffer FDL se clasifican en buffers de retardo fijo y de retardo variable. Los Buffers de retardo fijo son simplemente líneas de retardo conectadas en serie en una red como se muestra en la figura 2.3.

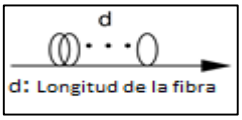

Figura 2.3. Buffer FDL de Retardo Fijo [21].

Por otro lado, los buffers de retardo variable son líneas de retardo conectadas en paralelo, a diferencia del Buffer de retardo fijo, este se encuentra constituido por conmutadores 2x2 y una línea de retardo fijo como se puede ver en la figura 2.4. La ráfaga puede pasar por la línea de retardo fijo o la línea sin retardo en su recorrido por el Buffer variable, esto genera la posibilidad de obtener retardos variables como se muestra en la figura 2.4, sin embargo realizar esta configuración es más compleja y costosa que la obtenida con los Buffer de retardo fijo.

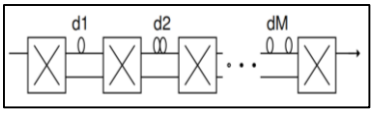

Figura 2.4. Buffer FDL de retardo Variable [21].

Según su arquitectura los buffer FDL se clasifican en pre-alimentado, retroalimentado y arquitectura hibrida. En la arquitectura de Buffer pre-alimentado, cada línea de retardo conecta un puerto de salida de un conmutador, con un puerto de entrada del conmutador en la siguiente etapa, la ráfaga que llega a la salida del conmutador puede ser transmitida o descartada. Por otro lado en la arquitectura de Buffer retroalimentado, las ráfagas que son enviadas a través del conmutador son bloqueadas a la salida y se envían de nuevo a la entrada del mismo conmutador mediante una FDL, de este modo, las ráfagas circularan varias veces por el mismo hasta que sean transmitidas con éxito o descartadas como se muestra en la figura 2.5.

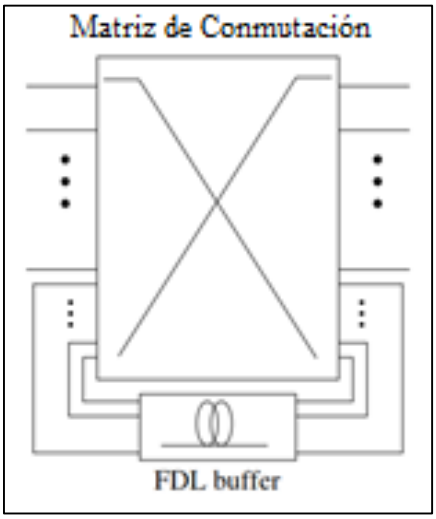

Figura 2.5. Buffer FDL Retroalimentado [21].

De acuerdo a la posición de los buffer, la conmutación de paquetes se divide en dedicada y compartida. En la configuración de FDL dedicada, un módulo de almacenamiento separado es dedicado a cada puerto de salida del conmutador, como se puede ver en la figura 2.6. Mientras que en la configuración FDL compartida, un módulo de almacenamiento puede ser compartido por algunos o todos los puertos de salida del conmutador como se muestra en la figura 2.7.

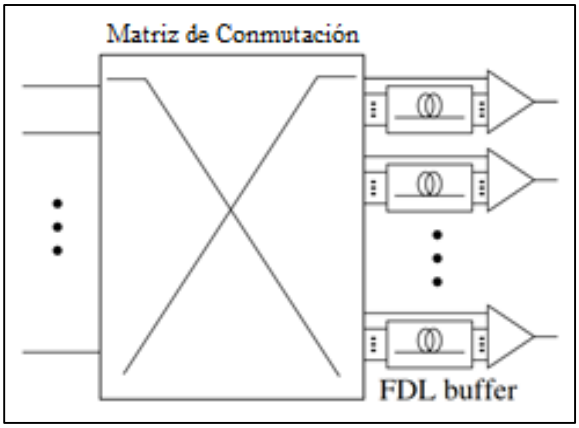

Figura 2.6. Conmutador con Buffer FDL dedicado y pre-alimentación [21].

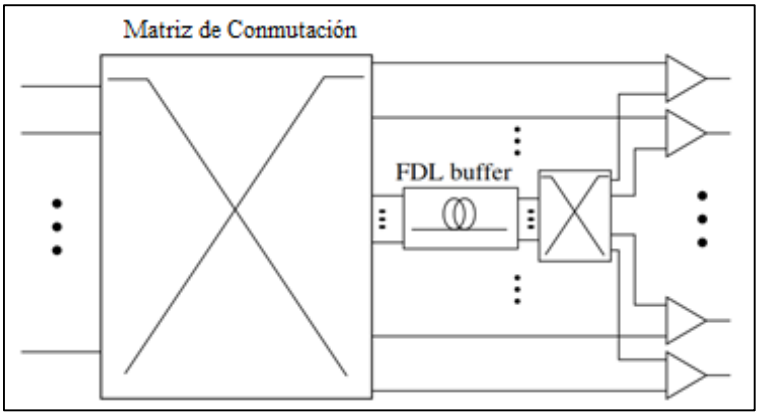

Figura 2.7. Conmutador con Buffer compartido y pre-alimentación [21].

2.6.2 Conversión de Longitud de Onda

Un enlace de fibra óptica ofrece la capacidad de usar distintos canales sobre el mismo enlace, esto debido a que se pueden utilizar varias longitudes de onda para transportar información. Sin embargo, en una red OBS se puede dar que dos o más ráfagas intenten usar el mismo canal de salida al mismo tiempo, lo que generaría un conflicto en dicho canal. Aunque suceda esto, gracias a la capacidad de la fibra se puede transmitir cada ráfaga sobre distintas longitudes de onda y así evitar dicho conflicto.

Con el fin de reducir la contención de las ráfagas se usan conversores de longitudes de onda en los conmutadores, cuya función es convertir la longitud de onda de una señal entrante a una diferente longitud de onda saliente. Esto permite aumentar el reúso de longitudes de onda entre un 10% a 40% con baja disponibilidad, ya que una misma longitud de onda podría llevar diferentes ráfagas en los distintos enlaces de fibra de la red [21] [22].

Los conversores de longitud de onda se dividen en dos tipos: fijos y ajustables. En los conversores de tipo fijo, cada canal entrante puede ser conectado a uno o más canales de salida predeterminados. Por otro lado, los conversores de tipo

ajustable tienen la capacidad de convertir cualquier longitud de onda entrante a cualquier longitud de onda saliente, estos conversores tienen un rango limitado de conversón, ya que la longitud de onda solo puede ser convertida a una longitud de onda adyacente.

Estos conversores son usados en los nodos centrales con distintas configuraciones. La más simple conocida como *conversión completa de longitud de onda,* en la que cualquier longitud de onda en la entrada del conmutador puede ser desplazada a cualquier longitud de onda en la salida del mismo.

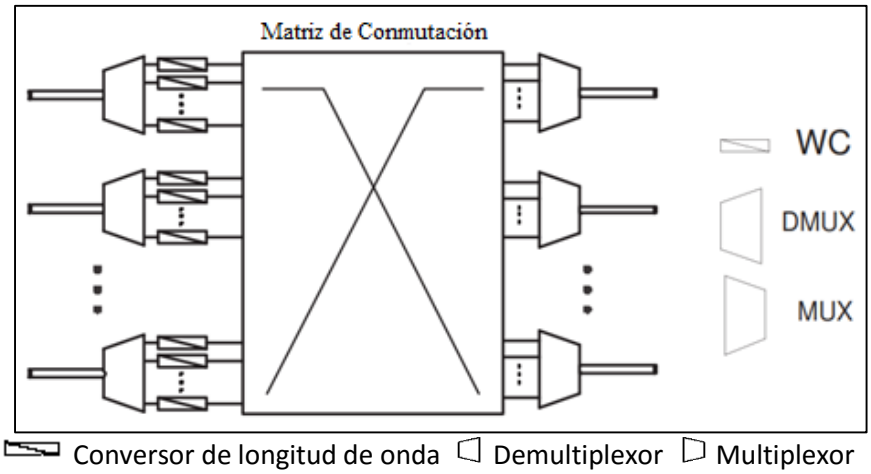

Figura 2.8. Nodo OBS con conversión completa de longitud de onda en los puertos de entrada [21].

Gracias a su capacidad para poder reutilizar las longitudes de onda, este tipo de configuración disminuye de gran manera la tasa de perdida en redes OBS lo que permite un mejor desempeño de la misma. Por otra parte, los costos de implementar este sistema son muy altos debido a que se necesita de un conversor en cada puerto del conmutador.

También es posible usar de manera parcial algún tipo de conversión de longitud de onda, es decir, que un conmutador pueda disponer de los distintos tipos conversores y sus configuraciones como lo son fija, completa, limitada y sin conversión. A este tipo de sistema se le conoce como *conversión parcial de longitud de onda*, y en comparación con *la conversión completa de longitud de onda,* tiene como ventaja que con un menor número de conversores puede lograr un mayor rendimiento reduciendo la tasa de bloqueo en el nodo. En la figura 2.9 se puede observar un nodo OBS con este tipo de sistema.

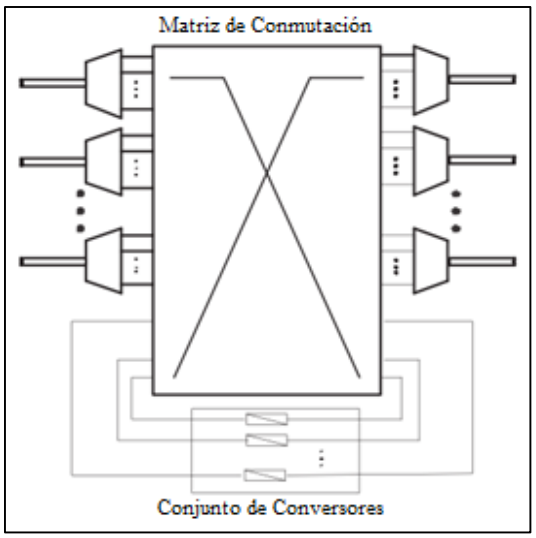

Figura 2.9. Nodo OBS con conversión parcial de longitud de onda [21].

2.6.3 Desviación por Enrutamiento

Los mecanismos de contención usados en una red OBS generalmente necesitan que se le añadan ciertos elementos a la red o a los nodos de la misma para poder realizar el proceso de contención de una manera eficaz. El uso de estos elementos resulta efectivo para solucionar dichos problemas de congestión, sin embargo, al implementarlos se tiene como desventaja que conformar este tipo de redes consuma una gran cantidad de recursos y haga que todo el sistema sea más robusto. Por estas razones, se toma como alternativa el uso de la misma red para la contención de las ráfagas, por medio de la desviación por enrutamiento.

La desviación por enrutamiento consiste en usar los enlaces de la misma red para mejorar el rendimiento de la misma, al desviar las ráfagas que probablemente irán a un canal congestionado hacia otro canal que este desocupado. Sin embargo, no es tan sencillo solo pasar de un canal a otro debido a que la ráfaga podría perder su dirección de destino, es por eso, que a medida que la ráfaga va recorriendo la red se va restableciendo poco a poco la ruta de esta para que al final llegue a su destino. Además el nodo que decida desviar alguna ráfaga debe realizar una verificación de la trayectoria, con el fin de establecer en qué medida aumentara el tiempo del recorrido de la ráfaga hasta llegar a su destino, lo que permite añadir un tiempo de offset a esta para que no vaya a alcanzar el paquete de control generando la pérdida total de la información.

Gracias al uso de este tiempo de offset todos los nodos que atraviese la ráfaga, podrán procesar correctamente el paquete de control y así manejarla debidamente hasta que llegue a su destino [21] [22].

2.6.4 Segmentación de la Ráfaga

Generalmente los métodos utilizados para realizar la contención de ráfagas en una red OBS, tienden a descartar una ráfaga en su totalidad cuando no pueden dar

solución a dicho problema como se muestra en la figura 2.10. En el mejor de los casos, es necesario realizar el reenvío de la ráfaga descartada, generando pérdida de tiempo y de recursos, que afectan tanto la comunicación entre los nodos como de la red en general.

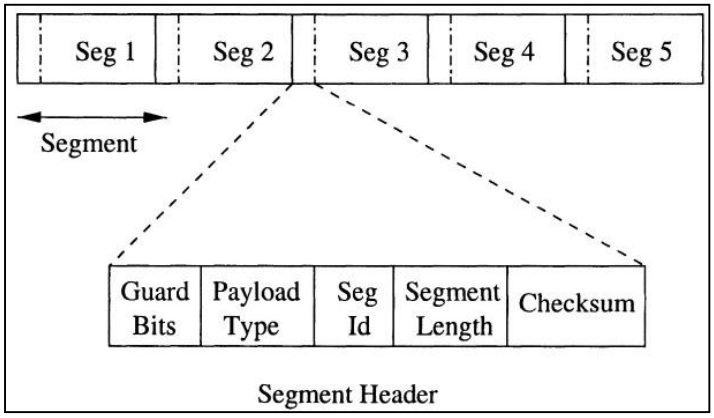

Figura 2.10. Segmentación de Ráfaga [22].

Es así como se tiene la opción de utilizar la segmentación de ráfaga, para solventar este tipo de problemas y realizar el control de contención conjuntamente. Este método de contención disminuye en gran medida la perdida de paquetes, debido a que se dividen la ráfaga en segmentos, donde solo se descartan los segmentos de ráfaga que se superponen con otros segmentos de otras ráfagas, evitando así que se elimine la ráfaga en su totalidad como se puede ver en la figura 2.11.

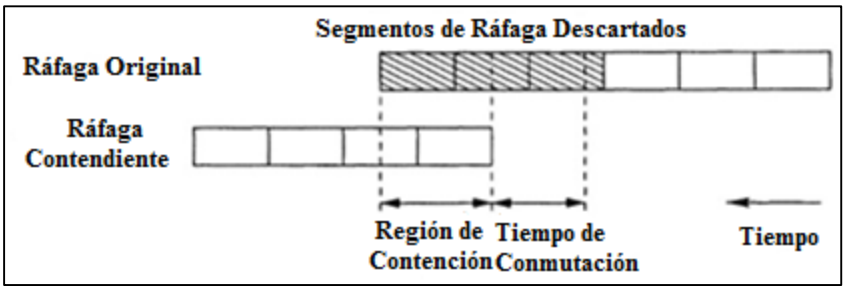

Figura 2.11. Selección de segmentos descartados para dos ráfagas superpuestas [22].

Estos segmentos están compuestos por una cabecera y una carga útil, en la cabecera se encuentra información de fuente y destino, tamaño del segmento, bits de sincronización e información de corrección de errores. Por otra parte, la carga útil puede llevar algún tipo de datos, como paquetes IP, tramas Ethernet o celdas ATM [21] [22].

2.7 Metaheurísticas

Las metaheurísticas proporcionan un aporte esencial para el desarrollo de algunos componentes de la red, debido a que permitirán solucionar de manera eficiente las dificultades de enrutamiento y contención necesarias para el funcionamiento de la red. Por lo tanto se hace un énfasis en las características de cada tipo de metaheurística, con el fin de establecer con cuál de ellas se tendrá un mejor rendimiento del sistema y proporcionara mayor estabilidad.

Son un tipo de algoritmos avanzados que permiten o con los que se intenta buscar solución a algún problema de gran complejidad. Su definición viene desde el término heurística que son soluciones creadas para algún tipo de problema específico que dará una solución satisfactoria a dicho problema, el término “meta” agregado quiere decir que es algo más complejo y que por lo tanto va más allá al momento de resolver problemas, es decir, abarca una mayor cantidad de métodos para tratar algún problema y proporciona más soluciones posibles para a resolución del mismo.

Este tipo de algoritmos más complejos implican cierto grado de evolución y comprensión, lo que se podría ver como tener cierto grado de conciencia del entorno y del problema para el que se han diseñado, lo que se conoce como inteligencia artificial. La importancia de estos algoritmos radica en que pueden encontrar soluciones a problemas que normalmente sin el uso de estas tardaría una gran cantidad de tiempo, ya que posiblemente no habría ninguna base de donde iniciar la resolución del mismo. En cambio, las metahurísticas proveen cierto tiempo de “conocimiento” previo para la solución del problema, por lo que solo es necesario entender el problema y establecer algunos datos importantes que después de todo llevaran a la solución de este de una forma eficiente y en menor tiempo. El éxito, al usar estos algoritmos implica en la aleatoriedad que usan para encontrar la solución a su tarea específica, es así como puede estudiar gran cantidad de posibilidades en tiempo relativamente cortos, logrando con esto un desarrollo avanzado en infinidad de procesos que al final conllevan a la evolución de todos los sistemas que usan estos algoritmos [23] [24].

2.7.1 Tipos de Metaheurística

Como en todo gran sistema, las metaheurísticas también se dividen en distintos tipos dependiendo de su función, de los datos que manejan, las soluciones que presentan, los métodos que usan, etc. Sin embargo, aún se siguen desarrollando otros tipos de algoritmos que podrían mejorar la eficiencia y rendimiento. Por ahora se dará una explicación de los tipos de metaheurísticas más relevantes y desarrollados hasta ahora, que implican una gran evolución en la inteligencia artificial para la solución de procesos complejos y que muestran el manejo que cada clase le da a la información para llegar a un resultado satisfactorio [23] [24] [25].

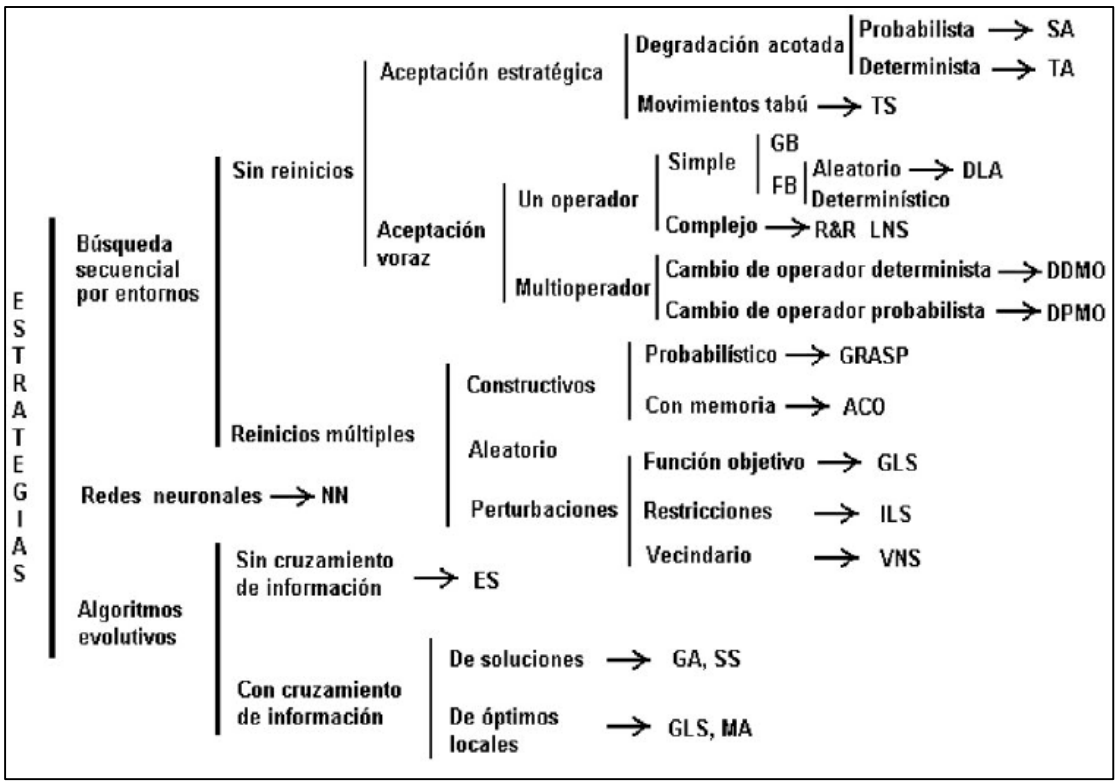

Figura 2.12. Estrategias de Resolución de Problemas (Metaheurísticas) [26].

En la tabla 2.2, se nombran algunos tipos de metaheuristicas.

Tabla 2.2 Tipos de Metaheurística.

Siglas	Nombre
SA	Recocido Simulado
TA	Aceptación por Umbrales
FB	Primero la mejor
DLA	Agregación Limitada por Difusión
LNS	Búsqueda por Grandes Entornos
DMO	Optimización basada en Malla Dinámica
GRASP	Procedimientos de Búsqueda basados en funciones Greedy Aleatorizadas

ACO	Optimización por colonia de Hormigas
GLS	Búsqueda Local Guiada
ILS	Búsqueda Local Iterada
VNS	Búsqueda de Entorno Variable
GA	Algoritmos Genéticos
SS	Búsqueda Dispersa

2.7.1.1 Metaheurística de Relajación

Generalmente para darle solución a un problema se trabaja únicamente en una única posibilidad para obtener un resultado satisfactorio y que cumpla con las expectativas esperadas. Sin embargo, no siempre es posible obtener una solución apropiada y en algunos casos es demasiado compleja como para implementarse, es por esto, que se utilizan métodos de relajación para llegar a dar solución a dicho problema. Este método consiste en crear o utilizar modelos relajados, es decir, con variaciones del modelo original en los que se puede eliminar o modificar algunas condiciones del problema inicial. Entonces al dar solución a alguna de estas ***relajaciones***, prácticamente se está dando solución al problema real o de alguna forma puede llevar a un método más fácil para su resolución.

2.7.1.2 Metaheurísticas Constructivas

Este método proporciona la solución a un problema, por medio de la construcción de una estructura que representa la solución al mismo. Esta estructura se va conformando poco a poco por elementos que se van seleccionando dependiendo de las condiciones del problema y la solución a la que se espera llegar.

Existen algunas estrategias en las que se determina llegar a la solución de una manera rápida sin la necesidad de agregar demasiados elementos a la estructura, aunque esto implique una mayor complejidad para llegar a la solución. Una de estas estrategias es conocida como voraz o greedy.

2.7.1.3 Metaheurísticas de Búsqueda

Considerada el tipo más importante de las metaheurística debido a que su modelo de solución implica la regla primordial de las heurísticas, que es buscar la solución a un problema y mejorarla mientras sea posible hacerlo. Entonces el método que implementan las metahehurísticas de búsqueda para resolver un problema, consiste en recorrer un conjunto de soluciones que se establece sobre un espacio de búsqueda. Generalmente, este espacio de búsqueda se divide en tres tipos dependiendo de la “población” en el que se busquen las diversas soluciones conocidas como metahurística de búsqueda local y metaheurísticas de búsqueda global. El primer tipo realiza un análisis local de las posibles soluciones y toma la

mejor de ellas siempre y cuando pueda tener alguna mejora, pero manteniéndose en su entorno considerando siempre un óptimo local. El segundo tipo va más allá de un óptimo local, ya que se ha desarrollado para escapar de esa restricción e ir más allá en la búsqueda de una solución, esto lo logra mediante el reinicio de la búsqueda pero con otra solución, también modificando el entorno en el que está buscando la solución y transformando las otras soluciones posibles pero que fueron descartadas por no tener una mejora.

2.7.1.4 Metaheurísticas Evolutivas

Es el tipo más complejo de metaheurística ya que la forma en la que llega a la resolución de un problema consiste en fomentar la evolución en el conjunto de soluciones, generando una interacción entre estas soluciones, también llamado población, con el fin de acercarse a una solución óptima para el problema.

Este tipo de metaheuríticas se clasifican dependiendo de la forma en que combina la información obtenida por cada tipo de solución, y se conocen como procedimientos aleatorios o sistemáticos.

2.8 Conclusión del capítulo

Por lo anterior se encamina el estudio hacia los componentes necesarios para el desarrollo de una red OBS en general, sin embargo, el sistema a simular es una red OBS distribuida por lo cual es necesario amoldar los módulos presentados para que puedan funcionar correctamente y además identificar los parámetros necesarios para realizar las diversas funciones dentro de la red.

También es importante establecer si el uso de metaheurísticas es necesario o aportaría alguna ventaja en el desarrollo o solución del problema que se busca abarcar, por lo que el análisis del funcionamiento de estos algoritmos puede proporcionar información importante con relación al posible uso de alguno de estos en el desarrollo del sistema. Es así como se plantea el uso de metaheurísticas de búsqueda para el desarrollo del mecanismo de control de contención por deflexión de ruta, ya que al momento que se presente una contención se debe realizar un cambio de ruta de la ráfaga, lo que implica realizar inicialmente la búsqueda de las posibles rutas alternas que puede tomar y posteriormente escogiendo la mejor de ellas, que en este caso sea la ruta alterna más corta hacia el destino.

En conclusión los temas tratados en esta sección proporcionan una base firme sobre los conocimientos necesarios para desarrollar un sistema que sea capaz de simular una red OBS distribuida con control de contención y en la que se puedan configurar parámetros necesarios para obtener información detallada sobre el comportamiento de la misma.

Haciendo énfasis en esto, la información obtenida da pie para realizar la planificación e implementación de un entorno de simulación cuya explicación se realizara en el siguiente capítulo.

CAPÍTULO 3

DESARROLLO Y SIMULACIÓN

3.1 Introducción

En este capítulo se describe la metodología empleada para la realización de este proyecto, la cual tiene como objetivo realizar un estudio sobre los diferentes mecanismos de control de contención en una red OBS distribuida. Se inicia el proceso con el diseño de una red de transporte de conmutación de ráfagas, en la cual se simula el flujo de los mensajes, se hace luego una comparación del comportamiento de la red al aplicarse a los nodos centrales tres mecanismos de control de contención diferentes.

El modelo general mostrado en la figura 3.1 que se tiene en cuenta cumple con las secciones de planificación, diseño y simulación del sistema que se describen a continuación.

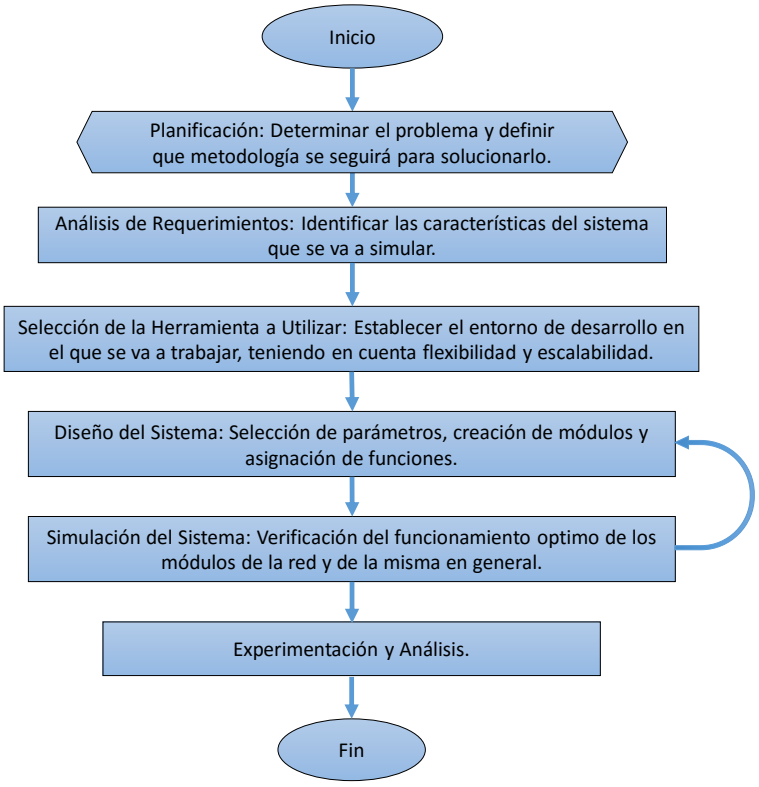

Figura 3.1. Metodología propuesta para el desarrollo y simulación del sistema

3.2 Planificación

La pérdida de ráfagas se presenta cuando hay bloqueo en su planificación por falta de recursos. Éste es un problema que se resuelve en gran medida aplicando en el núcleo de la red algún mecanismo de control de contención.

Existen varios mecanismos para el control de congestión, en este trabajo se analizan tres: desviación por deflexión de ruta, cambio de longitud de onda y líneas de retardo. Cada mecanismo tiene el mismo objetivo, evitar la pérdida de ráfagas, pero también tienen sus desventajas; el trabajo se encuentra entonces en elegir cuál de ellos es el que presenta el comportamiento más acertado para determinadas características de la red y cuales consecuencias trae su aplicación. Se procede entonces, a realizar un estudio independiente de cada uno de ellos, variando diferentes parámetros y comparando resultados para obtener el mecanismo que mejor se adapte a las necesidades de la red.

3.2.1 Análisis de Requerimientos

El presente proyecto basa su topología en el modelo de red de transporte más grande de Europa llamado GEANT2; en esta red se definen los mecanismos que hacen posibles el transporte de los mensajes y se aplican sobre la misma los mecanismos de control de contención de ráfagas en el núcleo de la red [27].

La red GEANT2 cuenta con 25 nodos que interconectan 34 países Europeos, como se puede ver en la figura 3.2. Ha sido diseñada para soportar alto tráfico con canales de gran ancho de banda con enlaces de fibra óptica alcanzando los 10Gbps, interconectados mediante conmutadores y enrutadores, con topología estrella extendida y que soporte los protocolos IPv4 e IPv6.

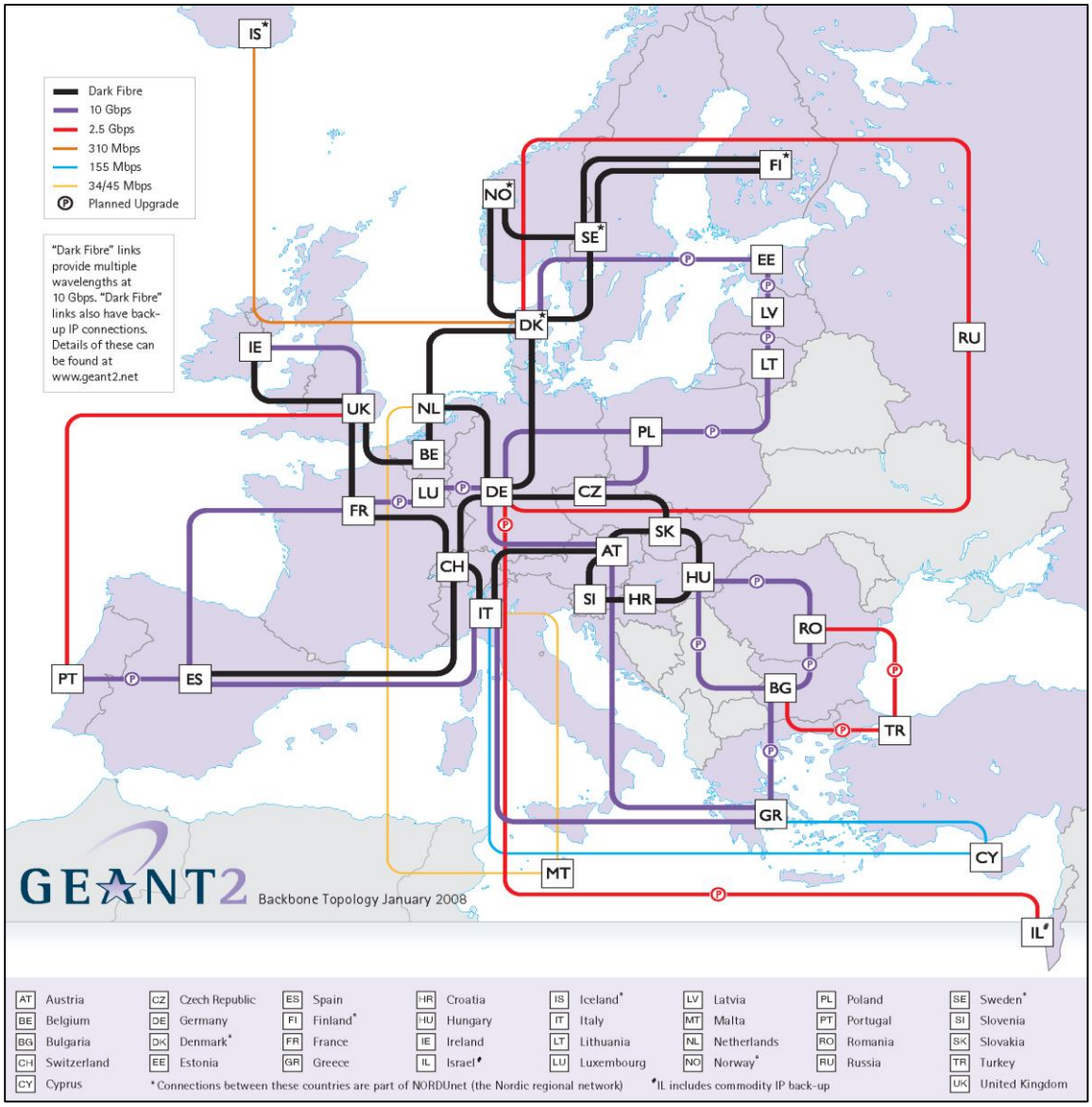

Figura 3.2. Topología de la red GEANT2 [27].

Esta tecnología de red se desarrolla para permitir la gestión de proyectos colaborativos con una gran rapidez, se ofrece para el desarrollo y pruebas de las nuevas tecnologías de la información en las cuales se abarca la telemedicina, conferencias virtuales de índole pedagógico, acceso a equipos remotos, entre otros.

La red simulada cuenta con módulos simples y módulos compuestos los cuales describen los diferentes elementos que componen su topología, es en éstos módulos donde se obtienen los datos estadísticos requeridos para deducir las observaciones y sacar conclusiones.

3.2.2 Selección de la herramienta a utilizar

Se ha seleccionado como Entorno de Desarrollo Integrado (IDE, *Integrated Development Environment*) la herramienta de simulación OMNeT++ para el desarrollo del presente trabajo dada su conveniencia, puesto que la red definida en este simulador permite ver fácilmente la transferencia de los datos tanto de información como de señalización ya que éstos se modelan como mensajes de simulación a eventos discretos.

Con respecto al diseño y modelado de la topología de red, OMNeT utiliza el lenguaje NED, lenguaje de alto nivel para la creación de los módulos, con éste se puede definir de una forma sencilla la topología, los canales y sus parámetros, mensajes, conexiones, etc.

Para la programación el IDE utiliza el lenguaje C++, éste es un lenguaje muy utilizado en el desarrollo de software, ya que tiene una gran flexibilidad y además cuenta con un soporte bastante amplio para su implementación.

Su funcionamiento modular permite facilidad de organización y brinda flexibilidad para el diseño de la red. Se puede crear un módulo simple que simule un generador/receptor de tráfico, un conmutador o cualquier otro elemento que se deba utilizar dentro de la red.

OMNeT++ permite hacer simulaciones simultáneas consecutivas y rápidas, esta funcionalidad permite ahorrar tiempo a la hora de extraer datos estadísticos los cuales se utilizan en el análisis de resultados y comprobación de algoritmos.

3.3 Diseño del Sistema

Inicialmente se plantea un método generalizado, en el que se muestra de manera específica como trabaja el sistema y que se puede ver en la figura 3.3, el cual permite establecer las funciones necesarias que se deben considerar para el diseño de los módulos de la red.

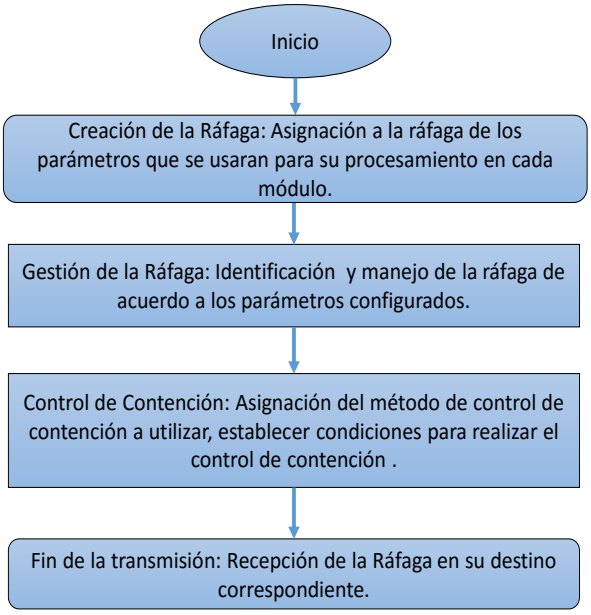

Figura 3.3. Diagrama de flujo general del funcionamiento del sistema.

Inicialmente se crea una ráfaga, a la que se le asigna un paquete de control que debe contener información necesaria (fuente, destino, tamaño, prioridad, entre otros) para el posterior tratamiento de la ráfaga en los nodos de la red. Después de ser creada la ráfaga se envía e ingresa al nodo que se encargará de su enrutamiento, el cual "lee" toda la información correspondiente al proceso que va a realizar y examina de manera rápida si los recursos necesarios para el envío de la ráfaga se encuentran disponibles en el momento, posteriormente la ráfaga es tratada por un nodo de control de contención que es el responsable de que todo el tráfico que recorre la red se mantenga sin ningún tipo de conflicto o bloqueo que pueda generar errores en el correcto funcionamiento de la red. Finalmente, después de ser manejada por el módulo de control de contención la ráfaga es enviada a su destino.

3.3.1 Componentes de Red OBS

- Nodo Hibrido

Debido a las características de la red a simular se trabajara con nodos híbridos que cumplen las funciones de nodos núcleo y de frontera en una red OBS normal, esto se debe a que la red con la que se trabajará se conoce como red OBS distribuida con topología irregular y todos los procesos se realizan en los nodos dentro de la red, a diferencia de las redes comunes que son centralizadas. Es así como este nodo se encarga de gestionar la ráfaga que es recibida de los módulos app, y por ende cada módulo app tiene su respectivo nodo hibrido que se puede observar en la figura 3.4, y a su vez todos los nodos híbridos se encuentran

interconectados entre sí, por lo que la topología de la red se consideraría como una red en malla.

Figura 3.4. Nodo Hibrido en red OBS simulada.

Entonces al ingresar al nodo, las ráfagas son clasificadas de acuerdo con los datos obtenidos de la misma, y posteriormente es enviada a su destino correspondiente.

- Módulo App (fuente/destino)

Este módulo que se muestra en la figura 3.5, representa un usuario o una fuente de paquetes en una red OBS, encargado de generar los datos que posteriormente se enviará. Tiene una conexión bidireccional lo que le permite enviar y recibir datos de la red, también proporciona al paquete de datos la información necesaria para ser gestionado por el nodo híbrido y así poder llegar a su destino. Al enviar una ráfaga y llegar satisfactoriamente a su destino, se muestra una señal de confirmación para mostrar que el envío y la recepción se hicieron de manera correcta.

Figura 3.5. Módulo App en red OBS simulada.

- Módulo de Enrutamiento

El módulo de enrutamiento que se muestra en la figura 3.6, tiene una conexión de entrada y una conexión de salida, la primera se encuentra conectada al app y es bidireccional debido a que por allí le llegarán los mensajes que envía el app, pero por esta misma también enviará hacia el app la información que le corresponde, es decir, los paquetes que lo tengan configurado como su destino. Este módulo es uno de los más importantes ya que direcciona los paquetes hacia su destino en toda la red, y es el que se encargará de que estos no se pierdan entre los nodos del sistema, debido a que en algunas ocasiones es necesario pasar por varios nodos antes de llegar a su destino final.

Figura 3.6. Módulo de Enrutamiento en red OBS simulada.

- Módulo de Clasificación

En la figura 3.7 se muestra el módulo que se encarga de clasificar los paquetes dependiendo de su longitud de onda, con el fin de poder realizar un mejor control del uso de estas, y también para posteriormente ser enviadas al módulo de contención que gracias a esta clasificación podrá tratar de una manera más eficiente la información.

Figura 3.7. Módulo de Clasificación en red OBS simulada.

- Módulo de Control de contención

Después de ser clasificado, el paquete llega al módulo de control de contención que se muestra en la figura 3.8, donde se maneja de tal manera que no se produzca congestión en la red, puesto que los recursos necesarios para su envío pueden estar parcial o totalmente ocupados.

Aquí los paquetes son sometidos a algún mecanismo de control de congestión que permita su posterior envío sin que esto afecte la integridad de la información que lleva, evitando colisiones o pérdidas de información al traslaparse con otros paquetes.

Figura 3.8. Módulo Control de contención en red OBS simulada.

- Módulo de Encolamiento

El módulo de encolamiento que se muestra en la figura 3.9, es el módulo de salida del nodo hibrido por medio del cual salen las ráfagas generadas por el nodo, como también las ráfagas que atraviesan este, pero tienen un destino diferente, además es el primer módulo al que llegan las ráfagas cuando entran al nodo. Se encarga de verificar el destino que tiene cada paquete para reenviarlo posteriormente, además tiene dos estados, disponible y ocupado, que funcionan de tal manera que cuando esté disponible envía la ráfaga sin contravenciones, en cambio cuando su estado es ocupado pone las ráfagas en una cola y las va enviando una a una. Por cada conexión que tenga el nodo principal, se tendrá un nodo de encolamiento

que realizará el manejo de la información que ingrese o salga por dichas conexiones y que posteriormente será gestionada por el nodo de enrutamiento en el caso que sea información entrante.

Figura 3.9. Módulo de Encolamiento red OBS simulada.

3.3.2 Módulos de Contención

- Módulo Línea de Retardo de Fibra

Este nodo simula una línea de retardo de fibra óptica, mediante el uso de un retardo en la salida del mismo. Entonces, la ráfaga que ingresa al módulo, experimenta un retardo antes de ser enviado por el mismo al siguiente módulo, este retardo es establecido con anterioridad en un tiempo base para todo el sistema y se puede variar para observar cómo afecta esto la comunicación en la red, ver figura 3.10.

Figura 3.10. Módulo FDL en red OBS simulada.

- Módulo Cambio de Longitud de Onda

Todas las ráfagas que recorren la red, tienen consigo una longitud de onda asignada desde su creación y esto simula los diferentes canales por los que la información es enviada en una red real. El módulo de cambio de longitud de onda mostrado en la figura 3.11, detecta cuando un canal (longitud de onda) se encuentra ocupado para el envío de la información y es en este momento cuando realiza el cambio de longitud de onda asignada a la ráfaga por otra que se encuentre disponible para su uso.

Figura 3.11. Módulo cambio de longitud de onda en red OBS simulada.

Finalmente realiza un conteo de cuantas veces ha sido usada una longitud de onda y para evitar el uso excesivo de esta, omite su uso por un tiempo aleatorio mientras se descongestiona.

- Módulo Deflexión de Ruta

Uno de los métodos de control de contención más complejo es el de deflexión de ruta ya que se debe realizar un manejo y estudio de los recursos disponibles de la red más detallado. Además, este módulo prácticamente debe trazar una nueva ruta en el caso que haya congestión, lo que requiere un nivel de procesamiento mayor ya que se debe estudiar todos las posibles rutas y elegir la más conveniente para el envío de la ráfaga. En el sistema de simulación, cada nodo hibrido tendrá su propio módulo de contención, por lo que habrá una mayor flexibilidad y un mejor manejo en el procesamiento para elegir el camino en el que se pueda tener un rendimiento más eficiente, la figura 3.12 muestra la representación de este módulo en la red.

Figura 3.12. Módulo Deflexión de ruta en red OBS simulada.

3.4 Simulación del sistema

El sistema general que se va a simular es una red OBS distribuida mostrada en la figura 3.13, con la diferencia que se plantea un entorno de simulación diferente, para cada uno de los mecanismos de control de contención.

Puesto que la distribución de los nodos y sus conexiones depende de la presentada en la topología de la red GEANT 2, es necesario establecer que nodos de la red real se van a usar para la simulación, ya que en todos no se pueden obtener resultados efectivos del desempeño de la red, por esta razón se escogen los nodos que tengan más conexiones entre ellos, suponiendo que presentaran mayor congestión. Ya que se han escogido los nodos, se procede a asignarle a cada uno, el número de conexiones que tendrán entre si y que deben funcionar como puertos, que permitan el envío y recepción de información por el mismo enlace, lo que implica manejar en la lógica de los nodos el identificador que se asignara a cada compuerta para tener un correcto funcionamiento del sistema.

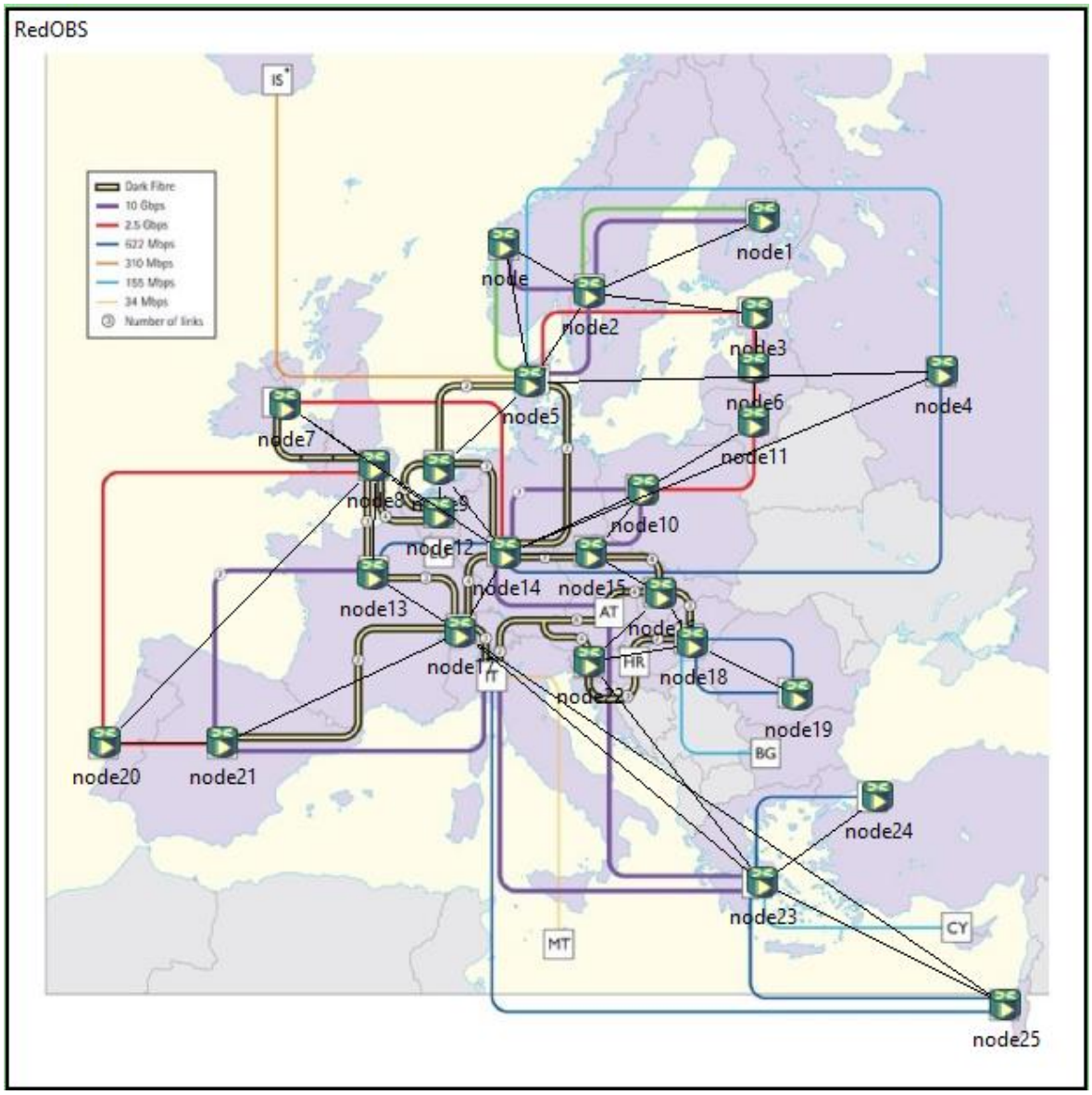

Figura 3.13. Red OBS distribuida simulada en OMNeT++.

En la simulación inicial cada nodo de red envía varios paquetes para verificar el correcto funcionamiento de la red. Después de verificar que cada nodo de red está trabajando correctamente se procede a realizar cambios en las configuraciones del sistema para obtener la información requerida, tal como retardo punto a punto, throughput[1], perdida de paquetes, probabilidad de bloqueo, entre otras. Para poder realizar estas medidas en cada entorno de simulación inicialmente se aumenta en la fuente el número de paquetes que se van a generar, lo que permitirá observar en las gráficas de mejor manera como se comporta la red con un mayor volumen de tráfico. También se puede realizar cambios en el tamaño de

[1] Tasa de transmisión de los paquetes de información entre los nodos de la red.

cada paquete, con lo que se busca congestionar la red, para que se pueda ver en su totalidad el funcionamiento de los módulos de control de congestión desarrollados. Teniendo ya aplicadas las configuraciones generales de la red, se procederá a simular la red y obtener las gráficas correspondientes para cada sistema.

CAPÍTULO 4

EXPERIMENTACIÓN Y ANÁLISIS DE RESULTADOS

4.1 introducción

En este capítulo se presenta el análisis de los resultados obtenidos en las simulaciones realizadas con los distintos métodos de control de contención para una red OBS distribuida, cuyo sistema fue implementado en el IDE OMNet++ donde el estudio se hace en base a el tráfico generado en la red a medida que transcurre el tiempo, el retardo punto a punto y la probabilidad de bloqueo[2-3]. También se analiza el comportamiento de la red al variar ciertos parámetros específicos establecidos para cada sistema.

Estos resultados permiten establecer el impacto en la red al usar métodos de control de contención, por esto se plantean los siguientes sistemas en los que se tienen distintos mecanismos de control de contención y en los que a su vez se realiza la modificación de distintas variables, para los casos de estudio que se muestran a continuación:

- Primer Caso de Estudio: Red OBS distribuida sin ningún método de control de contención.
- Segundo Caso de Estudio: Red OBS distribuida con control de contención mediante el uso FDL.
- Tercer Caso de Estudio: Red OBS distribuida con control de contención por Cambio de Longitud de Onda
- Cuarto Caso de Estudio: Red OBS distribuida con control de contención por Deflexión de Ruta

4.2 Análisis de Resultados

Para el estudio y comparación de los sistemas se cuenta con parámetros fijos que son idénticos para cada simulación, así como también parámetros únicos que corresponden a cada uno de los módulos de contención, que son representados en sus respectivos casos de estudio.

[2] $Probabilidad\ de\ bloqueo = 1 - \frac{Paquetes\ Recibidos}{Paquetes\ enviados}$

[3] Se utilizaron estos parámetros, ya que en la literatura científica al respecto [28] [29], se usan para evaluar el desempeño de una red OBS, puesto que proporcionan la información necesaria para establecer cómo se comporta el sistema respecto al tráfico establecido.

En la tabla 4.1 se pueden observar los parámetros generales que comparten todos los sistemas y que dan una perspectiva general de las condiciones en las que se van a realizar dichas simulaciones.

Tabla 4.1. Parámetros Generales de Simulación.

Parámetro	Valor
Tiempo de Simulación	1.5 Segundos
Tamaño del Paquete	Entero uniforme entre 128-1024 bytes
Número de Nodos	26
Tiempo de Generación entre Paquetes	Décima parte de un valor exponencial aleatorio de 0.2ms
Longitudes de Onda	6
Velocidad de Transmisión del canal	1Gbps y 2.5Gbps
Retardo del Canal	0.1ms

Los parámetros únicos para cada sistema se mostrarán individualmente en el estudio que se presenta a continuación, con el fin de tener una mejor perspectiva de cómo afectan los módulos de contención.

Para obtener un mejor estudio sobre el impacto de los mecanismos de contención en la red, se obtuvieron los resultados en el periodo de tiempo correspondiente entre 0.5 y 1.5 segundos donde se esperaba que los módulos de la red presenten un estado estacionario, es decir, que el sistema ya se encuentre estable, debido a que al inicio de la simulación todos los módulos de la red se encontrarían en sus condiciones óptimas, es decir, sin ninguna carga por lo tanto no se podría observar en gran medida el funcionamiento de los mecanismos de contención.

Los resultados obtenidos se graficaron con la herramienta OriginLab, ya que proporciona características de diseño avanzado, por medio de las cuales se pueden realizar gráficos profesionales, como la posibilidad de obtener la comparación de curvas dadas para distintos datos establecidos, figuras en tercera dimensión y de dispersión, cuya presentación es más adecuada para el análisis y presentación de la información.

4.2.1 Red OBS distribuida sin ningún tipo de control de contención

Esta simulación se realizó como base para realizar una comparación posterior, con el fin de observar las diferencias entre tener un sistema sin ningún tipo de contención y un sistema con alguno de los métodos de control de contención. Por lo tanto, las configuraciones realizadas en esta red son las generales que

comparten todas las demás redes simuladas para los distintos métodos de control de contención en este trabajo.

El tráfico que se mueve por la red presenta niveles que presentan variaciones importantes a medida que transcurre el tiempo, pero se puede notar cierta tendencia a mantenerse entre 10 y 30 Erlangs como se puede ver en la figura 4.1. También se puede ver que en ciertos periodos de tiempo no se muestran algunos valores de tráfico y esto se debe a que todo el tráfico generado en ese tiempo fue bloqueado por la red.

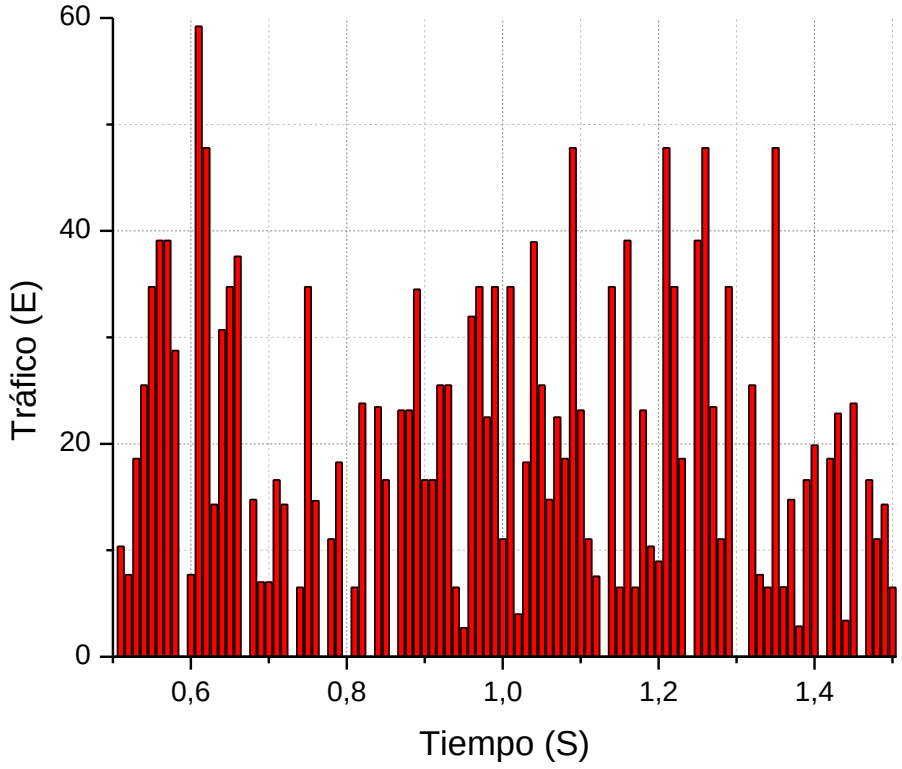

Figura 4.1. Tráfico generado en la red respecto al tiempo.

Puesto que este sistema no cuenta con ningún método de control de contención, se tiene que a medida que aumenta el tráfico en la red, la probabilidad de bloqueo aumenta de manera significativa, alcanzando un nivel máximo de bloqueo como se muestra en la figura 4.2 (a), lo que implica una gran pérdida de información y hace que este sistema no sea práctico. Por otra parte en la medida del retardo punto a punto inicialmente se presenta un valor constante de 0,2ms para valores de tráfico entre los 6 a 15 Erlangs lo que estaría bien considerando que para estos niveles tan bajos de tráfico la saturación de la red es mínima, esto se puede ver en la figura 4.2 (b). Ahora a medida que el trafico aumenta la red empieza a saturarse y el retardo punto a punto aumenta en gran magnitud, entonces el tráfico que soporta la red para niveles medios y altos no solo presenta una gran probabilidad de ser descartado, sino también sufre de retardos extremadamente altos para llegar a su destino.

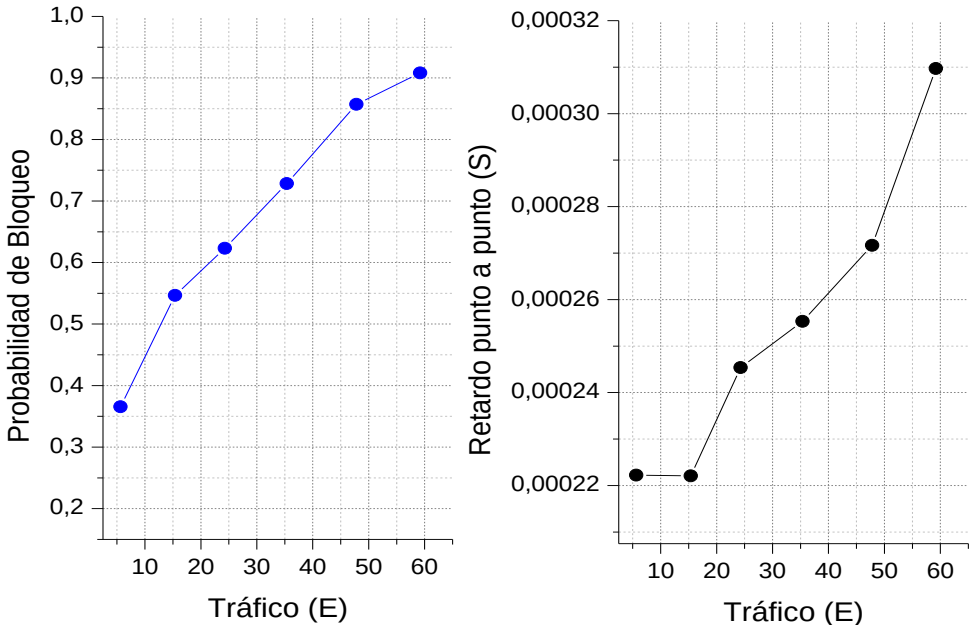

Figura 4.2. Comportamiento del tráfico en el sistema a) Probabilidad de Bloqueo b) Retardo punto a punto.

Teniendo una probabilidad de bloqueo tan alta para los niveles de tráfico más comunes presentados en la red, es de esperarse que finalmente los paquetes que logren llegar a su destino sean muy pocos, lo cual se muestra en la figura 4.3, donde solamente son recibidos en su destino el 18% del total de paquetes enviados por todos los nodos de la red, es decir, que este sistema presenta un bajo desempeño ya que en una red tan grande y con tantos nodos, las colisiones entre ráfagas pueden ocurrir muy a menudo generando esta gran pérdida de información.

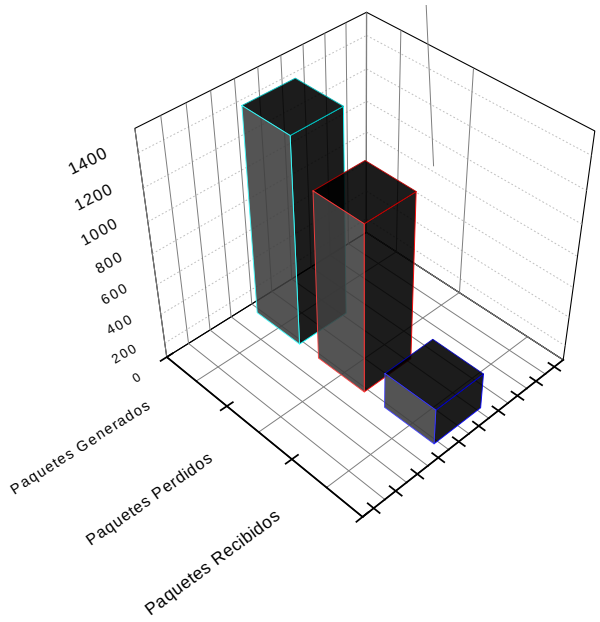

Figura 4.3. Paquetes Generados, perdidos y recibidos.

4.2.2 Red OBS distribuida con control de contención mediante el uso FDL

En esta simulación se implementó el método de contención para una red OBS conocido como línea de retardo de fibra o FDL por sus siglas en inglés, para esto se usó el módulo correspondiente con las configuraciones mostradas en la tabla 4.2.

El módulo de contención tiene la posibilidad de configurarse para usar desde 1 hasta 4 FDL, donde los retardos asignados a cada FDL varían dependiendo del tamaño L de los paquetes de información.

Tabla 4.2. Configuración módulo de contención FDL.

	Ecuación para asignación de retardo	Valores posibles de retardo
Retardo FDL 1	Retardo= $\frac{L}{10^7}$ segundos.	12.8us - 102.4us
Retardo FDL 2	Retardo = $\frac{2L}{10^7}$ segundos.	25.6us – 204.8us
Retardo FDL 3	Retardo = $\frac{3L}{10^7}$ segundos.	38.4us - 307.2us
Retardo FDL 4	Retardo = $\frac{4L}{10^7}$ segundos.	51.2us - 409.6us

En la figura 4.4 se muestra el tráfico generado en la red, durante el periodo de simulación establecido para la obtención de los resultados, se observa un tráfico

con un valor medio constante que presenta algunos picos, los cuales se deben a que en ciertos instantes de tiempo la red se satura.

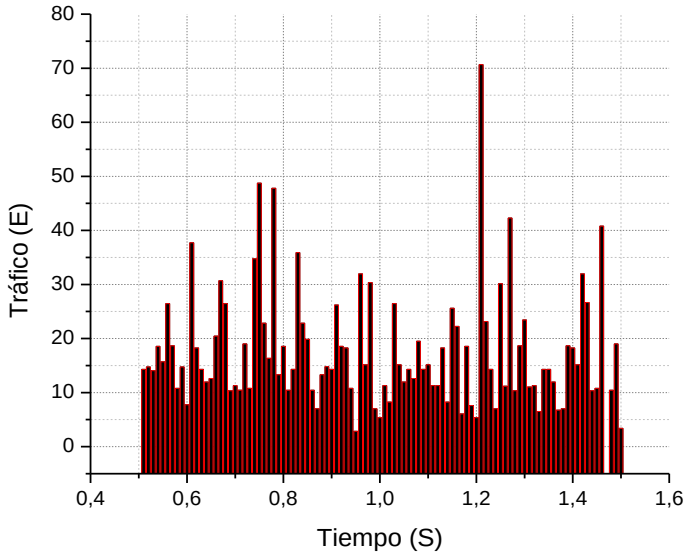

Figura 4.4. Tráfico generado en la red respecto al tiempo.

Debido a que en el sistema se generan aleatoriamente los paquetes, el tráfico en la misma no presenta un aumento lineal, lo que implica que las variaciones de este pueden ser considerables a medida que los módulos de contención trabajan para evitar que la red se congestione. Sin embargo, es de esperarse que el sistema de control de contención tenga que bloquear algunos paquetes con el fin de evitar la pérdida de muchos más y este fenómeno se puede observar en la figura 4.5 (b), donde la probabilidad de bloqueo de la red aumenta gradualmente a medida que el tráfico lo hace.

Ahora aunque la probabilidad de bloqueo aumenta respecto al tráfico generado en la red en determinado tiempo, esto no implica que el retardo punto a punto también lo haga como lo muestra la figura 4.5 (a), ya que el método de control de contención por FDL le puede dar cierto retardo a un paquete dependiendo de la disponibilidad de recursos o descartar ese paquete en caso de que no haya recursos disponibles, es por esto que se dan estas fluctuaciones en el retardo punto a punto, ya que aunque el tráfico en la red sea bajo estos paquetes puedan estar solicitando los mismos recursos en los nodos de la red, lo que implicaría realizar control de contención para dichos paquetes y por ende el retardo asignado a estos paquetes podría ser mayor que el retardo que impactaría a un mayor tráfico de información.

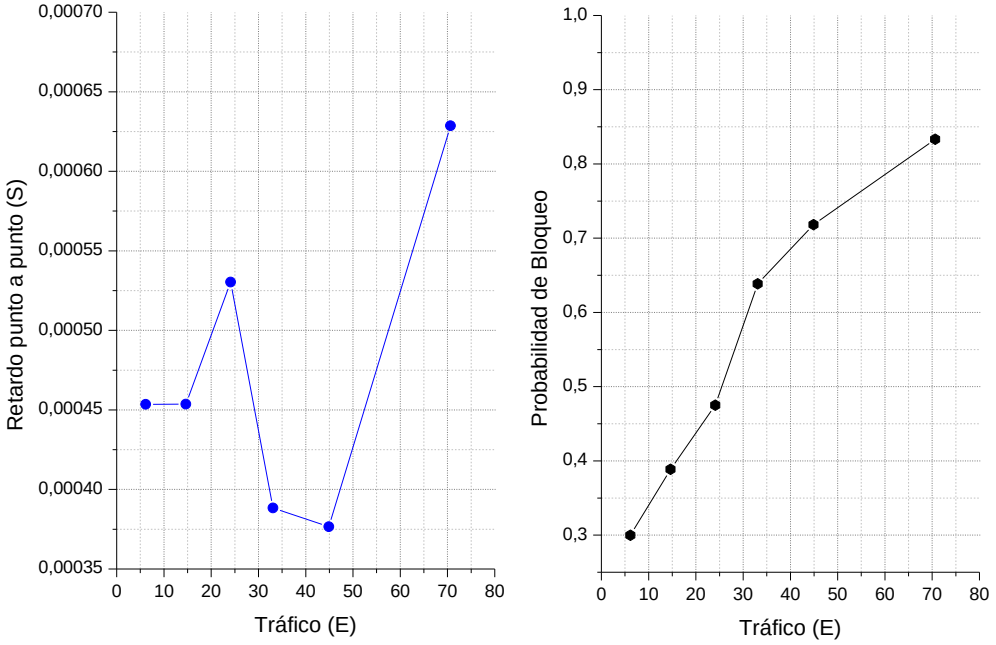

Figura 4.5. Comportamiento del tráfico en el sistema con FDL a) Retardo punto a punto b) Probabilidad de Bloqueo.

Como se puede observar en la figura 4.6 la tasa de paquetes generados, perdidos y recibidos en la red, para velocidades de transmisión de 1 Gbps y 2.5 Gbps es igual, esto se debe a que no afecta directamente en el método de control de contención, sino que su impacto se ve reflejado en el retardo punto a punto de cada paquete, por lo que al aumentar la velocidad es de esperarse que los paquetes lleguen con mayor rapidez a su destino.

Además se muestra como a medida que aumenta el número de FDL, se obtiene una gran mejora en cuanto a los paquetes recibidos que pasa de 431 con 1 FDL a 761 con 4 FDL. Esto se debe a que el módulo de control de contención puede descartar menos paquetes, ya que tiene más canales por donde enviar la información, aunque se les asigne un retardo mayor.

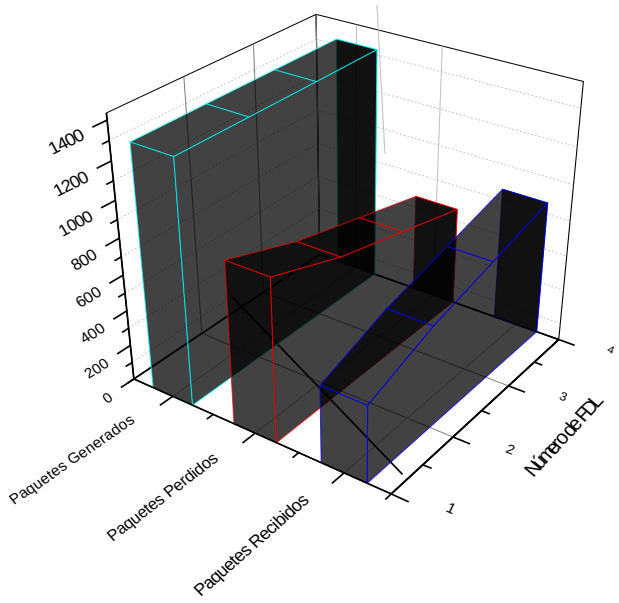

Figura 4.6. Paquetes Generados, perdidos y recibidos para distintas FDL.

Sin embargo, a medida que se aumenta el número de FDL, el retardo punto a punto aumenta en gran medida de forma lineal como se muestra en la figura 4.7, también permite ver la diferencia de retardos entre las velocidades de transmisión, obteniendo una pequeña mejora con una tasa de 2.5Gbps.

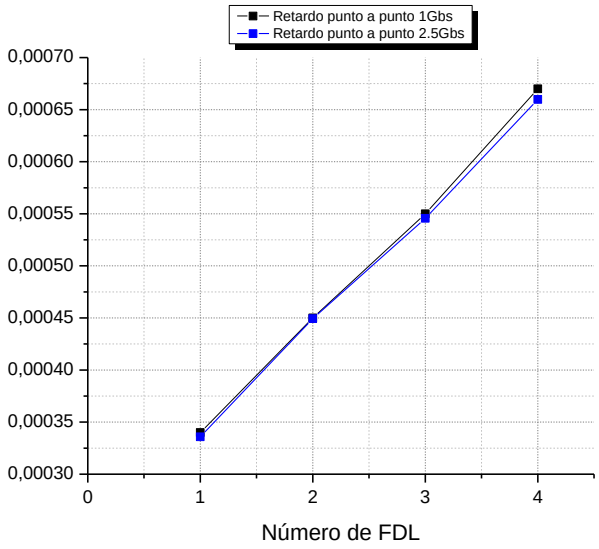

Figura 4.7. Variación del retardo punto a punto con tasas de transmisión de 1Gbps y 2.5Gbps.

4.2.3 Red OBS distribuida con control de contención por Cambio de Longitud de Onda

En esta simulación se implementó el método de contención para una red OBS conocido como cambio de longitud de onda, para lo cual se pueden usar desde 2

hasta 16 longitudes de onda en el sistema, la información recolectada se ha tomado con la asignación de 6 longitudes de onda y una velocidad de transmisión en el enlace de 1Gbps, ya que permite realizar un mejor estudio del funcionamiento del método de control de contención.

Para el sistema con cambio de longitud de onda se tiene un volumen de tráfico bastante variable como se muestra en la figura 4.8, donde se puede ver una leve tendencia entre 10 a 20 Erlangs, aunque también se presentan picos que se distribuyen durante el periodo estudiado y que se dan cada 0.3s.

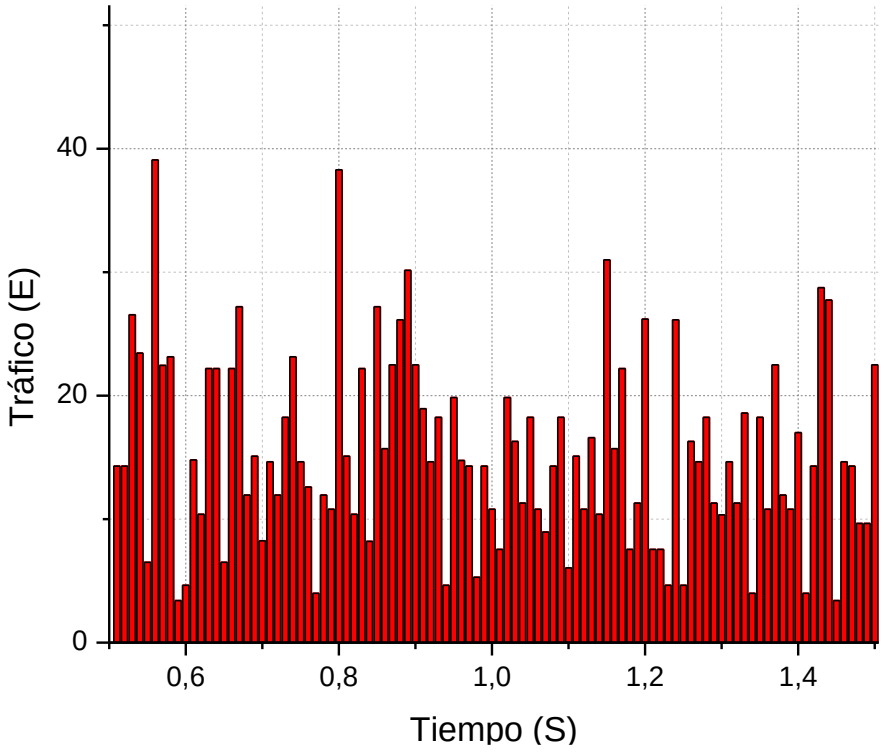

Figura 4.8. Tráfico generado en la red respecto al tiempo.

Como lo muestra la figura 4.9 (a) la probabilidad de bloqueo aumenta respecto al aumento del tráfico que se genera en la red, aunque tiene una leve tendencia a decaer que se da por el manejo de los paquetes que hacen los módulos de contención en cada nodo, donde a medida que va aumentando el tráfico los modulos responden realizando el control de contención, pero usando de manera equitativa todas las longitudes de onda disponibles para su uso. Ahora aunque este método de control de contención no aumenta el retardo punto a punto, se puede ver en la figura 4.9 (b) que este si aumenta a medida que aumenta el tráfico, ya que al aumentar el flujo de información, esto trae consigo un aumento en el tiempo de procesamiento de la misma. Aunque en cierto punto se presenta una ligera disminución del retardo, es producto de una liberación en los recursos del sistema que se da cuando la red se satura.

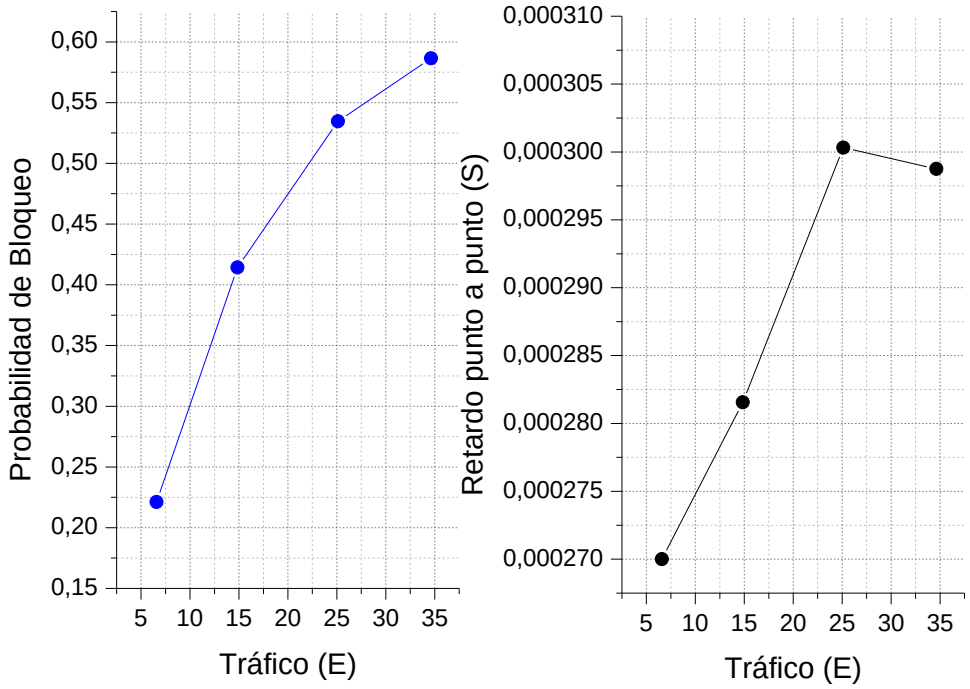

Figura 4.9. Comportamiento del tráfico en el sistema con FDL a) Probabilidad de Bloqueo b) Retardo punto a punto.

En este sistema se tuvieron cambios en el número de paquetes generados, recibidos y eliminados al simular con velocidades de transmisión de 1Gbps y de 2.5Gbps como se puede apreciar en la figura 4.10, sin embargo no fueron valores significativos que afectaran en gran medida el desempeño de la red. Lo que sí se puede apreciar en gran medida, es como mejora de una manera sustancial, el número de paquetes recibidos a medida que se aumenta el número de longitudes de onda disponibles en el sistema, pasando de tener un mínimo de paquetes recibidos entre 419 a 436 con 2 longitudes de onda, hasta un máximo entre 1128 a 1069 paquetes recibidos con 16 longitudes de onda. Por lo que se puede establecer que para un mejor funcionamiento tanto de la red, como del método de control de contención por cambio de longitud de onda, es conveniente utilizar un mayor número de longitudes de onda para el manejo de la información.

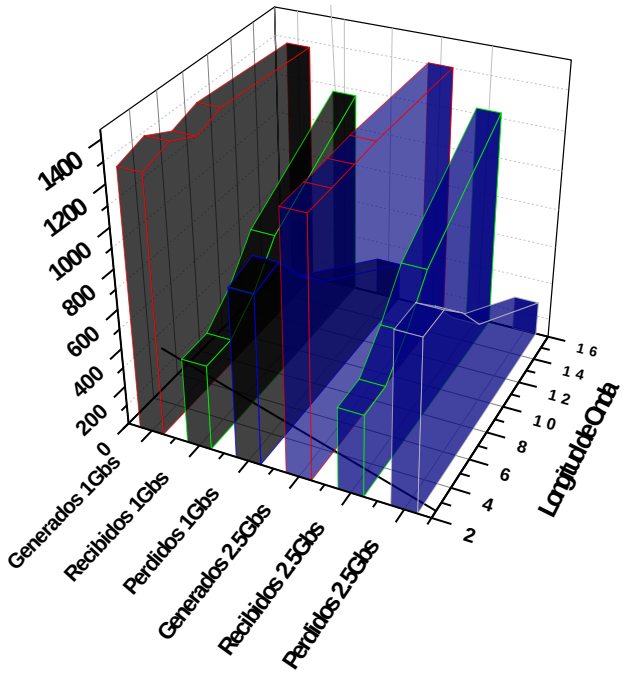

Figura 4.10. Paquetes Generados, perdidos y recibidos para distintas longitudes de onda con velocidades de 1 y 2.5 Gbps.

Sin embargo al aumentar el número de longitudes de onda el retardo punto a punto tienda a aumentar como lo muestra la figura 4.11, aunque es posible hacer que este retardo disminuya, al aumentar la velocidad de transmisión del canal ya que se evidencia una leve mejoría al usar una tasa de transmisión de 2.5Gbps.

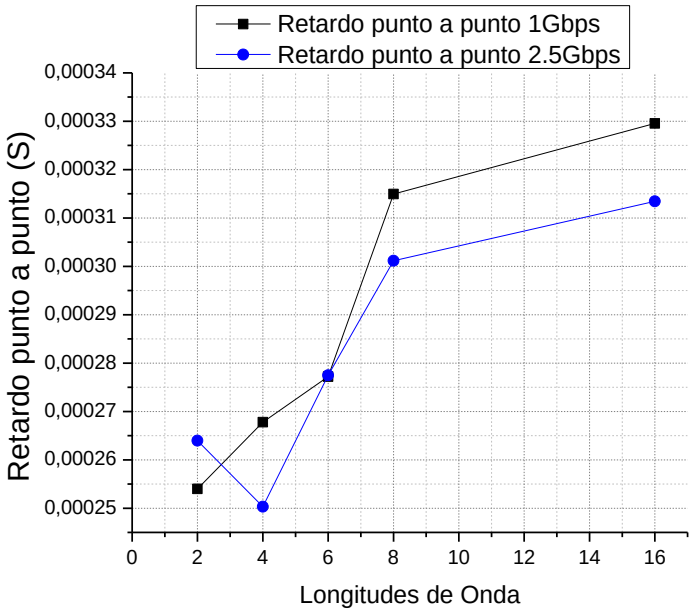

Figura 4.11. Variación del retardo punto a punto con tasas de transmisión de 1Gbps y 2.5Gbps.

En el retardo punto a punto obtenido al usar 4 longitudes de onda, se presenta una diferencia de casi 0.2ms que se debe a una ligera variación en la cantidad de tráfico generado en la red para las distintas velocidades que fue un poco mayor para el sistema con velocidad de transmisión en el enlace de 1Gbps.

4.2.4 Red OBS distribuida con control de contención por Deflexión de Enrutamiento

Se realiza la simulación de la red OBS distribuida aplicando el mecanismo de control de contención por deflexión de ruta cuyo funcionamiento ha sido descrito en los apartados anteriores. Se procede hacer el estudio de este mecanismo variando el tamaño de las ráfagas y teniendo el número de longitudes de onda y la velocidad de transmisión en el enlace constante, en la tabla 4.3 se especifican los parámetros de configuración de la red. Para el propósito de comparación de este mecanismo con los anteriores, se toma una longitud de ráfaga fijo y un periodo de simulación de 1.5 segundos.

Tabla 4.3. Configuración módulo de contención por deflexión de ruta.

Velocidad de transmisión de la fibra	Retardo	Tamaño promedio de la ráfaga	Tiempo promedio de procesamiento en cada nodo	offset	No. Longitudes de onda
1Gbps	0.1ms	37.5Kbytes	0.1 ms	2,7 ms	6

De acuerdo al algoritmo empleado para la deflexión de las ráfagas tendríamos paquetes de control que no pueden encontrar recursos para su respectiva ráfaga los cuales son dirigidos al nodo vecino provocando mayor congestión en el enlace y sumando a su retardo el tiempo de procesamiento de un nodo más. Esto no contribuye al tráfico de datos dado que los paquetes se mueven en el plano de control, pero sí contribuya a la pérdida de ráfagas cuando el tiempo de offset se agote y el paquete con su ráfaga tengan que ser descartados. En la gráfica siguiente se observa el flujo de tráfico en el tiempo, en ella podemos visualizar cómo la red se encuentra invadida de paquetes. Cuando el mecanismos de control de contención por deflexión de ruta es invocado, el paquete es obligado a tomar un nuevo camino, tomando en su ruta mayor número de nodos congestionando más ese enlace. Si el paquete deflectado no encuentra recursos en el nuevo nodo, éste se devuelve al nodo de deflexión inicial debido a que es por éste donde encuentra la ruta mas corta. En ciertos momentos pueden haber muchos paquetes sin recursos que causan más trafico en los enlaces, en la figura 4.12 se puede apreciar los momentos donde el tráfico se dispara.

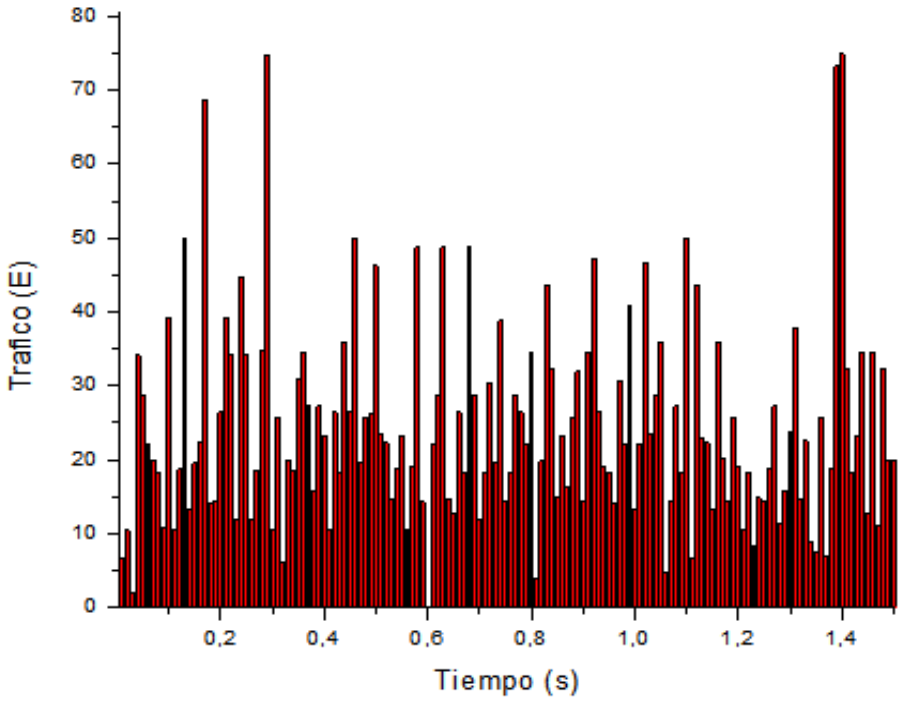

Figura 4.12. Tráfico generado en la red respecto al tiempo.

El retardo aumenta considerablemente con el aumento del tráfico como se puede ver en la figura 4.13 (b), dado que un paquete es eliminado únicamente cuando su tiempo de offset se ha agotado y puede permanecer en la red hasta que encuentre la ruta que lo lleve a su destino.

Cuando el paquete de control llega a un nodo, éste lleva la información para que el nodo reserve los recursos a su ráfaga, si el canal está congestionado lo más probable es que no encuentre recursos para su ráfaga de lo cual se dirige a otro nodo contando con igual probabilidad de poder seguir su ruta, por lo tanto cuando aumenta el tráfico en el canal menos posibilidad tiene un paquete de continuar su camino. Los datos tomados de tráfico con respecto a la probabilidad de bloqueo nos muestran que a medida que éste aumenta la probabilidad de pérdida de paquetes y por supuesto, de ráfagas como se muestra en la figura 4.13 (a).

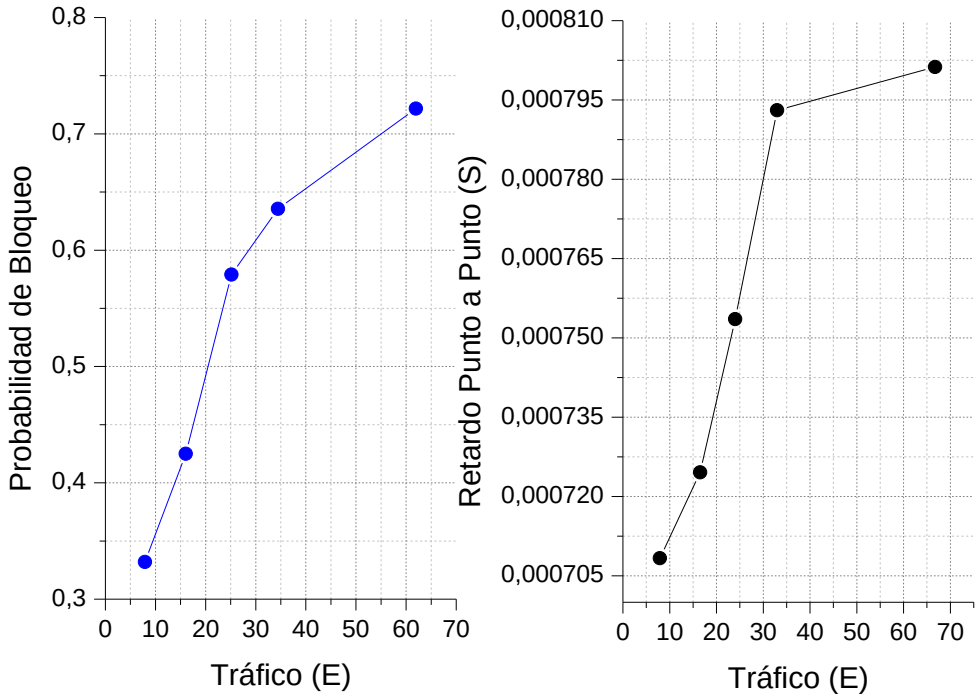

Figura 4.13. Comportamiento del tráfico en el sistema con Deflexión a) Probabilidad de Bloqueo b) Retardo punto a punto.

Unos de los parámetros más importantes en esta red es el tamaño de la ráfaga, dado que ráfagas muy grandes entrarían en contienda mucho más rápido que una de menor tamaño y al contrario de esto, ráfagas pequeñas estarían muy separadas espacialmente las unas con la otras evitando los choques que conducen a su eliminación, pero si ésta es muy pequeña la red OBS no sería eficiente, se trata entonces de hallar un equilibrio, entre pérdida de paquetes y eficiencia de la red, Para esta simulación se tomaron 7 valores diferentes de longitud de ráfagas, en la figura 4.14 se observan los rangos establecidos para estas longitudes, con los datos tomados de la simulación se establecen las gráficas descritas en las cuales podemos apreciar lo siguiente:

Para longitudes pequeñas de ráfagas el número de paquetes recibidos supera al número de paquetes perdidos y al aumentar la longitud de la ráfaga, el número de paquetes recibidos decrece provocando, por supuesto, el aumento de los paquetes perdidos.

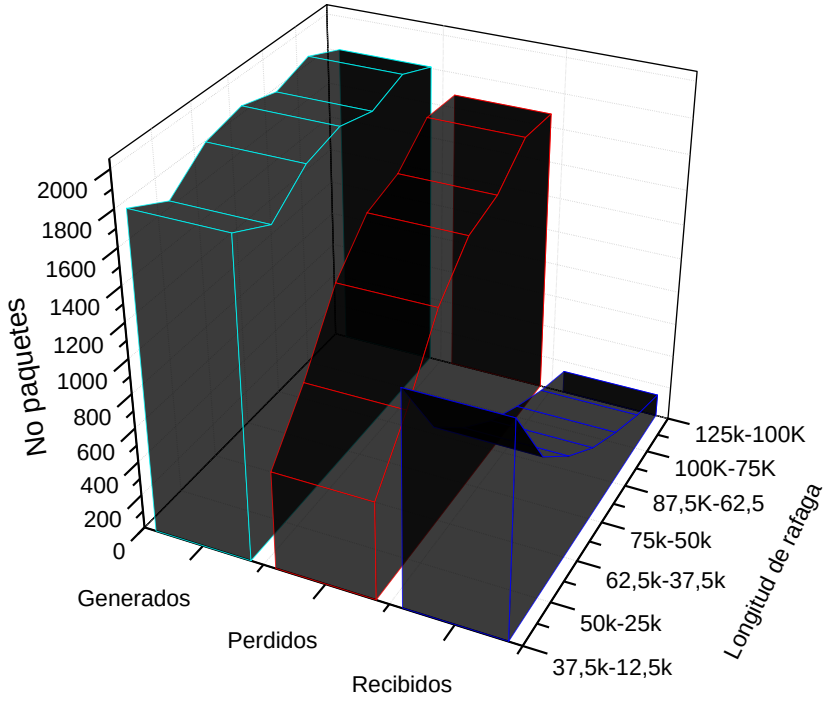

Figura 4.14. Paquetes Generados, perdidos y recibidos para distintas longitudes de ráfaga.

4.3 Comparación de los métodos de control de contención

En los apartados anteriores se hizo un estudio detallado de los parámetros más relevantes para analizar el desempeño de cada sistema, con lo que se pudo identificar los cambios que se daban en la red al implementar cada uno de los métodos de control de contención para distintas configuraciones de velocidad, retardo, longitudes de onda y tamaño de la ráfaga.

En esta sección se realiza la comparación de los distintos métodos de control de contención, con el fin de tener una idea más específica sobre las diferencias que genera su implementación en la red.

Debido a que los distintos métodos de control de contención realizan un manejo independiente y diferenciado de la información que por ellos transita, el retardo que cada uno agrega al tráfico de la red presenta grandes cambios como se puede ver en la figura 4.15, donde se puede observar que el mecanismo de control de contención por deflexión de ruta tiene un retardo mucho mayor que los otros métodos de control de contención, y esto se debe principalmente a que al realizar la deflexión de ruta, las ráfagas se pueden quedar transitando en la red durante un tiempo bastante significativo lo que se ve reflejado en el retardo punto a punto. Por otra parte el método de control de contención por cambio de longitud de onda, presenta el retardo punto a punto más bajo de los tres mecanismos

puesto que agrega retardos mínimos, que se dan más que todo en el procesamiento de la información, por lo que el retardo punto a punto con mayor valor es 0.3ms lo que equivale a un poco menos de la mitad del valor máximo en el retardo punto a punto mediante el uso de FDL.

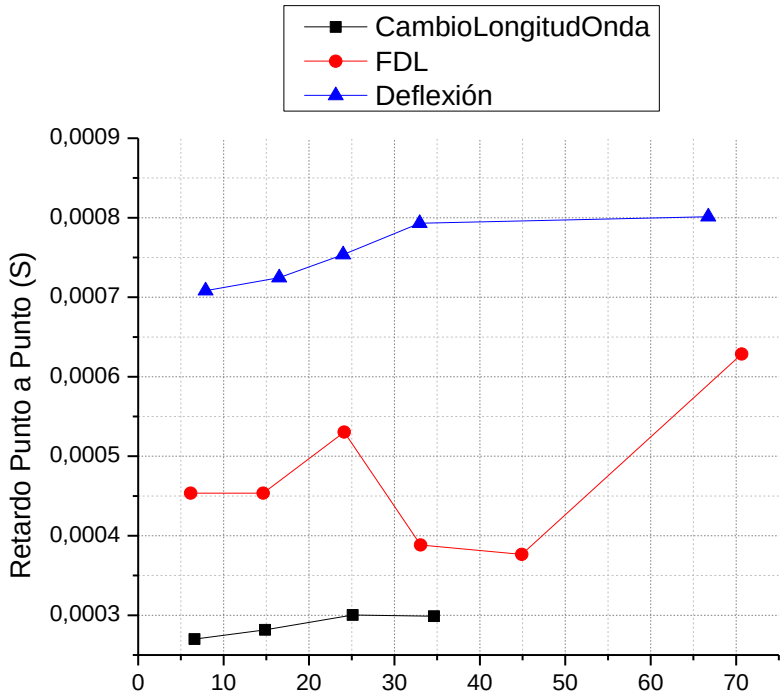

Figura 4.15. Comparación del retardo punto a punto generado para cada método de control de contención.

Ahora aunque el retardo punto a punto es un factor muy importante a tener en cuenta el desempeño de los distintos métodos de control de contención, no es el único que permite verificar cuál de los 3 mecanismos tuvo un mejor rendimiento. Es así como al comparar la probabilidad de bloqueo, se puede apreciar que los métodos de control de contención por deflexión de ruta y cambio de longitud de onda, tuvieron una menor probabilidad de descartar el tráfico que recorría la red cuyos valores máximos fueron del 73% y 59% respectivamente para el tráfico más alto presente en cada sistema como se muestra en la figura 4.16.

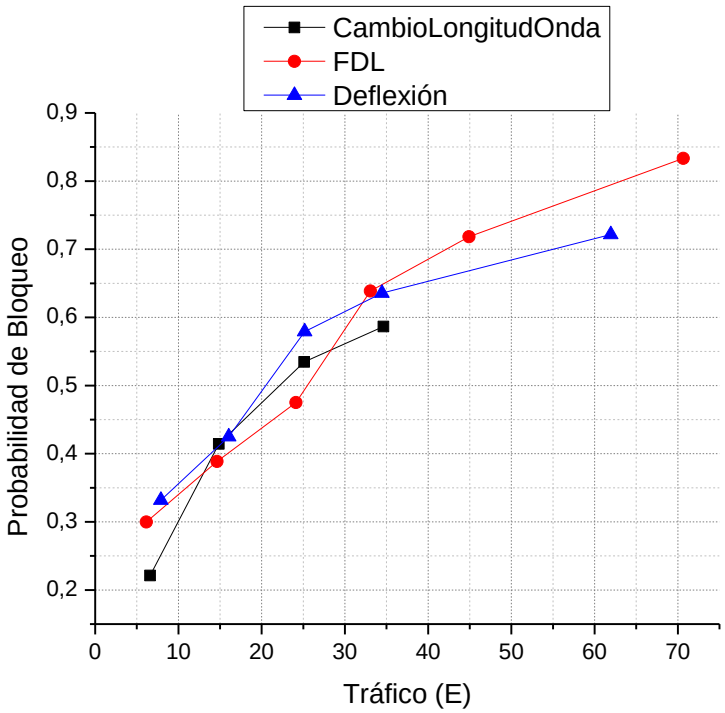

Figura 4.16. Comparación de la probabilidad de bloqueo para cada método de control de contención.

Puesto que la red con método de control de contención mediante el uso de FDL maneja niveles de tráfico más altos, presenta una probabilidad de bloqueo de casi el 85% lo que se podría tomar como un desempeño muy inferior respecto a los otros sistemas, sin embargo es válido afirmar que si se compara la probabilidad de bloqueo para niveles de trafico iguales, el desempeño de este sistema fue incluso mejor que el de los otros, y esto se puede apreciar en los valores de tráfico entre 15 a 25 Erlang. Por lo que finalmente se puede establecer que la probabilidad de bloqueo para todos los sistemas es muy similar, presentando una tasa de bloqueo más pequeña para el método de control de contención por cambio de longitud de onda.

En la tabla 4.4 se resume los resultados obtenidos de probabilidad de bloqueo, retardo extremo a extremo y paquetes eliminados, para los distintos métodos de control de contención.

El método de deflexión de ruta es el que genero retardos mayores debido a que agrega retardo de procesamiento en los nodos y retardo del canal, lo que termina añadiendo un gran nivel de retardo a la ráfaga. La principal dificultad del uso de este mecanismo es que un paquete puede quedarse transitando por los nodos durante un tiempo indefinido, por esta razón es necesario encontrar un tiempo de offset óptimo a cada ráfaga con el fin de evitar saturar la red.

Con los resultados obtenidos de retardo punto a punto y probabilidad de bloqueo, se puede decir que el método de control de contención por cambio de longitud de onda presenta un mejor rendimiento en el manejo de la contención en una red OBS distribuida, presentando niveles bajos de estos parámetros esenciales para el funcionamiento efectivo de la red.

Tabla 4.4. Comparación de los resultados obtenidos sobre los parámetros de desempeño establecidos.

Método de Control de Contención	Probabilidad de bloqueo máxima	Retardo punto a punto máximo	Paquetes eliminados
Cambio de Longitud de Onda	58%	0.3ms	707
Línea de Fibra de Retardo	83%	0.67ms	758
Deflexión de Ruta	73%	0.8ms	875

CAPÍTULO 5

CONCLUSIONES, RECOMENDACIONES Y TRABAJOS FUTUROS

5.1 Conclusiones

Al implementar una red óptica, se hace indispensable el uso de métodos de control de contención, no solo para evitar la pérdida de información, sino también para mejorar el desempeño en general de la misma. Ya que realizar el control de contención en los nodos de red proporciona estabilidad y eficiencia en el uso de tecnologías de conmutación como OBS, y asegura que la gran mayoría del tráfico que circula por la red pueda llegar a su destino final.

En una red que cuente con una gran cantidad de nodos, la necesidad de realizar cambios de ruta para las ráfagas es muy necesario, por lo que el uso de metaheurísticas para establecer cuál de todas las posibilidades de enrutamiento será el más adecuado para el envío de la información resulta conveniente en la resolución de este problema, ya que sin la aplicación de estos algoritmos podrían generarse conflictos entre los nodos de red, lo que finalmente conduciría a un colapso de la red.

El modelo GEANT 2, utilizado para el desarrollo de la topología de red, permitió cumplir los objetivos de simulación propuestos, ya que gracias a este se pudo observar los cambios que se presentaban en el desempeño de la red, al utilizar en los nodos cada uno de los mecanismos de control de contención. Se seleccionó debido a que este modelo cuenta con un gran número de nodos, por lo que el tráfico en la red fue mucho mayor que el presentado en otros modelos, además que cada nodo comparte muchos más enlaces entre sí, lo que proporciona un entorno ideal para la aplicación de control de contención.

Según el principio del funcionamiento del control de contención por deflexión de ruta, la principal dificultad del uso de este mecanismo, es que se pueden generar bucles en la red, ya que una ráfaga puede quedarse transitando por la misma, durante un tiempo indefinido. Por esta razón es necesario encontrar y asignar un tiempo de offset óptimo a cada ráfaga con el fin de evitar el saturamiento de la red, al configurarlo de tal manera que cuando este tiempo llegue a un valor de cero, la ráfaga sea descartada por el sistema.

5.2 Recomendaciones

Para la generación de paquetes, tener en cuenta la función que se va a utilizar y los valores que arrojará, debido a que en la herramienta de simulación IDE OMNet++ el tiempo entre generación de paquetes va ligado inherentemente al funcionamiento del sistema, por lo que si no se tiene en cuenta este parámetro es posible que la red no funcione como se espera o incluso los datos a analizar puedan ser erróneos.

Se sugiere utilizar señales en el sistema que ayudan a ofrecer información más exacta y detallada que los datos arrojados en la consola del simulador, puesto que generaliza los tiempos en ingreso o salida de los distintos nodos de la red, por lo que al tomar estos datos y compararlos con los obtenidos por las señales, se nota una gran diferencia en los valores que puede ser critica al momento de realizar un análisis a fondo.

Respecto al simulador tener muy en cuenta las conexiones que se van a realizar en la red, verificar que el tamaño de los enlaces interconectados sean iguales o podría generar conflictos al momento de realizar la simulación, igualmente verificar si el enlace va a ser simple (entrada o salida) o por otro lado se va a comportar como un puerto (entrada y salida) ya que una mala configuración de esto puede estropear la red.

Al desarrollar el algoritmo de trabajo en los nodos, identificar los parámetros que se van a usar para gestionar el flujo de datos, ya que si alguna variable no es tomada correctamente, aunque parezca que el sistema funciona de una manera adecuada, al analizar mejor se puede observar una gran pérdida de información en la red, que evitara una correcta interpretación de los resultados.

Para obtener diversos resultados, se pueden usar diversas señales en la programación de los módulos, que sean configuradas para proporcionan cierta información importante para el desarrollo del trabajo. Es importante establecer cómo se van a mostrar estas señales, ya que el simulador puede guardar esta en información en dos formas diferente, como escalares o como vectores, dependiendo de la elección es importante tener en cuenta el formato en el que se muestra la información en estos archivos, para no cometer errores de interpretación al leer los datos.

En cuanto a los resultados obtenidos, se recomienda desarrollar alguna función o algoritmo que permita establecer el tráfico máximo que se generara en la red, puesto que dependiendo del sistema a simular, el tráfico varia significativamente lo que puede afectar un poco el análisis de los resultados al comparar distintos entornos de simulación.

El IDE OMNet++ es una herramienta robusta que consume gran cantidad d recursos, por lo que se recomienda el uso de un equipo con procesador i7 y memoria RAM de 8Gb o más, para evitar problemas en la ejecución del programa desarrollado. También es necesario instalar la herramienta de manera adecuada, porque si se realiza un mal procedimiento el simulador será inestable y demorara mucho en compilar.

5.3 Trabajos futuros

Desarrollar e implementar un método de control de contención para distintas topologías de red, que se adapte a los cambios de tráfico y enrutamiento, proporcionando una mejora en el desempeño de cada sistema.

Añadir como mejora para el sistema, la asignación de prioridad a las ráfagas generadas estableciendo distintas clases para realizar un mejor análisis en el comportamiento del sistema.

Diseñar algoritmos que empleen distintos métodos de contención para trabajar en conjunto, tales como FDL con cambio de longitud de onda, FDL con deflexión de ruta y cambio de longitud de onda con deflexión de ruta.

Referencias

[1] B. O. J. C. K. C. U. C. S. F. Gustavo Puerto Leguizamón, «Evolución de las redes de datos: Hacia una plataforma de comunicaciones completamente óptica,» *Rev. Fac. Ing. Univ. Antioquia,* nº N.° 45, pp. 148-156, 2008.

[2] M. Á. Tena Martín, «Evaluación de arquitecturas de red híbridas OBS/OCS,» Barcelona, 2009.

[3] «HUAWEI,» [En línea]. Available: http://www.huawei.com/. [Último acceso: 20 8 2015].

[4] L. C. Hinojosa Gomez, «Tópicos selectos de fibra óptica,» Hidalgo, 2007.

[5] A. O. Manzanera, «Diseño y evaluación de prestaciones de una red óptica OBS multinodo mediante herramienta de simulación,» Cartagena, 2009.

[6] A. P. V. J. O. Saleck Marquez Lozano, «MUTIPLEXACIÓN POR DIVISIÓNN DE LONGITUD DE ONDA (WDM),» Santa Cruz de la Sierra, 2012.

[7] S. Thibaudeau, «Network Layers,» 13 Septiembre 2013. [En línea]. Available: http://simonthibaudeau.org/NetworkLayers/?p=107. [Último acceso: 15 Mayo 2015].

[8] G. Chavarría, «Slideshare,» 21 Marzo 2012. [En línea]. Available: http://es.slideshare.net/gersonchavarriavera/dwdm-12108779. [Último acceso: 05 Mayo 2015].

[9] «PacketLight Networks,» 30 Marzo 2015. [En línea]. Available: http://www.packetlight.com/technology/dwdm-over-cwdm/. [Último acceso: 14 Abril 2015].

[10] C. G. Morales, «Wikispaces,» 31 Octubre 2012. [En línea]. Available: http://sx-de-tx.wikispaces.com/DWDM+y+CWDM. [Último acceso: 26 Abril 2015].

[11] «Cisco,» Cisco Systems, Abril 2009. [En línea]. Available: http://www.cisco.com/c/en/us/products/collateral/interfaces-modules/transceiver-modules/product_data_sheet0900aecd806a1c36.pdf. [Último acceso: 01 Mayo 2015].

[12] G. J. Andrade, «Estudio de la tecnología de conmutación óptica por ráfagas

OBS y migración de redes ópticas pasivas a esta tecnología,» Sangolquí, 2011.

[13] I. Graña Cereijo, «Simulación y comparativa de mecanismos de conmutación en redes ópticas,» Vigo, 2005.

[14] Á. Ferreiro, «Escuela Politécnica Superior,» 19 Junio 2008. [En línea]. Available: http://arantxa.ii.uam.es/~ferreiro/sistel2008/anexos/WDM.pdf. [Último acceso: 25 Abril 2015].

[15] D. A. Flores Cárdenas, «Estudio y simulación de la tecnología de conmutación óptica por ráfagas,» Quito, 2010.

[16] A. L. A. Freire, «COMPORTAMIENTO DE REDES OBS (OPTICAL BURST SWITCHING) CON TRÁFICO TCP,» QUITO, 2010.

[17] O. González de Dios, Ignacio de Miguel, V. López Álvarez, J. D. Ramón , . N. Merayo y J. F. Lobo Poyo, «ESTUDIO Y SIMULACIÓN DE TCP EN REDES DE CONMUTACIÓN ÓPTICA DE RÁFAGAS (OBS),» *XV Jornadas Telecom I+D,* Noviembre 2005.

[18] T. H. V. M. X. H. M. a. T. A. Y. M. Sun, «Design and implementation of an optical burst-switched network testbed,» *IEEE Commun,* vol. 43, nº 11, pp. S48 - S55, Noviembre 2005.

[19] C. Q. a. X. Y. Y. Chen, «Optical burst switching: A new area in optical networking research,» *IEEE Netw,* vol. 18, nº 3, pp. pp.16 -23, 2004.

[20] C. Q. a. M. Yoo, «Optical burst switching (OBS)-A new paradigm for an optical internet,» *J. High-Speed Netw.,* vol. 8, nº 1, pp. pp.69 -84, 1999.

[21] A. Rostami, «Traffic Shaping for Contention Control in OBS Networks,» Berlín, 2010.

[22] V. M. V. Jason P. Jue, Optical Burst Switched Networks, Boston: Springer Science+Bussiness Media, 2005.

[23] G. F. P. R. B. Héctor Cancela, «Fing,» 10 Agosto 2004. [En línea]. Available: www.fing.edu.uy/inco/grupos/invop/mh/material/pres4.ppt. [Último acceso: 27 Marzo 2015].

[24] J. A. M. Pérez, «Metaheurísticas: Concepto y Propiedades,» Santa Cruz de

Tenerife, 2004.

[25] S. Luke, «Essentials of Metaheuristics,» Virginia, 2014.

[26] J. R. M. F. N. Julio Mario Daza, «RESOLUCIÓN DEL PROBLEMA DE ENRUTAMIENTO DE VEHÍCULOS CON LIMITACIONES DE CAPACIDAD UTILIZANDO UN PROCEDIMIENTO METAHEURÍSTICO DE DOS FASES,» *EIA,* nº 12, pp. 23-38, 2009.

[27] «INTERNETZUGANG,» Medizinische Universität Innsbruck, 27 Agosto 2012. [En línea]. Available: https://www.i-med.ac.at/itservices/systeme/netzwerk/internet/. [Último acceso: 23 Julio 2015].

[28] C. Q. S. D. Myungsik Yoo, «A Comparative Study of Contention Resolution Policies in Optical Burst Switched WDM Networks,» Buffalo, 2000.

[29] M. K. J. S. Christoph M. Gauger, «Comparison of Contention Resolution Strategies in OBS Network Scenarios,» Stuttgart, 2004.

APÉNDICES

APÉNDICE A. ARCHIVOS GENERALES DE LOS SISTEMAS

En los siguientes apartados se muestra los códigos utilizados para los módulos comunes entre los sistemas, donde se presenta la función y configuración de los mismos.

A.1 MÓDULO APP

```
#ifdef _MSC_VER
#pragma warning(disable:4786)
#endif

#include <stdio.h>
#include <vector>
#include <omnetpp.h>
#include "Packet_m.h"


/**
 * Generador de tráfico para la red
 */
class App : public cSimpleModule
{
  private:
    // Variables de configuración.
    int myAddress;
    int numClass;
    long numSent;
    long numReceived;
    std::vector<int> destAddresses;
    cPar *sendIATime;
    cPar *packetLengthBytes;
    cMessage *generatePacket;
    long pkCounter;

    // Variables para obtención de señales.
    simsignal_t endToEndDelaySignal;
    simsignal_t hopCountSignal;
    simsignal_t sourceAddressSignal;

  public:
    App();
    virtual ~App();

  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
    virtual void updateDisplay();
};

Define_Module(App);
```

```
App::App()
{
    generatePacket = NULL;
}

App::~App()
{
    cancelAndDelete(generatePacket);
}

void App::initialize()
{
    //Inicialización de las variables en cero y asignación de valores.
    numSent = 0;
    numReceived = 0;
    myAddress = par("address");
    packetLengthBytes = &par("packetLength");
    sendIATime = &par("sendIaTime");
    pkCounter = 0;
    numClass = par("numClases");
    const char *destAddressesPar = par("destAddresses");
    cStringTokenizer tokenizer(destAddressesPar);
    const char *token;
    while ((token = tokenizer.nextToken())!=NULL)
        destAddresses.push_back(atoi(token));

    if (destAddresses.size() == 0)
        throw cRuntimeError("At least one address must be specified in
the destAddresses parameter!");
    generatePacket = new cMessage("nextPacket");
    //Tiempo entre generación de paquetes
    scheduleAt(sendIATime->doubleValue()/10, generatePacket);
    //Visualización las variables asociadas.
    WATCH(pkCounter);
    WATCH(myAddress);
    WATCH(numSent);
    WATCH(numReceived);
    endToEndDelaySignal = registerSignal("endToEndDelay");
    hopCountSignal =  registerSignal("hopCount");
    sourceAddressSignal = registerSignal("sourceAddress");
}

void App::handleMessage(cMessage *msg)
{
    if (msg == generatePacket)//Creación de Paquete.
    {
        // Envío de paquetes.
        //Se realiza la asignación de nombre, longitud de onda, tamaño
        del paquete y destino.
        int destAddress = destAddresses[intuniform(0,
destAddresses.size()-1)];
        int Class;
        int longOn = destAddresses[intuniform(0, destAddresses.size()-
11)] ;
        char pkname[40];
```

```
        sprintf(pkname,"pk-%d-to-%d-#%ld", myAddress, destAddress,
pkCounter++);
        EV << "generating packet " << pkname << endl;
        EV << "Longitud de onda asignada" << longOn << endl;
        EV << "tiempo" << sendIATime->doubleValue() << endl;
        Packet *pk = new Packet(pkname);
        pk->setByteLength(packetLengthBytes->longValue());
        pk->setSrcAddr(myAddress);
        pk->setLongOnd(longOn);
        Class = intuniform(0,numClass-1);
        pk->setKind(Class);
        pk->setDestAddr(destAddress);
        //Envío del paquete.
        send(pk,"out");
        if(simTime()>=0.5 && simTime()<=1.5){
            numSent++;
                    }
        scheduleAt((simTime()+(sendIATime->doubleValue())/10),
        generatePacket);
        if (ev.isGUI())
        getParentModule()->bubble("Generating packet...");
        updateDisplay();
    }
    else
    {
        // Manejo de paquetes entrantes.
        Packet *pk = check_and_cast<Packet *>(msg);
        EV << "received packet " << pk->getName() << " after " << pk-
        >getHopCount() << "hops" << endl;
        emit(endToEndDelaySignal, simTime() - pk->getCreationTime());
        emit(hopCountSignal, pk->getHopCount());
        emit(sourceAddressSignal, pk->getSrcAddr());
        if(simTime()>=0.5 && simTime()<=1.5){
            numReceived++;
                            }
        delete pk;
        if (ev.isGUI())
        {
        getParentModule()- >getDisplayString().setTagArg("i",1,"green");
        getParentModule()->bubble("Arrived!");
        updateDisplay();
        }
    }
}
void App::updateDisplay() //Visualización del número de paquetes enviados
y recibidos en el módulo.
{
    char buf[40];
    sprintf(buf, "rcvd: %ld sent: %ld", numReceived, numSent);
    getDisplayString().setTagArg("t",0,buf);

}
```

A.2 MODULO ROUTING

```
#ifdef _MSC_VER
#pragma warning(disable:4786)
#endif

#include <map>
#include <omnetpp.h>
#include "Packet_m.h"


/**
 * Enrutamiento del flujo de datos.
 */
class Routing : public cSimpleModule
{
  private:
    //Variables de configuración.
    int myAddress;
    int IndexM;
    int mayor=0;
    //Obtención de la tabla de enrutamiento
    typedef std::map<int,int> RoutingTable;
    RoutingTable rtable;

  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};

Define_Module(Routing);


void Routing::initialize()
{
    myAddress = getParentModule()->par("address");
    //Obtención de la topología para cada nodo.
    cTopology *topo = new cTopology("topo");
    std::vector<std::string> nedTypes;
    nedTypes.push_back(getParentModule()->getNedTypeName());
    topo->extractByNedTypeName(nedTypes);
    EV << "cTopology found " << topo->getNumNodes() << " nodes\n";
    cTopology::Node *thisNode = topo->getNodeFor(getParentModule());

    // Encontrar y almacenar la ruta.
    for (int i=0; i<topo->getNumNodes(); i++)
    {
        if (topo->getNode(i)==thisNode) continue;
        topo->calculateUnweightedSingleShortestPathsTo(topo->getNode(i));
        if (thisNode->getNumPaths()==0) continue;
        cGate *parentModuleGate = thisNode->getPath(0)->getLocalGate();
        int gateIndex = parentModuleGate->getIndex();
        int address = topo->getNode(i)->getModule()->par("address");
        rtable[address] = gateIndex;
        EV << "  towards address " << address << " gateIndex is " <<
        gateIndex << endl;
```

```
    if(mayor<=gateIndex){
        mayor=gateIndex;
    }

    }
    delete topo;
}

void Routing::handleMessage(cMessage *msg)
{
    //Manejo y envío del mensaje.
    Packet *pk = check_and_cast<Packet *>(msg);
    int destAddr = pk->getDestAddr();
    if (destAddr == myAddress)
    {
        EV << "local delivery of packet " << pk->getName() << endl;
        send(pk, "localOut");
        return;
    }
    RoutingTable::iterator it = rtable.find(destAddr);
    if (it==rtable.end())
    {
        EV << "address " << destAddr << " unreachable, discarding packet
" << pk->getName() << endl;
        delete pk;
        return;
    }
    int outGateIndex = (*it).second;
    EV << "forwarding packet " << pk->getName() << " on gate index " <<
outGateIndex << endl;
    pk->setHopCount(pk->getHopCount()+1);
    IndexM = outGateIndex;
    pk->setIndex(IndexM);
    send(pk, "out", outGateIndex);

}
```

A.3 MODULO CLASIFICADOR

```
#include <omnetpp.h>
#include <map>
#include "Packet_m.h"

class Clasificador : public cSimpleModule
{
    private:
        //Variables para la clasificación por longitud de onda.
        int LamdaPaquete;
        long lamda0;long lamda1;long lamda2;long lamda3; long lamda4;long
lamda5;
        long lamda6;long lamda7;long lamda8;long lamda9;long lamda10;long
lamda11;
        long lamda12;long lamda13;long lamda14;long lamda15;

    protected:
        virtual void initialize();
```

```
        virtual void handleMessage(cMessage *msg);
};

Define_Module(Clasificador);


void Clasificador::initialize()
{
    //Inicialización de las variables en cero.
    LamdaPaquete = 0;
    lamda0 = 0;lamda1 = 0;lamda2 = 0;lamda3 = 0; lamda4 = 0;lamda5 = 0;
    lamda6 = 0;lamda7 = 0;lamda8 = 0;lamda9 = 0;lamda10 = 0;lamda11 = 0;
    lamda12 = 0;lamda13 = 0;lamda14 = 0;lamda15 = 0;

}

void Clasificador::handleMessage(cMessage *msg)
{
    //Manejo del mensaje, obtención y clasificación por longitud de onda.
    Packet *pk = check_and_cast<Packet *>(msg);
    int longitudOn = pk->getLongOnd();
    //Envío del mensaje por longitud de onda asignada.
    LamdaPaquete = longitudOn;
    EV << "Direccion de destino "<< pk->getDestAddr()<<".\n";
    EV << "Emitiendo con Î» = "<< pk->getLongOnd() << ".\n";
    if(LamdaPaquete==0){ev<< "Salida por λ0" <<
".\n";send(pk,"salidaClas",0);lamda0++;}
    if(LamdaPaquete==1){ev<< "Salida por λ1" <<
".\n";send(pk,"salidaClas",1);lamda1++;}
    if(LamdaPaquete==2){ev<< "Salida por λ2" <<
".\n";send(pk,"salidaClas", 2);lamda2++;}
    if(LamdaPaquete==3){ev<< "Salida por λ3" << ".\n";send(pk,
"salidaClas",3);lamda3++;}
    if(LamdaPaquete==4){ev<< "Salida por λ4" <<
".\n";send(pk,"salidaClas", 4);lamda4++;}
    if(LamdaPaquete==5){ev<< "Salida por λ5" <<
".\n";send(pk,"salidaClas", 5);lamda5++;}
    if(LamdaPaquete==6){ev<< "Salida por λ6" <<
".\n";send(pk,"salidaClas",6);lamda6++;}
    if(LamdaPaquete==7){ev<< "Salida por λ7" <<
".\n";send(pk,"salidaClas",7);lamda7++;}
    if(LamdaPaquete==8){ev<< "Salida por λ8" <<
".\n";send(pk,"salidaClas", 8);lamda8++;}
    if(LamdaPaquete==9){ev<< "Salida por λ9" << ".\n";send(pk,
"salidaClas",9);lamda9++;}
    if(LamdaPaquete==10){ev<< "Salida por λ10" <<
".\n";send(pk,"salidaClas", 10);lamda10++;}
    if(LamdaPaquete==11){ev<< "Salida por λ11" <<
".\n";send(pk,"salidaClas", 11);lamda11++;}
    if(LamdaPaquete==12){ev<< "Salida por λ12" <<
".\n";send(pk,"salidaClas", 12);lamda12++;}
    if(LamdaPaquete==13){ev<< "Salida por λ13" << ".\n";send(pk,
"salidaClas",13);lamda13++;}
    if(LamdaPaquete==14){ev<< "Salida por λ14" <<
".\n";send(pk,"salidaClas", 14);lamda14++;}

        if(ev.isGUI()){
```

```
            char buf[40];
                sprintf(buf2, "\n \n \n λ8: %ld \n λ9: %ld \n λ10: %ld \n
λ11: %ld \n λ12: %ld \n λ13: %ld \n λ14: %ld \n λ15: %ld", lamda0,
lamda1, lamda2, lamda3, lamda4, lamda5, lamda6, lamda7 , lamda8, lamda9,
lamda10, lamda11, lamda12, lamda13, lamda14, lamda15);
                getDisplayString().setTagArg("t",0,buf);

        }
    }
```

A.4 MODULO QUEUE

```
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>


/**
 * Manejo del encolamiento
 */
class L2Queue : public cSimpleModule
{
  private:
    //Variables de estado.
    long frameCapacity;
    volatile double serviceTime;
    cQueue queue;
    cMessage *endTransmissionEvent;
  public:
    L2Queue();
    virtual ~L2Queue();

  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
    virtual void startTransmitting(cMessage *msg);
    virtual void displayStatus(bool isBusy);
};

Define_Module(L2Queue);

L2Queue::L2Queue()
{
    endTransmissionEvent = NULL;
}

L2Queue::~L2Queue()
{
    cancelAndDelete(endTransmissionEvent);
}

void L2Queue::initialize()
{
    //Asignación de valores y nombres para las variables de estado.
    queue.setName("queue");
    endTransmissionEvent = new cMessage("endTxEvent");
```

```
    frameCapacity = par("frameCapacity");
}

void L2Queue::startTransmitting(cMessage *msg)
{
    //Transmisión de los mensajes.
    if (ev.isGUI()) displayStatus(true);
    EV << "Starting transmission of " << msg << endl;
    int64 numBytes = check_and_cast<cPacket *>(msg)->getByteLength();
    send(msg, "line$o");
    // Estudio del mensaje, para obtener el tiempo en el que el último
    bit se ha enviado.
    simtime_t endTransmission = gate("line$o")->getTransmissionChannel()-
     >getTransmissionFinishTime();
    scheduleAt(endTransmission, endTransmissionEvent);
}

void L2Queue::handleMessage(cMessage *msg)
{
    //Manejo de la información para establecer las condiciones de
    encolamiento.
    if (msg==endTransmissionEvent)
    {
        EV << "Transmission finished.\n";
        if (ev.isGUI()) displayStatus(false);
        if (queue.empty())
        {
            emit(busySignal, 0);
        }
        else
        {
            msg = (cMessage *) queue.pop();
            emit(queueingTimeSignal, simTime() - msg->getTimestamp());
            emit(qlenSignal, queue.length());
            startTransmitting(msg);
        }
    }
    else if (msg->arrivedOn("line$i"))
    {
    emit(rxBytesSignal, (long)check_and_cast<cPacket *>(msg)-
     >getByteLength());
        send(msg,"out");
    }
    else
    {
        if (endTransmissionEvent->isScheduled())
        {
            if (frameCapacity && queue.length()>=frameCapacity)
            {
           EV << "Received " << msg << " but transmitter busy and queue
           full: discarding\n";
            emit(dropSignal, (long)check_and_cast<cPacket *>(msg)-
           >getByteLength());
                delete msg;
            }
            else
            {
```

```
            EV << "Received " << msg << " but transmitter busy: queueing
            up\n";
                msg->setTimestamp();
                queue.insert(msg);
                emit(qlenSignal, queue.length());
            }
        }
        else
        {
            EV << "Received " << msg << endl;
            emit(queueingTimeSignal, 0.0);
            startTransmitting(msg);
            emit(busySignal, 1);
        }
    }
}

void L2Queue::displayStatus(bool isBusy)
{
    //Visualización de los estados del módulo.
    getDisplayString().setTagArg("t",0, isBusy ? "transmitting" :
    "idle");
    getDisplayString().setTagArg("i",1, isBusy ? (queue.length()>=3 ?
    "red" : "yellow") : "");
}
```

APENDICE B. ARCHIVOS ESPECIFICOS DE CADA SISTEMA

En los siguientes apartados se muestra los respectivos códigos para cada método de control de contención.

B.1 MODULO DE CONTENCIÓN CAMBIO LONGITUD DE ONDA

```
#ifdef _MSC_VER
#pragma warning(disable:4786)
#endif

#include <vector>
#include <omnetpp.h>
#include <map>
#include "Packet_m.h"
#include <cstdio>
#include <cstdlib>
#include <stdio.h>

class Contencion : public cSimpleModule
{
    private:
        //Variables para el manejo de la contención.
        int condicion;
        int LamdaPaquete;
        long contador0;long contador1;long contador2;long contador3;long
contador4;long contador5;long contador6;long contador7;
        long contador8;long contador9;long contador10;long
contador11;long contador12;long contador13;long contador14;long
contador15;
        long DelT;
        long Del0;long Del1;long Del2;long Del3;long Del4;long Del5;long
Del6;long Del7;
        long Del8;long Del9;long Del10;long Del11;long Del12;long
Del13;long Del14;long Del15;
        int i;
        std::vector<int> destAddresses;
    protected:
        virtual void initialize();
        virtual void handleMessage(cMessage *msg);
};

Define_Module(Contencion);

void Contencion::initialize()
{
    //Inicializar las variables en cero.
    contador0 =
0;contador1=0;contador2=0;contador3=0;contador4=0;contador5=0;
    contador6 = 0;contador7=0;contador8 = 0;contador9=0;contador10 =
0;contador11=0;contador12 = 0;contador13=0;contador14 = 0;contador15=0;
    LamdaPaquete = 0;
    condicion = 0;
    DelT=0;
```

```
Del0=0;Del1=0;Del2=0;Del3=0;Del4=0;Del5=0;Del6=0;Del7=0;Del8=0;Del9=0;Del
10=0;Del11=0;Del12=0;Del13=0;Del14=0;Del15=0;
    const char *destAddressesPar = par("destAddresses");
       cStringTokenizer tokenizer(destAddressesPar);
       const char *token;
       while ((token = tokenizer.nextToken())!=NULL)
       destAddresses.push_back(atoi(token));
}

void Contencion::handleMessage(cMessage *msg)
{
    //Manejo del paquete, obtención de variables para realizar el proceso
     de control de contención.
    Packet *pk = check_and_cast<Packet *>(msg);
    int IndexOut = pk->getIndex();
    int lamda = pk->getLongOnd();
    condicion = lamda;
    //Lógica del control de contención.
    if(condicion==0){
            contador0++ ;
        if(contador0==1){
            EV << "Enviando con longitud de onda ="<<lamda<< "
original.\n";
            send(pk,"out",IndexOut);
               }
        else if(contador0>1 && contador0<14){
            do{
            int NuevaLamda = destAddresses[intuniform(0,
destAddresses.size()-11)];
            LamdaPaquete = NuevaLamda;
            EV << "Cambio de longitud de onda "<< lamda <<" a "
<<LamdaPaquete<< ".\n";
            pk->setLongOnd(LamdaPaquete);
            EV << "Nueva longitud de onda ="<<LamdaPaquete<< " .\n";
            } while(LamdaPaquete==0);
            send(pk,"out",IndexOut);
            }
        else if(contador0==14){
            EV << "Longitud de onda"<<condicion<< " congestionada .\n";
            EV << "Paquete Descartado" << pk->getName() <<" en tiempo de
simulación"<< simTime()<< "~Fuente->"<< pk->getSrcAddr<<endl;
            delete pk;
            if(simTime()>=0.5 && simTime()<=1.5){
            Del0++;
            }
            contador0=0;
        }
    }

    if(condicion==1){
            contador1++ ;
        if(contador1==1){
            EV << "Enviando con longitud de onda ="<<lamda<< "
original.\n";
            send(pk,"out",IndexOut);
            }
```

```
        else if(contador1>1 && contador1<14){
            do{
            int NuevaLamda = destAddresses[intuniform(0,
destAddresses.size()-11)];
            LamdaPaquete = NuevaLamda;
            EV << "Cambio de longitud de onda "<< lamda <<" a "
<<LamdaPaquete<< ".\n";
            pk->setLongOnd(LamdaPaquete);
            EV << "Nueva longitud de onda ="<<LamdaPaquete<< " .\n";
            } while(LamdaPaquete==1);
            send(pk,"out",IndexOut);
            }
        else if (contador1==14){
            EV << "Longitud de onda"<<condicion<< " congestionada .\n";
            EV << "Paquete Descartado" << pk->getName() << " en tiempo de
simulación"<< simTime()<< "~Fuente->"<< pk->getSrcAddr<<endl;
            delete pk;
            if(simTime()>=0.5 && simTime()<=1.5){
            Del1++;
            }
            contador1=0;
            }
            }

    if(condicion==2){
           contador2++ ;
        if(contador2==1){
            EV << "Enviando con longitud de onda ="<<lamda<< "
original.\n";
            send(pk,"out",IndexOut);
            }
        else if(contador2>1 && contador2<14){
            do{
            int NuevaLamda = destAddresses[intuniform(0,
destAddresses.size()-11)];
            LamdaPaquete = NuevaLamda;
            EV << "Cambio de longitud de onda "<< lamda <<" a "
<<LamdaPaquete<< ".\n";
            pk->setLongOnd(LamdaPaquete);
            EV << "Nueva longitud de onda ="<<LamdaPaquete<< " .\n";
            }while(LamdaPaquete==2);
            send(pk,"out",IndexOut);
            }
        else if (contador2==14){
            EV << "Longitud de onda"<<condicion<< " congestionada .\n";
            EV << "Paquete Descartado" << pk->getName() << "en tiempo de
simulación"<< simTime()<< "~Fuente->"<< pk->getSrcAddr<<endl;
            delete pk;
            if(simTime()>=0.5 && simTime()<=1.5){
            Del2++;
            }
            contador2=0;
            }
            }
    if(condicion==3){
            contador3++ ;
        if(contador3==1){
```

```
            EV << "Enviando con longitud de onda ="<<lamda<< "
original.\n";
            send(pk,"out",IndexOut);
            }
        else if(contador3>1 && contador3<14){
             do{
             int NuevaLamda = destAddresses[intuniform(0,
destAddresses.size()-11)];
             LamdaPaquete = NuevaLamda;
             EV << "Cambio de longitud de onda "<< lamda <<" a "
<<LamdaPaquete<< ".\n";
             pk->setLongOnd(LamdaPaquete);
             EV << "Nueva longitud de onda ="<<LamdaPaquete<< " .\n";
             } while(LamdaPaquete==3);
             send(pk,"out",IndexOut);
             }
        else if (contador3==14){
             EV << "Longitud de onda"<<condicion<< " congestionada .\n";
             EV << "Paquete Descartado" << pk->getName() << "en tiempo de
simulación"<< simTime()<< "~Fuente->"<< pk->getSrcAddr<<endl;
             delete pk;
             if(simTime()>=0.5 && simTime()<=1.5){
             Del3++;
             }
             contador3=0;
             }
             }
    if(condicion==4){
            contador4++ ;
        if(contador4==1){
            EV << "Enviando con longitud de onda ="<<lamda<< "
original.\n";
            send(pk,"out",IndexOut);
            }
        else if(contador4>1 && contador4<14){
            do{
             int NuevaLamda = destAddresses[intuniform(0,
destAddresses.size()-11)];
             LamdaPaquete = NuevaLamda;
             EV << "Cambio de longitud de onda "<< lamda <<" a "
<<LamdaPaquete<< ".\n";
             pk->setLongOnd(LamdaPaquete);
             EV << "Nueva longitud de onda ="<<LamdaPaquete<< " .\n";
             }while(LamdaPaquete==4);
             send(pk,"out",IndexOut);
             }
            else if (contador4==14){
             EV << "Longitud de onda"<<condicion<< " congestionada .\n";
             EV << "Paquete Descartado" << pk->getName() << "en tiempo de
simulación"<< simTime()<< "~Fuente->"<< pk->getSrcAddr<<endl;
             delete pk;
             if(simTime()>=0.5 && simTime()<=1.5){
             Del4++;
             }
             contador4=0;
             }
             }
```

```
    if(condicion==5){
              contador5++ ;
         if(contador5==1){
              EV << "Enviando con longitud de onda ="<<lamda<< "
original.\n";
              send(pk,"out",IndexOut);
              }
          else if(contador5>1 && contador5<14){
              do{
              int NuevaLamda = destAddresses[intuniform(0,
destAddresses.size()-11)];
              LamdaPaquete = NuevaLamda;
              EV << "Cambio de longitud de onda "<< lamda <<" a "
<<LamdaPaquete<< ".\n";
              pk->setLongOnd(LamdaPaquete);
              EV << "Nueva longitud de onda ="<<LamdaPaquete<< " .\n";
              }while(LamdaPaquete == 5);
              send(pk,"out",IndexOut);
              }
          else if (contador5==14){
              EV << "Longitud de onda"<<condicion<< " congestionada .\n";
              EV << "Paquete Descartado" << pk->getName() << "en tiempo de
simulación"<< simTime()<< "~Fuente->"<< pk->getSrcAddr<<endl;
              delete pk;
              if(simTime()>=0.5 && simTime()<=1.5){
              Del5++;
              EV << "Contador";
              }
              contador5=0;
              }
              }
    if(condicion==6){
              contador6++ ;
         if(contador6==1){
              EV << "Enviando con longitud de onda ="<<lamda<< "
original.\n";
              send(pk,"out",IndexOut);
              }
          else if(contador6>1 && contador6<14){
              do{
              int NuevaLamda = destAddresses[intuniform(0,
destAddresses.size()-11)];
              LamdaPaquete = NuevaLamda;
              EV << "Cambio de longitud de onda "<< lamda <<" a "
<<LamdaPaquete<< ".\n";
              pk->setLongOnd(LamdaPaquete);
              EV << "Nueva longitud de onda ="<<LamdaPaquete<< " .\n";
              } while(LamdaPaquete==6);
              send(pk,"out",IndexOut);
              }
          else if(contador6==14){
              EV << "Longitud de onda"<<condicion<< " congestionada .\n";
              EV << "Paquete Descartado" << pk->getName() <<" en tiempo de
simulación"<< simTime()<< "~Fuente->"<< pk->getSrcAddr<<endl;
              delete pk;
              if(simTime()>=0.5 && simTime()<=1.5){
              Del6++;
```

```
            }
            contador6=0;
            }
            }
    if(condicion==7){
            contador7++ ;
         if(contador7==1){
            EV << "Enviando con longitud de onda ="<<lamda<< "
original.\n";
            send(pk,"out",IndexOut);
            }
         else if(contador7>1 && contador7<14){
            do{
            int NuevaLamda = destAddresses[intuniform(0,
destAddresses.size()-11)];
            LamdaPaquete = NuevaLamda;
            EV << "Cambio de longitud de onda "<< lamda <<" a "
<<LamdaPaquete<< ".\n";
            pk->setLongOnd(LamdaPaquete);
            EV << "Nueva longitud de onda ="<<LamdaPaquete<< " .\n";
            } while(LamdaPaquete==7);
            send(pk,"out",IndexOut);
            }
         else if (contador7==14){
            EV << "Longitud de onda"<<condicion<< " congestionada .\n";
            EV << "Paquete Descartado" << pk->getName() << " en tiempo
de simulación"<< simTime()<< "~Fuente->"<< pk->getSrcAddr<<endl;
            delete pk;
            if(simTime()>=0.5 && simTime()<=1.5){
            Del7++;
            }
            contador7=0;
            }
            }
    if(condicion==8){
            contador8++ ;
         if(contador8==1){
            EV << "Enviando con longitud de onda ="<<lamda<< "
original.\n";
            send(pk,"out",IndexOut);
            }
         else if(contador8>1 && contador8<14){
            do{
            int NuevaLamda = destAddresses[intuniform(0,
destAddresses.size()-11)];
            LamdaPaquete = NuevaLamda;
            EV << "Cambio de longitud de onda "<< lamda <<" a "
<<LamdaPaquete<< ".\n";
            pk->setLongOnd(LamdaPaquete);
            EV << "Nueva longitud de onda ="<<LamdaPaquete<< " .\n";
            } while(LamdaPaquete==8);
            send(pk,"out",IndexOut);
            }
         else if (contador8==14){
            EV << "Longitud de onda"<<condicion<< " congestionada .\n";
            EV << "Paquete Descartado" << pk->getName() << " en tiempo
de simulación"<< simTime()<< "~Fuente->"<< pk->getSrcAddr<<endl;
```

```
                delete pk;
                if(simTime()>=0.5 && simTime()<=1.5){
                Del8++;
                }
                contador8=0;
                }
                }
    if(condicion==9){
                contador9++ ;
          if(contador9==1){
                EV << "Enviando con longitud de onda ="<<lamda<< "
original.\n";
                send(pk,"out",IndexOut);
                }
          else if(contador9>1 && contador9<14){
                do{
                int NuevaLamda = destAddresses[intuniform(0,
destAddresses.size()-11)];
                LamdaPaquete = NuevaLamda;
                EV << "Cambio de longitud de onda "<< lamda <<" a "
<<LamdaPaquete<< ".\n";
                pk->setLongOnd(LamdaPaquete);
                EV << "Nueva longitud de onda ="<<LamdaPaquete<< " .\n";
                } while(LamdaPaquete==9);
                send(pk,"out",IndexOut);
                }
          else if (contador9==14){
                EV << "Longitud de onda"<<condicion<< " congestionada .\n";
                EV << "Paquete Descartado" << pk->getName() << " en tiempo
de simulación"<< simTime()<< "~Fuente->"<< pk->getSrcAddr<<endl;
                delete pk;
                if(simTime()>=0.5 && simTime()<=1.5){
                Del9++;
                }
                contador9=0;
                }
                }
    if(condicion==10){
                contador10++ ;
          if(contador10==1){
                EV << "Enviando con longitud de onda ="<<lamda<< "
original.\n";
                send(pk,"out",IndexOut);
                }
          else if(contador10>1 && contador10<14){
                do{
                int NuevaLamda = destAddresses[intuniform(0,
destAddresses.size()-11)];
                LamdaPaquete = NuevaLamda;
                EV << "Cambio de longitud de onda "<< lamda <<" a "
<<LamdaPaquete<< ".\n";
                pk->setLongOnd(LamdaPaquete);
                EV << "Nueva longitud de onda ="<<LamdaPaquete<< " .\n";
                }while(LamdaPaquete==10);
                send(pk,"out",IndexOut);
                }
          else if (contador10==14){
```

```
            EV << "Longitud de onda"<<condicion<< " congestionada .\n";
            EV << "Paquete Descartado" << pk->getName() << "en tiempo de
simulación"<< simTime()<< "~Fuente->"<< pk->getSrcAddr<<endl;
            delete pk;
            if(simTime()>=0.5 && simTime()<=1.5){
            Del10++;
            }
            contador10=0;
            }
            }
    if(condicion==11){
            contador11++ ;
         if(contador11==1){
            EV << "Enviando con longitud de onda ="<<lamda<< "
original.\n";
            send(pk,"out",IndexOut);
            }
         else if(contador11>1 && contador11<14){
            do{
            int NuevaLamda = destAddresses[intuniform(0,
destAddresses.size()-11)];
            LamdaPaquete = NuevaLamda;
            EV << "Cambio de longitud de onda "<< lamda <<" a "
<<LamdaPaquete<< ".\n";
            pk->setLongOnd(LamdaPaquete);
            EV << "Nueva longitud de onda ="<<LamdaPaquete<< " .\n";
            } while(LamdaPaquete==11);
            send(pk,"out",IndexOut);
            }
         else if (contador11==14){
            EV << "Longitud de onda"<<condicion<< " congestionada .\n";
            EV << "Paquete Descartado" << pk->getName() << "en tiempo de
simulación"<< simTime()<< "~Fuente->"<< pk->getSrcAddr<<endl;
            delete pk;
            if(simTime()>=0.5 && simTime()<=1.5){
            Del11++;
            }
            contador11=0;
            }
            }
    if(condicion==12){
            contador12++ ;
         if(contador12==1){
            EV << "Enviando con longitud de onda ="<<lamda<< "
original.\n";
            send(pk,"out",IndexOut);
            }
         else if(contador12>1 && contador12<14){
            do{
            int NuevaLamda = destAddresses[intuniform(0,
destAddresses.size()-11)];
            LamdaPaquete = NuevaLamda;
            EV << "Cambio de longitud de onda "<< lamda <<" a "
<<LamdaPaquete<< ".\n";
            pk->setLongOnd(LamdaPaquete);
            EV << "Nueva longitud de onda ="<<LamdaPaquete<< " .\n";
            }while(LamdaPaquete==12);
```

```
            send(pk,"out",IndexOut);
            }
        else if (contador12==14){
            EV << "Longitud de onda"<<condicion<< " congestionada .\n";
            EV << "Paquete Descartado" << pk->getName() << "en tiempo de
simulación"<< simTime()<< "~Fuente->"<< pk->getSrcAddr<<endl;
            delete pk;
            if(simTime()>=0.5 && simTime()<=1.5){
            Del12++;
            }
            contador12=0;
            }
            }
    if(condicion==13){
            contador13++ ;
        if(contador13==1){
            EV << "Enviando con longitud de onda ="<<lamda<< "
original.\n";
            send(pk,"out",IndexOut);
            }
        else if(contador13>1 && contador13<14){
            do{
            int NuevaLamda = destAddresses[intuniform(0,
destAddresses.size()-11)];
            LamdaPaquete = NuevaLamda;
            EV << "Cambio de longitud de onda "<< lamda <<" a "
<<LamdaPaquete<< ".\n";
            pk->setLongOnd(LamdaPaquete);
            EV << "Nueva longitud de onda ="<<LamdaPaquete<< " .\n";
            }while(LamdaPaquete == 13);
            send(pk,"out",IndexOut);
            }
        else if (contador13==14){
            EV << "Longitud de onda"<<condicion<< " congestionada .\n";
            EV << "Paquete Descartado" << pk->getName() << "en tiempo de
simulación"<< simTime()<< "~Fuente->"<< pk->getSrcAddr<<endl;
            delete pk;
            if(simTime()>=0.5 && simTime()<=1.5){
            Del13++;
            }
            contador13=0;
            }
            }
    if(condicion==14){
            contador14++ ;
        if(contador14==1){
            EV << "Enviando con longitud de onda ="<<lamda<< "
original.\n";
            send(pk,"out",IndexOut);
            }
        else if(contador14>1 && contador14<14){
            do{
            int NuevaLamda = destAddresses[intuniform(0,
destAddresses.size()-11)];
            LamdaPaquete = NuevaLamda;
            EV << "Cambio de longitud de onda "<< lamda <<" a "
<<LamdaPaquete<< ".\n";
```

```
            pk->setLongOnd(LamdaPaquete);
            EV << "Nueva longitud de onda ="<<LamdaPaquete<< " .\n";
            } while(LamdaPaquete==14);
            send(pk,"out",IndexOut);
            }
        else if(contador14==14){
            EV << "Longitud de onda"<<condicion<< " congestionada .\n";
            EV << "Paquete Descartado" << pk->getName() <<" en tiempo de
simulación"<< simTime()<< "~Fuente->"<< pk->getSrcAddr<<endl;
            delete pk;
            if(simTime()>=0.5 && simTime()<=1.5){
            Del14++;
            }
            contador14=0;
            }
            }
    if(condicion==15){
            contador15++ ;
        if(contador15==1){
            EV << "Enviando con longitud de onda ="<<lamda<< "
original.\n";
            send(pk,"out",IndexOut);
            }
        else if(contador15>1 && contador15<14){
            do{
            int NuevaLamda = destAddresses[intuniform(0,
destAddresses.size()-11)];
            LamdaPaquete = NuevaLamda;
            EV << "Cambio de longitud de onda "<< lamda <<" a "
<<LamdaPaquete<< ".\n";
            pk->setLongOnd(LamdaPaquete);
            EV << "Nueva longitud de onda ="<<LamdaPaquete<< " .\n";
            } while(LamdaPaquete==15);
            send(pk,"out",IndexOut);
            }
        else if (contador15==14){
            EV << "Longitud de onda"<<condicion<< " congestionada .\n";
            EV << "Paquete Descartado" << pk->getName() << " en tiempo
de simulación"<< simTime()<< "~Fuente->"<< pk->getSrcAddr<<endl;
            delete pk;
            if(simTime()>=0.5 && simTime()<=1.5){
            Del15++;
            }
            contador15=0;
            }
            }
//Contador para los mensajes eliminados.
DelT=Del0+Del1+Del2+Del3+Del4+Del5+Del6+Del7+Del8+Del9+Del10+Del11+Del12+
Del13+Del14+Del15;
if(ev.isGUI()){
//Visualización de las loongitudes de onda usadas y paquetes eliminados.
     char buf[40];
     sprintf(buf, "λ0: %ld \n λ1: %ld \n λ2: %ld \n λ3: %ld \n λ4: %ld \n
     λ5: %ld \n λ6: %ld \n λ7: %ld \n λ8: %ld \n λ9: %ld \n λ10: %ld \n
     λ11: %ld \n λ12: %ld \n λ13: %ld \n λ14: %ld \n λ15: %ld \n DEL:
     %ld", contador0, contador1, contador2, contador3, contador4,
```

```
     contador5, contador6, contador7,contador8, contador9, contador10,
     contador11, contador12, contador13, contador14, contador15, DelT);
     getDisplayString().setTagArg("t",0,buf);
}
}
```

B.2 MODULO DE CONTENCION FDL

```
#ifdef _MSC_VER
#pragma warning(disable:4786)
#endif

#include <vector>
#include <omnetpp.h>
#include <map>
#include "Packet_m.h"
#include <cstdio>
#include <cstdlib>
#include <stdio.h>

class Contencion : public cSimpleModule
{
    private:
        //Variables para el manejo de la contención por FDL.
        int condicion;
        int LamdaPaquete;
        double DelayT;
        double DueDelayT;
        double TreDelayT;
        double CuatDelayT;
        long contador0;long contador1;long contador2;long contador3;long
contador4;long contador5;
        long DelT;
        long Del0;long Del1;long Del2;long Del3;long Del4;long Del5;
        int i;
        std::vector<int> destAddresses;
    protected:
        virtual void initialize();
        virtual void handleMessage(cMessage *msg);
};

Define_Module(Contencion);


void Contencion::initialize()
{
//Inicialización de las variables en cero.
contador0 = 0;contador1=0;contador2=0;contador3=0;contador4=0;contador5=0;
    LamdaPaquete = 0;
    condicion = 0;
    DelT=0;
    Del0=0;Del1=0;Del2=0;Del3=0;Del4=0;Del5=0;
    const char *destAddressesPar = par("destAddresses");
       cStringTokenizer tokenizer(destAddressesPar);
       const char *token;
       while ((token = tokenizer.nextToken())!=NULL)
```

```
        destAddresses.push_back(atoi(token));
}

void Contencion::handleMessage(cMessage *msg)
{
    //Manejo del mensaje, obtención de los parámtros necesarios para
    realizar el control de contención.
    Packet *pk = check_and_cast<Packet *>(msg);
    int IndexOut = pk->getIndex();
    int lamda = pk->getLongOnd();
    double Delay = pk->getByteLength();
    DelayT = (Delay/10000000); //Retardo primer FDL.
    DueDelayT = 2*DelayT;//Retardo segunda FDL.
    TreDelayT = 3*DelayT;//Retardo tecer FDL.
    CuatDelayT = 4*DelayT;//Retardo cuarta FDL.
    condicion = lamda;
    if(condicion==0)
    {
        contador0++ ;
        if(contador0==1){
            EV << "Enviando sin usar FDL.\n";
            send(pk,"out",IndexOut);
               }
        else if(contador0==2){
            EV << "Enviando por FDL con retardo agregado de
"<<DelayT<<".\n";
            EV << "Retardo primer contención: "<<DelayT<< ".\n";
            sendDelayed(pk,DelayT,"out",IndexOut);
            }
        else if(contador0==3){
            EV << "Enviando por FDL con retardo agregado de
"<<DueDelayT<<".\n";
            EV << "Retardo segunda contención: "<<DueDelayT<< ".\n";
            sendDelayed(pk,DueDelayT,"out",IndexOut);
            }
        else if(contador0==4){
            EV << "Enviando por FDL con retardo agregado de
"<<TreDelayT<<".\n";
            EV << "Retardo tercera contención: "<<TreDelayT<< ".\n";
            sendDelayed(pk,TreDelayT,"out",IndexOut);
            }
        else if(contador0==5){
            EV << "Enviando por FDL con retardo agregado de
"<<CuatDelayT<<".\n";
            EV << "Retardo cuarta contención: "<<CuatDelayT<< ".\n";
            sendDelayed(pk,CuatDelayT,"out",IndexOut);
            }
        else if(contador0==6){
            EV << "FDL congestionada .\n";
            EV << "Paquete Descartado" << pk->getName() <<" en tiempo de
simulaciÃ³n"<< simTime()<< "~Fuente->"<< pk->getSrcAddr<<endl;
            delete pk;
            if(simTime()>=0.5 && simTime()<=1.5){
            Del0++;
            }
            contador0=0;
        }
```

```
    }

    if(condicion==1)
    {
        contador1++ ;
        if(contador1==1){
            EV << "Enviando con longitud de onda ="<<lamda<< "
original.\n";
            send(pk,"out",IndexOut);
               }
        else if(contador1==2){
            EV << "Enviando por FDL con retardo agregado de
"<<DelayT<<".\n";
            EV << "Retardo primer contención: "<<DelayT<< ".\n";
            sendDelayed(pk,DelayT,"out",IndexOut);
            }
        else if(contador1==3){
            EV << "Enviando por FDL con retardo agregado de
"<<DueDelayT<<".\n";
            EV << "Retardo segunda contención: "<<DueDelayT<< ".\n";
            sendDelayed(pk,DueDelayT,"out",IndexOut);
            }
        else if(contador1==4){
            EV << "Enviando por FDL con retardo agregado de
"<<TreDelayT<<".\n";
            EV << "Retardo tercera contención: "<<TreDelayT<< ".\n";
            sendDelayed(pk,TreDelayT,"out",IndexOut);
            }
        else if(contador1==5){
            EV << "Enviando por FDL con retardo agregado de
"<<CuatDelayT<<".\n";
            EV << "Retardo cuarta contención: "<<CuatDelayT<< ".\n";
            sendDelayed(pk,CuatDelayT,"out",IndexOut);
            }
        else if(contador1==6){
            EV << "FDL congestionada .\n";
            EV << "Paquete Descartado" << pk->getName() <<" en tiempo de
simulaciÃ³n"<< simTime()<< "~Fuente->"<< pk->getSrcAddr<<endl;
            delete pk;
            if(simTime()>=0.5 && simTime()<=1.5){
            Del1++;
            }
            contador1=0;
        }
    }
    if(condicion==2)
    {
        contador2++ ;
        if(contador2==1){
            EV << "Enviando con longitud de onda ="<<lamda<< "
original.\n";
            send(pk,"out",IndexOut);
            }
        else if(contador2==2){
            EV << "Enviando por FDL con retardo agregado de
"<<DelayT<<".\n";
            EV << "Retardo primer contención: "<<DelayT<< ".\n";
```

```
            sendDelayed(pk,DelayT,"out",IndexOut);
            }
        else if(contador2==3){
            EV << "Enviando por FDL con retardo agregado de
"<<DueDelayT<<".\n";
            EV << "Retardo segunda contención: "<<DueDelayT<< ".\n";
            sendDelayed(pk,DueDelayT,"out",IndexOut);
            }
        else if(contador2==4){
            EV << "Enviando por FDL con retardo agregado de
"<<TreDelayT<<".\n";
            EV << "Retardo tercera contención: "<<TreDelayT<< ".\n";
            sendDelayed(pk,TreDelayT,"out",IndexOut);
            }
        else if(contador2==5){
            EV << "Enviando por FDL con retardo agregado de
"<<CuatDelayT<<".\n";
            EV << "Retardo cuarta contención: "<<CuatDelayT<< ".\n";
            sendDelayed(pk,CuatDelayT,"out",IndexOut);
            }
        else if(contador2==6){
            EV << "FDL congestionada .\n";
            EV << "Paquete Descartado" << pk->getName() <<" en tiempo de
simulaciÃ³n"<< simTime()<< "~Fuente->"<< pk->getSrcAddr<<endl;
            delete pk;
            if(simTime()>=0.5 && simTime()<=1.5){
            Del2++;
            }
            contador2=0;
        }
    }
    if(condicion==3)
    {
        contador3++ ;
        if(contador3==1){
            EV << "Enviando con longitud de onda ="<<lamda<< "
original.\n";
            send(pk,"out",IndexOut);
            }
        else if(contador3==2){
            EV << "Enviando por FDL con retardo agregado de
"<<DelayT<<".\n";
            EV << "Retardo primer contención: "<<DelayT<< ".\n";
            sendDelayed(pk,DelayT,"out",IndexOut);
            }
        else if(contador3==3){
            EV << "Enviando por FDL con retardo agregado de
"<<DueDelayT<<".\n";
            EV << "Retardo segunda contención: "<<DueDelayT<< ".\n";
            sendDelayed(pk,DueDelayT,"out",IndexOut);
            }
        else if(contador3==4){
            EV << "Enviando por FDL con retardo agregado de
"<<TreDelayT<<".\n";
            EV << "Retardo tercera contención: "<<TreDelayT<< ".\n";
            sendDelayed(pk,TreDelayT,"out",IndexOut);
            }
```

```
        else if(contador3==5){
            EV << "Enviando por FDL con retardo agregado de
"<<CuatDelayT<<".\n";
            EV << "Retardo cuarta contención: "<<CuatDelayT<< ".\n";
            sendDelayed(pk,CuatDelayT,"out",IndexOut);
            }
        else if(contador3==6){
            EV << "FDL congestionada .\n";
            EV << "Paquete Descartado" << pk->getName() <<" en tiempo de
simulaciÃ³n"<< simTime()<< "~Fuente->"<< pk->getSrcAddr<<endl;
            delete pk;
            if(simTime()>=0.5 && simTime()<=1.5){
            Del3++;
            }
            contador3=0;
        }
    }
    if(condicion==4)
    {
        contador4++ ;
        if(contador4==1){
            EV << "Enviando con longitud de onda ="<<lamda<< "
original.\n";
            send(pk,"out",IndexOut);
            }
        else if(contador4==2){
            EV << "Enviando por FDL con retardo agregado de
"<<DelayT<<".\n";
            EV << "Retardo primer contención: "<<DelayT<< ".\n";
            sendDelayed(pk,DelayT,"out",IndexOut);
            }
        else if(contador4==3){
            EV << "Enviando por FDL con retardo agregado de
"<<DueDelayT<<".\n";
            EV << "Retardo segunda contención: "<<DueDelayT<< ".\n";
            sendDelayed(pk,DueDelayT,"out",IndexOut);
            }
        else if(contador4==4){
            EV << "Enviando por FDL con retardo agregado de
"<<TreDelayT<<".\n";
            EV << "Retardo tercera contención: "<<TreDelayT<< ".\n";
            sendDelayed(pk,TreDelayT,"out",IndexOut);
            }
        else if(contador4==5){
            EV << "Enviando por FDL con retardo agregado de
"<<CuatDelayT<<".\n";
            EV << "Retardo cuarta contención: "<<CuatDelayT<< ".\n";
            sendDelayed(pk,CuatDelayT,"out",IndexOut);
            }
        else if(contador4==6){
            EV << "FDL congestionada .\n";
            EV << "Paquete Descartado" << pk->getName() <<" en tiempo de
simulaciÃ³n"<< simTime()<< "~Fuente->"<< pk->getSrcAddr<<endl;
            delete pk;
            if(simTime()>=0.5 && simTime()<=1.5){
            Del4++;
            }
```

```
            contador4=0;
        }
    }
    if(condicion==5)
    {
        contador5++ ;
        if(contador5==1){
            EV << "Enviando con longitud de onda ="<<lamda<< "
original.\n";
            send(pk,"out",IndexOut);
            }
        else if(contador5==2){
            EV << "Enviando por FDL con retardo agregado de
"<<DelayT<<".\n";
            EV << "Retardo primer contención: "<<DelayT<< ".\n";
            sendDelayed(pk,DelayT,"out",IndexOut);
            }
        else if(contador5==3){
            EV << "Enviando por FDL con retardo agregado de
"<<DueDelayT<<".\n";
            EV << "Retardo segunda contención: "<<DueDelayT<< ".\n";
            sendDelayed(pk,DueDelayT,"out",IndexOut);
            }
        else if(contador5==4){
            EV << "Enviando por FDL con retardo agregado de
"<<TreDelayT<<".\n";
            EV << "Retardo tercera contención: "<<TreDelayT<< ".\n";
            sendDelayed(pk,TreDelayT,"out",IndexOut);
            }
        else if(contador5==5){
            EV << "Enviando por FDL con retardo agregado de
"<<CuatDelayT<<".\n";
            EV << "Retardo cuarta contención: "<<CuatDelayT<< ".\n";
            sendDelayed(pk,CuatDelayT,"out",IndexOut);
            }
        else if(contador5==6){
            EV << "FDL congestionada .\n";
            EV << "Paquete Descartado" << pk->getName() <<" en tiempo de
simulaciÃ³n"<< simTime()<< "~Fuente->"<< pk->getSrcAddr<<endl;
            delete pk;
            if(simTime()>=0.5 && simTime()<=1.5){
            Del5++;
            }
            contador5=0;
        }
    }
    //Contador de los mensajes eliminados.
    DelT=Del0+Del1+Del2+Del3+Del4+Del5;
if(ev.isGUI()){
    //Visualización de las longitudes de onda usadas y mensajes
    eliminados.
    char buf[40];
    sprintf(buf, "λ0: %ld \n λ1: %ld \n λ2: %ld \n λ3: %ld \n λ4: %ld \n
λ5: %ld \n DEL: %ld", contador0, contador1, contador2, contador3,
contador4, contador5, DelT);
    getDisplayString().setTagArg("t",0,buf);
}}
```

B.3 MODULO DE CONTENCION DEFLEXIÓN DE RUTA

```
#ifdef _MSC_VER
#pragma warning(disable:4786)
#endif

#include <vector>
#include <omnetpp.h>
#include <map>
#include "Packet_m.h"
#include <cstdio>
#include <cstdlib>
#include <stdio.h>

using namespace std;


class Deflexion : public cSimpleModule
{
    private:
    //Variables de configuración para el control de contención y toma de
    datos.
        int myAddress;
        std::vector<int> destAddresses;
        int opcDireccion,cont_lambda[23000];
        int cont=0, x, y, c, a, d, e, cruce[10];
        double long_rafag;
        double long_rafag1;
        int i, j, f, h,o,p,q,r,puerto=0, salida, out, out2;
        double tiempo_lambda_salida[10][23000],
        tiempo_lambda_llegada[10][23000];
        double tiempo_llegada[23000];
        int posicion_rafaga[10][23000];
        double tiempo_salida[23000];
        double tiempoffset;
        int tamano;
        int path;
        int num_path;
        float nuevo_offset = 0;
        //Variables para el manejo del offset.
        volatile double serviceTime;
        volatile double nuevoOffset;
        volatile double tiempo;
        simsignal_t dropSignal;
    protected:
        virtual void initialize();
        virtual void handleMessage(cMessage *msg);

};

Define_Module(Deflexion);


void Deflexion::initialize()
{
    //Inicialización de los parámetros en cero y asignación de los
    valores para el manejo del offset.
```

```
    myAddress = par("address");
    opcDireccion = 0;
        for (x=1;x<7;x++){
            for(y=1;y<23000;y++){
                tiempo_lambda_llegada[x][1] = 0.02;
                tiempo_lambda_salida[x][1]= 0.02;
            }
        }
        for (o=1; o<7; o++){
            for (p=1; p<23000; p++){
        posicion_rafaga[o][p]=0;
            }
        }
        cont_lambda[1]=0;
        cruce[1]=0;
        cruce[2]=0;
        cruce[3]=0;
        cruce[4]=0;
        cruce[5]=0;
        cruce[6]=0;
        cruce[7]=0;
        salida=out;
    const char *destAddressesPar = par("destAddresses");
        cStringTokenizer tokenizer(destAddressesPar);
        const char *token;
        while ((token = tokenizer.nextToken())!=NULL)
            destAddresses.push_back(atoi(token));
}


void Deflexion::handleMessage(cMessage *msg)
{
    //Manejo del mensaje, obtención y asignación de parametros necesarios
    para realizar el control de contención.
    volatile double tiempo_envio = exponential(0.2);
    volatile double tiempo_envio_sobre10 = tiempo_envio/10;
    tiempo = par("serviceTime");
    Packet *pk = check_and_cast<Packet *>(msg);
        int destAddr = pk->getDestAddr();
        tiempoffset = pk->getOffset();
        tamano = pk->getByteLength();
        long_rafag1 = pk->getLongRafaga();
        long_rafag = long_rafag1/10;
        path = pk->getRuta();
        num_path = pk->getNueva_ruta();
        int nuevo_puerto_aleatorio;
        EV << "longitud : "<<long_rafag<< " .\n\n\n";
        ev <<pk->getHopCount()<<endl;
        opcDireccion = destAddr;
        cont++;
        puerto=0;
        ev<<"tiempo llegada ráfaga "<<cont<<" : "<<tiempoffset<<"s.\n";
        ev<<"tiempo salida ráfaga "<<cont<<" : "<<tiempoffset +
        long_rafag<<" s.\n\n";
        ev<<"tiempo envio "<<tiempo_envio<<" s.\n\n";
        tiempo_llegada[cont] = tiempoffset;
        tiempo_salida[cont] = long_rafag + tiempo_llegada[cont];
```

```
            for(d=1; d<=6; d++){
               for(e=1; e<cont; e++){


tiempo_lambda_llegada[d][e]=tiempo_lambda_llegada[d][e]-
tiempo_envio_sobre10;
                          tiempo_lambda_salida[d][e] =
tiempo_lambda_salida[d][e]- tiempo_envio_sobre10;
               }
            }
             for(i=1; i<7; i++){
                 for(j=1; j<cont; j++){


if((((tiempo_llegada[cont]>=tiempo_lambda_llegada[i][j])&&(tiempo_llegada
[cont]<=tiempo_lambda_salida[i][j]))||((tiempo_salida[cont] >
tiempo_lambda_llegada[i][j])&&(tiempo_salida[cont] <
tiempo_lambda_salida[i][j])))
                {
                 cruce[i]++;
                 puerto++;
                 j=cont;
                        }
                      }
                   }
             for(f=1; f<=6; f++){
                    if(cruce[f]==0){
                    tiempo_lambda_llegada[f][cont]= tiempoffset;
                    tiempo_lambda_salida[f][cont]= tiempoffset +
                    long_rafag;
                    cont_lambda[f]++;
                    posicion_rafaga[f][cont]++;
             EV << "tiempo ráfaga "<<cont<<" :
             "<<tiempo_lambda_llegada[f][cont]<<" en lambda : " <<f<<"
             .\n\n";
                    f=7;

                     }
                   }
             for(h=1; h<7; h++){
             cruce[h]=0;
             }
          if (puerto == 6){
          EV << "No hay lambda disponible para ráfaga "<<cont<<" .\n";
          EV << "Se envía ráfaga por siguiente puerto .\n";
          EV << "path "<<path<<".\n";
          if(num_path!=0){
              do{
               nuevo_puerto_aleatorio = intuniform(0,num_path);
              }while(nuevo_puerto_aleatorio==path);
          EV << "nuevo puerto********** " << nuevo_puerto_aleatorio
<<endl;
          send(pk, "out", nuevo_puerto_aleatorio);
              }
          else{
```

```
                EV << "Received " << msg << " Pero No hay recursos:
discarding\n";
                emit(dropSignal, (long)check_and_cast<cPacket *>(msg)-
>getByteLength());
                delete msg;
          }
          }
          else{
          EV << "path "<<path<<".\n";
          send(pk, "out", path); }}
```