ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Trabajo Fin de Máster

Diseño y evaluación de prestaciones de una red óptica OBS multinodo mediante herramienta de simulación

AUTOR: Alejandro Ortuño Manzanera
DIRECTOR: Pablo Pavón Mariño

Julio/ 09

Autor	Alejandro Ortuño Manzanera
E-mail del Autor	aom@alu.upct.es
Director(es)	Pablo Pavón Mariño
E-mail del Director	pablo.pavon@upct.es
Codirector(es)	
Título del PFC	Diseño y evaluación de prestaciones de una red óptica OBS multinodo mediante herramienta de simulación
Descriptores	

Resumen

El objetivo del trabajo fin de máster es la utilización del simulador oPASS existente para conmutación óptica de paquetes, para la simulación de topologías genéricas tanto mononodo como multinodo. Este trabajo se centrarà en evaluar las prestaciones de una red OBS (*Optical Burst Switching*) en la cual se .implementan distintos *Schedulers.*

Titulación	Máster en Tecnologías de la Información y las Comunicaciones
Intensificación	
Departamento	Tecnologías de la Información y las Comunicaciones
Fecha de Presentación	Julio– 2009

Capítulo 1
Introducción

1.1 Redes WDM

La rápida evolución de las redes de telecomunicaciones ha ido acompañada siempre por un aumento cada vez mayor de la demanda de los usuarios de nuevas aplicaciones y también por el continuo avance de las tecnologías disponibles. En los últimos años, hemos sido testigos del éxito y el crecimiento explosivo de Internet.

Para suplir la demanda de ancho de banda, se produjo una revolución con la introducción de la fibra óptica en las redes de telecomunicaciones. Como núcleo de esta revolución, las fibras ópticas han demostrado ser un excelente medio físico de transmisión para suministrar un elevado ancho de banda. Teóricamente, una simple fibra óptica monomodo, tiene un ancho de banda de cerca de 50 terabits por segundo (*Tbps*), cuatro órdenes de magnitud por encima de la actual velocidad de procesamiento electrónico alcanzable, que es de unos pocos gigabits por segundo (*Gbps*). Las fibras ópticas también tienen otras características importantes como su baja atenuación, su baja tasa de error de bit, baja distorsión de la señal, bajos requerimientos de potencia…Sin embargo, debido al límite de la velocidad de procesamiento electrónica, es poco probable que todo el ancho de banda de una fibra óptica pueda ser explotado usando un simple canal óptico de alta capacidad o una simple longitud de onda. Se requiere por tanto un sistema de multiplexación adecuado, que combine la transmisión de múltiples fuentes de tráfico, en el orden de los *Gbps*, en un agregado en el orden de (potencialmente) *Tbps.*

La emergencia de la tecnología de multiplexado por división en frecuencia (WDM) ha suministrado una solución para realizar este reto. Con la tecnología WDM, múltiples señales ópticas pueden ser transmitidas de forma simultánea e independientemente en diferentes canales ópticos sobre una simple fibra óptica, cada una a una tasa de unos pocos gigabíts por segundo lo que significa incrementar el ancho de banda empleado de una fibra óptica. WDM tiene también otras ventajas como su reducido coste de procesamiento electrónico y la transparencia de los datos. Como resultado WDM se ha convertido en la tecnología elegida para cumplir con la enorme demanda de ancho de banda existente [1] [7].

Inicialmente, todas las funciones de *routing* y *switching* se realizaban electrónicamente en cada nodo de la red. Esto obligaba a las señales ópticas a pasar a través de un conversor óptico-eléctrico (O/E) y eléctrico-óptico (E/O) en cada nodo intermedio mientras la señal se propagaba por un camino *end-to-end* desde un nodo hacia otro. En consecuencia, un nodo de la red no era capaz de procesar todo el tráfico introducido en sus canales de entrada, causando el problema del “cuello de botella electrónico”.

Para intentar evitar los problemas asociados al cuello de botella electrónico, y gracias a la mejora constante de la tecnología de dispositivos ópticos, se han planteado una serie de técnicas que se basan en el tratamiento óptico del tráfico que debe atravesar un nodo de

conmutación de la red de transporte (es decir, tráfico no originado ni destinado al mismo). Este tipo de alternativas se engloban dentro de lo que se denomina conmutación O-O-O. Desde la década de los 90, el estudio en este campo ha seguido tres estrategias distintas: Conmutación Óptica de Longitudes de Onda (*Wavelength Routing,* WR), Conmutación Óptica de Ráfagas (*Optical Burst Switching,* OBS) y Conmutación Óptica de Paquetes (*Optical Packet Switching,* OPS). En este proyecto se va a emplear la Conmutación Óptica de Paquetes [2].

1.2 Redes de Conmutación Óptica de Ráfagas

1.2.1 Modelo de red

Desde sus orígenes hace una década, el paradigma *Optical Burst Switching* (OBS) [1] [7] ha atraído un enorme interés en el campo de la investigación. OBS es una técnica que se encuentra en la mitad del espectro entre la conmutación de circuitos (OCS) y la conmutación de paquetes (OPS). Los sucesivos refinamientos y estudios han sido conducidos en numerosos campos que crecieron en relación con la tecnología OBS. Para nombrar algunos: técnicas de ensamblado de ráfagas y su efecto en el rendimiento del sistema y en los patrones de tráfico resultantes, los protocolos de reserva y la variación del offset inicial y del offset salto a salto, o los algoritmos de *scheduling* para los nodos de conmutación OBS.

En las redes *Optical Burst Switching* (OBS), el tráfico en formato electrónico (por ejemplo paquetes IP) se ensambla en ráfagas ópticas de longitud variable, las cuales se inyectan en la red OBS a través de los *edge nodes* y de forma transparente, se enrrutan a través de los nodos *traversing* cruzando la red OBS hasta el *edge node* destino de la ráfaga. Los *edge node* son los encargados del ensamblado de las ráfagas y de la inyección de estas a la red OBS. Los nodos *traversing* que hemos tenido en cuenta en este trabajo, no inyectan tráfico nuevo a la red sino que únicamente conmutan de forma transparente las ráfagas de entrada en formato óptico a su correspondiente fibra óptica de salida objetivo.

Antes de enviar la ráfaga correspondiente, se envía un paquete de control (*Burst Header Packet*) por cada ráfaga que se desea enviar. Este paquete de control se envía en una longitud de onda a parte de las de datos y precede a la ráfaga en un determinado tiempo de offset. Este offset es el intervalo de tiempo entre la transmisión por el *edge node* del primer bit del paquete de control y la transmisión del primer bit del *payload.* El paquete de control lleva entre otra información el tiempo de offset hasta el siguiente salto, la longitud de la ráfaga y la etiqueta MPLS. En cada nodo intermedio a lo largo del camino entre ambos *edge nodes*, el paquete de control reserva los recursos necesarios (por ejemplo ancho de banda en el canal de salida deseado) para la siguiente ráfaga que será desensamblada en el *edge node egress*.

En redes OBS predomina un protocolo de reserva que se denomina *just-enough-time* (JET) [3]. En JET, un paquete de control reserva una longitud de onda de salida por un determinado periodo de tiempo igual a la longitud de la ráfaga comenzando cuando en el momento es que se espera que la ráfaga llegue al *switch*. Este tiempo se determina a partir del valor de offset y del tiempo de procesamiento del paquete de control. Si la reserva tiene éxito, el paquete de control ajusta el valor de offset para el siguiente salto y lo envía al siguiente nodo en el camino; de lo contrario, la ráfaga es bloqueada y se descarta. En los últimos años se han estudiado distintas técnicas como son las *deflection routing* y la segmentación de ráfagas donde se estudia reducir la pérdida de datos en las redes OBS.

Dado el hecho de que las redes OBS usan protocolos de reserva en un solo sentido como por ejemplo el JET [3] y que una ráfaga no puede ser almacenada en ningún nodo intermedio debido a la falta de memoria óptica RAM (una FDL, si está disponible, solo puede

suministrar un limitado retardo y resolución de contenciones), la pérdida de ráfagas puede ser un serio problema para las redes OBS, especialmente cuando el sistema está mal sincronizado y las ráfagas de forma repetida llegan en el peor de los casos. Ya que la mayoría de los sistemas comerciales requieren un rendimiento en el peor caso aceptable, es importante entender el rendimiento en el peor de los casos y también el diseño de los *schedulers* con un peor caso optimizado.

La unidad de control procesa los BHPs que llegan, tomando las correspondientes decisiones de *scheduling*. Para cada ráfaga se determina la fibra óptica de salida después de procesa la cabecera. No se van a considerar en los trabajos propuestos ningún tipo de técnicas de *Deflection routing*. El algoritmo de *Scheduling* es el encargado de asignar una FDL libre de bloqueo y una longitud de onda para conmutar el *payload* de la ráfaga desde la fibra óptica de entrada a la fibra óptica de salida. Debido a la longitud variable de las ráfagas, este proceso puede implicar la creación de huecos de tiempo sin usar entre dos ráfagas consecutivas de una misma longitud de onda. Estos huecos son denominados *voids* y provocan un uso ineficiente de los recursos. Los algoritmos de *scheduling* encargados de asignar ráfagas de menor tamaño para rellenar los huecos en las longitudes de onda de salida se denominan algoritmos de *void filling.* Un resumen de los más importantes algoritmos de *scheduling* se propone en el apartado 1.3.3.

La Fig.1 muestra el esquema de una red OBS multinodo. En OBS, las longitudes de onda son compartidas entre las diferentes conexiones (similar a como ocurre en OPS). En los *edge nodes*, los paquetes que vienen de otras redes cliente (por ejemplo IP o redes ATM), son agregados dentro de ráfagas ópticas las cuales son transmitidas y conmutadas por los nodos *traversing* o *core nodes* a través de la red hasta su destino.

Cada ráfaga tiene asociado su paquete de control. El paquete de control y su *payload* son transmitidos de forma separada en longitudes de onda separadas. El paquete de control se envía al *core node* o nodo *traversing* con un tiempo de offset previo al envío de su *payload* asociado. De esta manera, la unidad de control del *core node* tiene tiempo suficiente para procesar la información de control asociada al BHP y para ajustar la matriz de conmutación óptica del nodo a la llegada del *payload*.

Una vez que el *payload* se inyecta a través del *edge node* a la red OBS, ésta permanece todo el camino hasta el *edge node* destino en el dominio eléctrico.

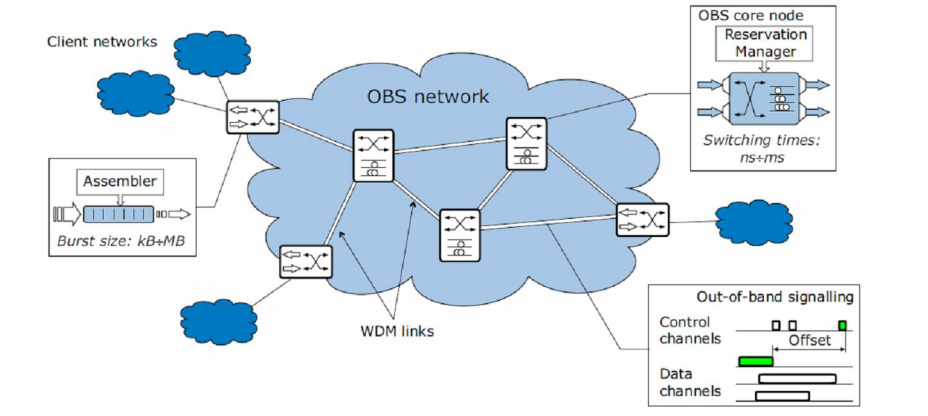

Fig.1: Esquema de red OBS

1.2.2 Protocolos de reserva

Existen varios protocolos de reserva que se emplean en redes de conmutación óptica de ráfagas. Estos protocolos de señalización que usan un intervalo temporal, offset, entre un paquete de control y la transmisión de la ráfaga debe usar algunos mecanismos para estimar este valor. Una estimación incorrecta de este valor podría suponer una pérdida de datos. Vamos a ver cada uno de ellos:

- *Tell And Go* (TAG): Con este protocolo, la fuente transmite la ráfaga inmediatamente después del paquete de control BHP (*Burst Header Packet*), tan pronto como ésta recibe el mensaje desde su capa de aplicación (o de alguna capa superior). Una copia del mensaje se conserva en la fuente hasta que ésta sepa que la ráfaga se ha recibido en el destino de forma correcta. El receptor envía un ACK de vuelta a la correspondiente fuente. Si un *Switch* tiene que descartar la ráfaga, retransmitirá la cabecera de la ráfaga al receptor para indicarle que la ráfaga asociada ha sido perdida. Por lo tanto, si la ráfaga se pierde a lo largo del camino, el receptor lo conocerá y entonces enviará un NACK a la fuente. Posteriormente, la fuente transmite la misma ráfaga de nuevo un tiempo después. Este protocolo es práctico únicamente cuando el tiempo de configuración del *switch* y el tiempo de procesamiento de un paquete de control son muy cortos.

- *Tell And Wait* (TAW): La ráfaga se transmite por el *edge node* solo si se ha establecido un camino virtual a través de la red hasta el *edge node* destino. El camino virtual se crea mediante la concatenación de longitudes de onda reservadas enlace a enlace, durante la fase de establecimiento previa a la transmisión de la ráfaga. En este sentido, esta técnica corresponde a la tradicional conmutación de circuitos. Este protocolo requiere que el offset sea al menos igual al tiempo requerido para recibir un reconocimiento del destino. Cuando un *edge node* tiene una nueva ráfaga para transmitir, éste envía, en una longitud de onda de control, un mensaje de control con destino al *edge node* destino de la ráfaga. Este mensaje tiene como objetivo reservar una longitud de onda de datos en cada cada enlace a través del camino entre el *edge node ingress* y el *edge node egress*. Cuando el mensaje de establecimiento se recibe por un nodo *traversing*, la unidad de control reserva una longitud de onda de datos libre por lo que ésta longitud de onda estará dedicada a la ráfaga hasta que se reciba un mensaje *release*. Una vez que la reserva se ha realizado, la unidad de control configura la matriz de conmutación del nodo determinando la correspondiente conversión de longitud de onda para solucionar la posible contención a la salida con otras ráfagas.

- *Just-Enough-Time* (JET): En este protocolo, se selecciona el offset para que sea lo suficientemente elevado como para tener en cuenta el tiempo de procesamiento del paquete de control en cada nodo intermedio. El paquete de control se envía desde el *edge node* previo a la correspondiente ráfaga un determinado tiempo de offset. Este paquete de control reserva una longitud de onda de salida por un periodo de tiempo igual al tamaño de la ráfaga a partir del tiempo esperado de llegada del *payload*. Cuando este offset expira, la ráfaga abandona el *edge node* y cruza los nodos configurados para su recepción. Un problema existente en el protocolo JET es determinar el número de nodos intermedios (*Hops*) entre la fuente y el destino. Sin embargo, el número de saltos en un camino no es un dato disponible generalmente. El protocolo JET presenta el mejor uso del ancho de banda a la hora de transmitir los recursos [3].

- *Only Destination Delay* (ODD): Dados los siguientes avances en implementación hardware de los protocolos de comunicaciones, parece razonable asumir que el tiempo de procesamiento en los nodos intermedios será muy pequeño para la mayoría de las funciones comunes de los protocolos de señalización. En este caso, se pueden emplear fibras de retardo de un tamaño razonable en los nodos intermedios para retardar cada ráfaga entrante por una cantidad de tiempo igual al tiempo de procesamiento. La idea es que no exista un establecimiento de offset en los *edge nodes* sino que por el contrario, sean los nodos *traversing* los que introduzcan un offset local por medio de una FDL colocada en los puertos de entrada del nodo (ver Fig. 3). Los paquetes de control y *payload* son enviados juntos hacia la red por el *edge node*. Cuando ambos paquetes alcanzan un nodo *traversing*, el paquete de control va directamente a la unidad de control mientras que la ráfaga se retarda en la FDL del puerto de entrada. Durante este tiempo, el paquete de control se procesa y la unidad de control puede establecer los elementos de conmutación ópticos correspondientes. Una regla importante es que el paquete de control, después de su procesamiento, espera a su ráfaga en la memoria de la unidad de control hasta que el *payload* salga de la FDL de entrada y ambos sean enviados de forma conjunta al siguiente nodo. En caso de que no hayan recursos disponibles, tanto el paquete de control como el *payload* son descartados.

1.3 Modelo del nodo

1.3.1 Nodo edge

La figura Fig. 2 muestra la estructura de un *edge node*:

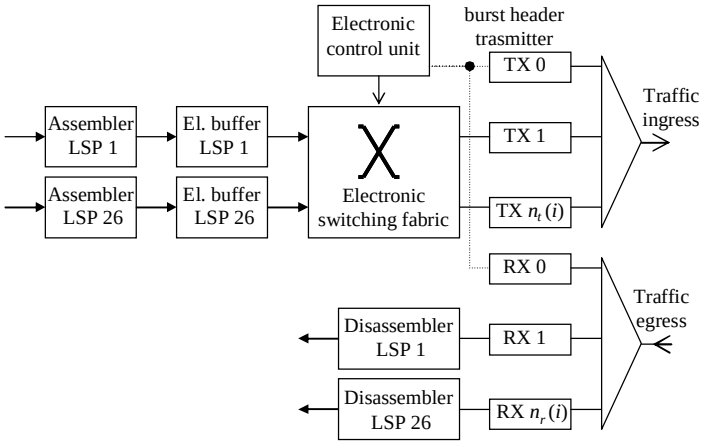

Fig. 2: Esquema de un nodo edge

Las velocidades de conmutación son un significante cuello de botella en las redes *all-optical*. En el estado del arte actual, los *switches* comercialmente disponibles tienen velocidades de conmutación de apenas menos de 1ms. Una solución empleada en OBS es la de agregar o ensamblar paquetes en ráfagas en el *edge node* para que los tiempos de conmutación y las ineficientes bandas de guarda sean despreciables con respecto a los tamaños de ráfaga. Por esto de la importancia del ensamblado. El tráfico eléctrico de entrada

de cada LSP se supone que está compuesto por paquetes IP, los cuales son ensamblados en una determinada ráfaga correspondiente módulo *assembler* [10]. Los dos principales algoritmos de ensamblado son: los basados en umbral y los basados en tiempo aunque también vamos a ver algunas modificaciones a ambos:

- Algoritmos de ensamblado basados en umbral

  Los algoritmos de ensamblado basados en umbral requieren de un sencillo y configurable parámetro, Bi_{th} [bytes] para cada cola *i*. Este parámetro corresponde aproximadamente a la longitud, en bytes, de cada ráfaga enviada desde esa cola. Si un paquete que llega provoca que el tamaño agregado exceda Bi_{th} para la cola *i*, la ráfaga es planificada para ser enviada y los paquetes que componen esta ráfaga son inmediatamente eliminados de la cola *i*. Este algoritmo no ofrece garantías para el retardo. Bajo una carga pequeña, este algoritmo puede llevar a que se necesita esperar un largo periodo de tiempo en la cola *i* hasta que se alcance el umbral Bi_{th}. Sin embargo, bajo una alta carga ofrecida, el umbral se alcanza rápidamente minimizando el retardo. En resumen, este algoritmo produce ráfagas de longitud fija con un tiempo aleatorio entre ráfagas de una misma cola. Normalmente el valor del umbral es constante para todas las colas de entrada.

- Algoritmos de ensamblado basados en temporizador

  De una forma similar a los algoritmos basados en umbral, los algoritmos basados en temporizador también requieren de un parámetro sencillo y configurable. El parámetro para cada cola es un periodo de tiempo, T_I, que corresponde aproximadamente al tiempo entre dos ráfagas consecutivas de una misma cola. En el ensamblado basado en temporizador, existe un temporizador para cada cola que se inicializa al comenzar el sistema e inmediatamente después de que la ráfaga previa para esa cola sea planificada para ser enviada o en otras versiones, inmediatamente después de que el primer paquete llegue después de que la cola se haya eliminado. En el momento en que expira el temporizador (después de un tiempo T_I) el *assembler* genera una ráfaga que contiene todos los paquetes que había en el buffer en ese momento. Con respecto al retardo, el rendimiento de este algoritmo es lo contrario al algoritmo basado en umbral descrito antes. Bajo una baja carga ofrecida, el ensamblado basado en temporizador garantiza un mínimo retardo fijo. Sin embargo, bajo una alta carga de entrada, puede generar ráfagas que son bastante grandes, tal vez incrementando de forma innecesaria el retardo. En resumen, este algoritmo produce ráfagas de tamaño aleatorio con un tiempo fijo entre dos ráfagas consecutivas las cuales pertenecen a una misma cola. Normalmente el valor del temporizador es constante para todas las colas de entrada.

- Algoritmos de ensamblado híbridos

  Los algoritmos basados en temporizador y en umbral se pueden emplear de forma conjunta para adquirir las ventajas de ambos algoritmos. En este sistema híbrido, las ráfagas se envían cuando alguna de las restricciones se alcanza. Por ejemplo, para periodos de baja carga de entrada, el temporizador expirará primero, y como resultado tendremos un espaciado entre ráfagas determinístico pero un tamaño de ráfaga aleatorio (pequeño). Sin embargo, para periodos de alta carga de entrada, el umbral del tamaño de ráfaga se va a alcanzar primero, resultando en un espaciado aleatorio y en un tamaño de ráfaga constante e igual al tamaño máximo de ráfaga.

- Algoritmos de ensamblado *Off-timer*

  Vamos a introducir una solución con poca complejidad que alivia los problemas de sincronización temporal. En los *assemblers* basados en temporizador, un *timer* comienza en la inicialización del sistema y normalmente inmediatamente después de que la ráfaga previa se envíe. Cuando expira el *timer*, el *assembler* genera una ráfaga que contiene todos los paquetes que hay en el buffer en ese punto. Este problema de sincronización y la posterior alta probabilidad de bloqueo de ráfaga se soluciona aleatorizando el tiempo de offset de la ráfaga. Aleatorizar los tiempos de offset de las ráfagas cambia la prioridad de las ráfagas. Es más, incrementar los tiempos de offset de las ráfagas se ha propuesto como un método efectivo de priorización en diferentes clases de tráfico. En lugar de resetear el *timer* cuando llegua un nuevo paquete o inmediatamente después de eliminar la ráfaga de la cola, nosotros esperamos hasta que la ráfaga completa ha sido enviada en el correspondiente enlace de salida, y después se resetea el *timer*. Este método se denomina *Off Timer Burst Assembly* (OTBA).

Suponemos que el proceso de agregación de datagramas para ensamblar paquetes ópticos ya se ha realizado previamente por lo que estamos recibiendo ráfagas ópticas en formato eléctrico. El tamaño en bytes de cada ráfaga dependerá de la longitud temporal de ésta, y de la velocidad binaria a la que inyecta la carga de datos (ejemplo: 40 Gbps, 1 µs: 5000 bytes por ráfaga). Además, este tráfico agregado ya viene con un camino establecido (entre nodo origen y destino).

En nuestro caso empleamos un *assembler* de tipo híbrido simulado cuyo tiempo entre dos ráfagas consecutivas de un mismo LSP sigue una distribución exponencial. Esta media se calcula para que coincida con el valor de carga deseado. El tamaño de ráfaga sigue una distribución uniforme entre el tamaño máximo de ráfaga (100.03 µs) y el tamaño mínimo de ráfaga (10.03 µs). El módulo opuesto al *assembler* es el *disassembler*. Este módulo es el encargado de recibir las ráfagas en formato eléctrico y de obtener los paquetes IP que componen esa ráfaga.

Los paquetes que son ensamblados por el módulo *assembler* se inyectan en colas electrónicas a la espera de ser inyectadas en la res. De esto se encarga el módulo *ingress*. En estas colas se produce la clasificación del tráfico en función de su clase (QoS) y de la fibra óptica de salida que se emplee para enviar el paquete al siguiente nodo. Previamente al ingreso en cola, se realiza un marcado del tráfico en función de que la cantidad de tráfico que se esté intentando inyectar en la red para una determinada clase y una fibra óptica de salida según exceda un determinado límite o no.

El siguiente módulo es la matriz de conmutación electrónica que se encarga de elegir los paquetes que salen en cada momento del *ingress* para ser inyectados a través de un conversor E/O a la red OBS. Para planificar el tiempo de transmisión y el canal de transmisión de una ráfaga, la matriz de conmutación electrónica ejecuta una variación del algoritmo LAUC-VF: selecciona el canal de transmisión que es capaz de acomodar el *payload* más pronto, pero no antes de un offset inicial. El sistema debe garantizar que para cada transmisor, la cola del *payload* se separa un mínimo de tiempo de la cabeza de la siguiente. Este mínimo *interburst-gap* se requiere entre dos ráfagas consecutivas para que los dispositivos de conmutación óptica se reconfiguren en el siguiente nodo. Vamos a asumir que este tiempo de reconfiguración óptico es igual a 0.03 µs. Una vez que se ha planificado la salida del *payload* del *edge node*, el *scheduler* genera un paquete de control (BHP),que inyecta en la red, previo al envío del *payload*.

1.3.2 Nodo de interconexión

La Fig. 3 ilustra la arquitectura genérica de un nodo de conmutación OBS con *N* fibras ópticas de entrada y de salida, una longitud de onda de control (λ_0) y *n* longitudes de onda de datos ($\lambda_1 \ldots \lambda_n$) por fibra. La matriz de conmutación óptica (*OSF*) de forma transparente conmuta las ráfagas ópticas desde los puertos de entrada a los puertos de salida. En este trabajo vamos a considerar *OSFs* que son capaces de emular el *buffering* a la salida, con una conversión entre longitudes de onda que es completa (cualquier longitud de onda de entrada puede ser convertida a cualquier longitud de onda disponible a la salida), y *D* líneas de retardo (FDLs) de duración *d=0,G,…,(D-1)G* donde *G* denota la granularidad de la FDL. La emulación de almacenamiento a la salida significa que la matriz OSF no añade ningún bloqueo interno al proceso de asignación de longitud de onda a la salida.

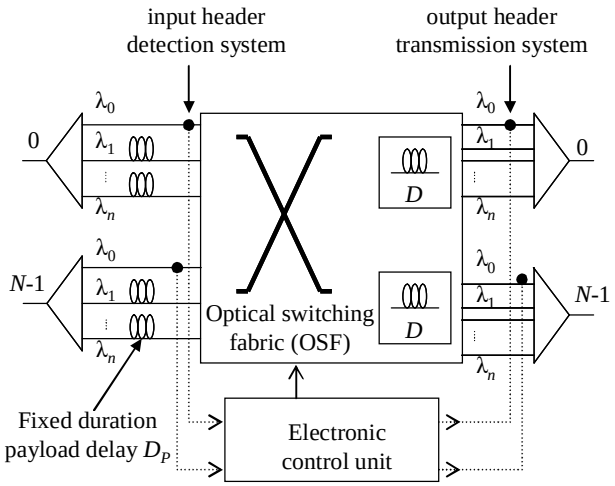

Fig. 3. Esquema de un nodo OBS

Como se puede observar, en un nodo *traversing* se ha eliminado toda la parte de inyección de tráfico que se realizaba en los *edge nodes*. No vamos a tener nodos *traversing* que tengan módulos de ensamblado de tráfico ni de conformación de tráfico. Esto es debido a que en las simulaciones realizadas, separamos la generación de tráfico local en un *edge node* adjunto a cada nodo *traversing*. Cabe destacar que aunque la inyección de tráfico la realizamos de esta manera, el simulador tiene capacidad de hacer que los nodos *traversing* puedan tener puertos *ingress* para la inyección de tráfico local (generado en el propio nodo *traversing*) aunque quedará fuera de este trabajo.

La unidad de control procesa los BHPs que llegan, tomando las correspondientes decisiones de *scheduling*. Para cada ráfaga se determina la fibra óptica de salida después de procesar la cabecera. No se van a considerar en los trabajos propuestos ningún tipo de técnicas de *deflection routing*. El algoritmo de *Scheduling* es el encargado de asignar una FDL libre de bloqueo y una longitud de onda para conmutar el *payload* de la ráfaga desde la fibra óptica de entrada a la fibra óptica de salida. Debido a la longitud variable de las ráfagas, este proceso puede implicar la creación de huecos de tiempo sin usar entre dos ráfagas consecutivas de una misma longitud de onda. Estos huecos son denominados *voids* y provocan un uso ineficiente de los recursos. Los algoritmos de *scheduling* encargados de asignar ráfagas de menor tamaño para rellenar los huecos en las longitudes de onda de salida se denominan algoritmos de *void filling.* Un resumen de los más importantes algoritmos de *scheduling* se propone en el apartado 1.3.3.

Vamos a considerar que el tiempo requerido para el procesamiento de las cabeceras T_{proc} por la unidad de control electrónica es constante en cada nodo y ráfaga. Los canales de datos en cada fibra de entrada pasan a través de una FDL de longitud D_p (como se puede ver en la figura 2). Por tanto, un *payload b* llega a la matriz de conmutación un determinado tiempo de offset más D_p µs después de la llegada de la cabecera de la ráfaga. Siguiendo este sencillo modelo, la reducción en el tiempo de offset de la ráfaga después de atravesar un nodo viene dada por la diferencia entre el tiempo de procesamiento de la ráfaga y el valor temporal de la FDL a la entrada.

De esta forma, incrementar el retardo a la entrada D_p que sufre el *payload* en un nodo, implica que la disminución del offset que este nodo provoca se reduce, y consecuentemente el offset inicial de los LSPs que lo atraviesan se puede reducir. En el límite, si el valor del retardo D_p coincide con el tiempo de procesamiento, entonces el offset es constante salto a salto, y el offset inicial se hace cero. Si el valor del retardo *D* es mayor que el tiempo de procesamiento, el offset se incrementa salto a salto.

1.3.3 Schedulers para redes OBS

En la literatura se han propuesto numerosos algoritmos de *scheduling*, donde los objetivos principales son los de suministrar una utilización de recursos eficiente y minimizar la pérdida de ráfagas.

Los algoritmos de *scheduling* de los canales de datos se pueden clasificar dentro de dos categorías: sin y con rellenado de huecos (*void filling*). Las capacidades de rellenado de huecos consiguen una mejora en términos de rendimiento alcanzado por el *scheduler*. Dentro de los algoritmos sin rellenado de huecos, vamos a presentar al algoritmo LAUC. Después vamos a presentar varias extensiones de este algoritmo para realizar rellenado de huecos.

- LAUC (*Latest Available Unscheduled Channel*): Es muy similiar al algoritmo Horizon [6]. La idea básica del algoritmo LAUC es la de minimizar los huecos seleccionando el canal de datos disponible más reciente para cada ráfaga de entrada. Dado el tiempo de llegada *t* de una ráfaga de datos de duración *L* ,el *scheduler* primero encuentra el canal de datos de salida que no tiene planificado nada aún (longitud de onda a la salida que no tiene ninguna ráfaga planificada después del instante de llegada de la ráfaga de entrada). Si existe al menos un canal que presente estas características, el *scheduler* selecciona el último canal disponible, por ejemplo, el canal que tiene un menor hueco entre *t* y el final de la ráfaga de datos justo antes de t, para llevar la ráfaga de datos que llega. El tiempo sin planificar del canal seleccionado (por ejemplo el tiempo futuro disponible) es actualizado a $t+L$. Por ejemplo en la Fig.4 , los canales 2 y 3 son canales sin planificar en el tiempo *t*, se selecciona el canal D_2 debido a que $t_2-t<t_3-t$. Si todos los canales se encuentran planificados en un tiempo *t* la ráfaga de datos se tiene que retardar por un tiempo múltiplo de la unidad FDL, digamos *i* unidades hasta que al menos se encuentre un canal de datos sin planificar. Si $1 \le i \le B$ (donde B es el máximo número de unidades FDL), el *scheduler* seleccionará el último canal disponible para transmitir la ráfaga de datos y actualizará el tiempo de canal sin planificar a un valor de $t+i \cdot D+L$. *Si* $i>B$, la ráfaga sencillamente se descarta. En la Fig.5 todos los canales de datos se encuentran planificados en un tiempo *t* pero el canal 1 y 3 no se encuentran planificados en un tiempo $t+D$. Esto hace que la ráfaga de datos que llega se retarde por un tiempo igual a una unidad de FDL y el canal D_3 se selecciona para llevar la ráfaga de datos. La simplicidad y facilidad de implementación son las ventajas

principales del algoritmo LAUC ya que el *scheduler* solo necesita recordar un valor (el tiempo sin planificar) para cada canal de datos. La simplicidad es muy importante en entornos con extremadamente altas velocidades. La desventaja del algoritmo LAUC es el ineficiente uso de los canales de datos ya que los huecos entre las ráfagas de datos no se emplean. La capacidad de almacenamiento del buffer FDL está determinada no solo por el número de FDLs sino también por el tamaño de cada FDL. Claramente, cuanto mayor sea la unidad FDL, mayor es el hueco introducido el cual hace al algoritmo LAUC menos eficiente, provocando una mayor probabilidad de pérdida de ráfaga. Para solucionar este problema, se emplean los algoritmos que implementan *void filling*.

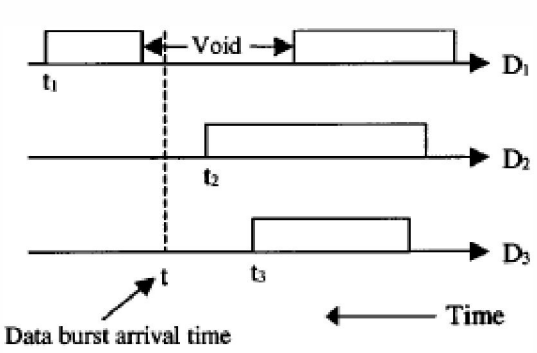

Fig. 4.Scheduler LAUC

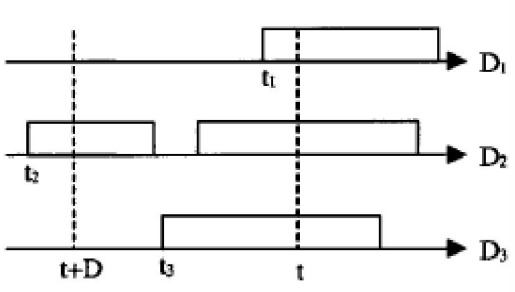

Fig. 5. Scheduler LAUC

- LAUC-VF (*Latest Available Unscheduled Channel-Void Filling*) [12]: En este caso, el hueco entre dos ráfagas de un canal de datos (ver canal D_1 de la Fig.4) es capacidad sin usar. Este algoritmo es similar al LAUC con la excepción de que los huecos se pueden rellenar por nuevas ráfagas de datos que lleguen al *scheduler*. La idea básica del algoritmo LAUC-VF es la de minimizar los huecos seleccionando el último canal de datos disponible sin usar para cada ráfaga de datos que llegue. Los canales de datos sin planificar son solo un caso especial de los canales de datos sin usar. Dado el tiempo de llegada *t* de una ráfaga de datos con duración *L*, el *scheduler* primero el canal de datos a la salida que se encuentre disponible por un periodo de tiempo de (*t, t+L*). Si hay al menos un canal de datos, el *scheduler* selecciona el último canal de datos disponible, por ejemploe, el canal que tiene el menor hueco entre *t* y el final de la última ráfaga justo antes de *t*. La Fig.5 muestra una ilustración del algoritmo LAUC-VF. En ella los canales D_1, D_2 y D_5 son canales de datos sin usar elegibles en un tiempo *t*. Sin embargo, los canales D_3 y D_4 no son elegibles en *t* debido a que el hueco es demasiado pequeño en D_3 para la ráfaga de datos y D_4 está ocupada en tiempo *t*. El canal de datos D_2 se elige para llevar la ráfaga ya que $t-t_2 < t-t_1 < t-t_3$. Si todos los canales

de datos no se pueden elegir en un tiempo *t*, el *scheduler* intentará encontrar un canal de datos que sea elegible en un tiempo *t+D* (por ejemplo elegible para un periodo de tiempo de [*t+D, t+D+L*]) y así sucesivamente. Si no se encuentran canales de datos que sean elegibles hasta un tiempo de *t+B·D* la ráfaga de datos de entrada y su correspondiente BHP son descartadas. Mencionar que *B·D* constituyen el mayor tiempo que una ráfaga puede ser almacenada (retardada).

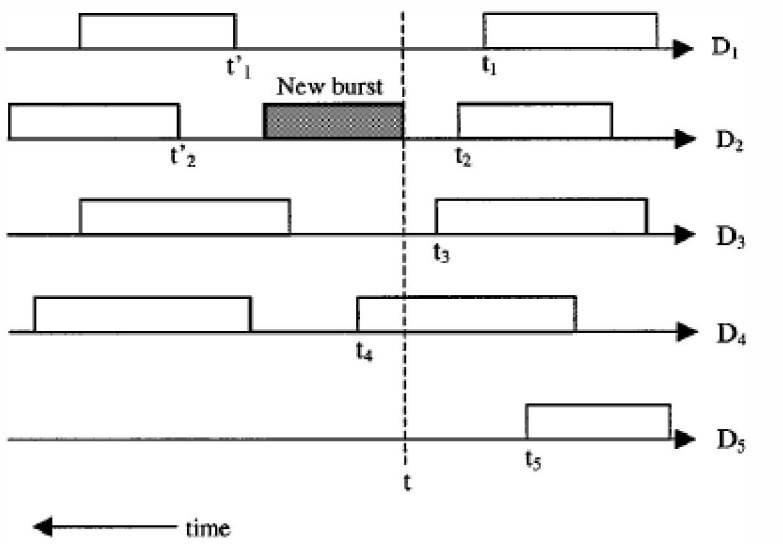

Fig.6: Scheduler LAUC-VF

- Generalized LAUC-VF: Este *scheduler* es una extensión del algoritmo LAUC-VF para incluir características de QoS. Las ráfagas de datos se dividen en *n* clases dependiendo de sus requerimientos de QoS. En el nodo *edge* la ráfaga de datos se ensambla agregando paquetes de la misma clase y con el mismo nodo *edge* de destino. En los nodos *traversing*, las ráfagas de datos que van al mismo enlace de salida se planifican con el *scheduler* asociado a ese enlace. Para cada enlace, su *scheduler* mantiene *n* colas $Q_1, Q_2,\ldots,Q_n$ siendo Q_i la cola encargada de almacenar los BHPs de la clase *i* en un orden FIFO. Para cada *slot*, el algoritmo se ejecuta una vez. Asumiendo que las ráfagas de datos de la clase *i* tienen una prioridad mayor que las ráfagas datos de la clase *j* siendo $i<j$, este algoritmo garantiza que aquellas ráfagas de datos cuyas BHPs se encuentran en la cola Q_i son planificadas antes que las ráfagas de datos cuyas BHPs se encuentran en la cola Q_j . Este algorimos LAUC-VF generalizado, emplea el algoritmo LAUC-VF como subalgoritmo. Dado que el algoritmo LAUC-VF se puede implementar en hardware, G-LAUC-VF también se puede implementar en hardware. Debido a las restricciones de tiempo real, es muy importante la simplicidad del algoritmo G-LAUC-VF. Usando LAUC-VF como subalgoritmo, se evita el completo rediseño de un algoritmo y su correspondiente implementación en hardware.

- MIN-SV (*Minimum Starting Void*)y MIN-EV(*Minimum Ending Void*): MIN-SV selecciona un hueco interno entre aquellos huecos elegibles para una ráfaga dada minimizando el hueco temporal entre el tiempo donde comienza el huevo y el tiempo de llegada de la ráfaga. MIN-EV minimiza el hueco entre el tiempo de llegada del último bit de la ráfaga de datos y el tiempo final del hueco. MIN-SV tiene un rendimiento un poco mejor que MIN-EV pero emplea un mayor tiempo de planificación para una ráfaga de datos de entrada. MIN-SV y MIN-EV presentan los mejores resultados en cuanto a relación entre

tasa de pérdida de ráfagas y tiempo de planificación. Una comparativa de los *schedulers* presentados se muestra en la Fig. 7.

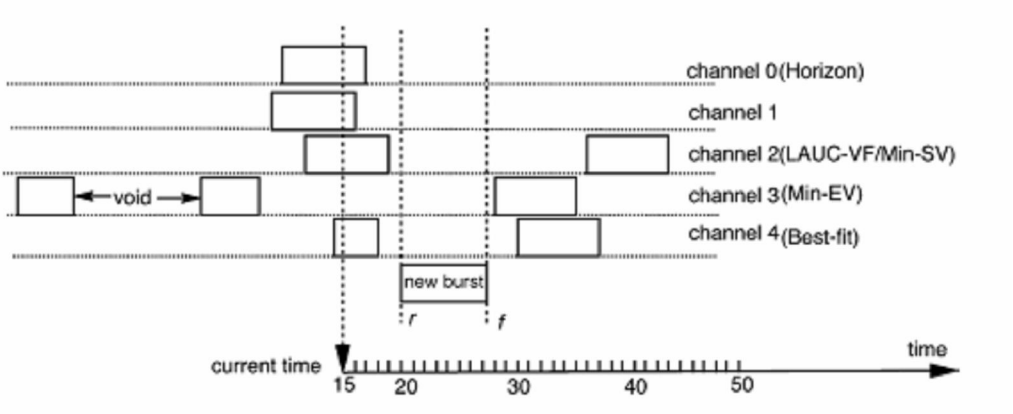

Fig.7 Comparativa de Schedulers

- HVF (*Heap Void Filling*): Cuando se recibe un BHP, se investigan todas las longitudes de onda que pertenecen al enlace de salida. Si existe al menos una longitud de onda cuyo horizonte temporal es menor que el tiempo de llegada que presenta la ráfaga, la ráfaga se envía en esa longitud de onda. Si más de una longitud de onda se encuentra disponible, aquella que minimiza el hueco entre dos ráfagas consecutivas es la que se elige. Si no existen longitudes de onda que satisfagan las restricciones, el algoritmo busca uno de los huecos internos elegibles previamente generado y en particular se elige el más antiguo (aquel que tiene el tiempo de inicio más temprano). Claramente, un hueco elegible significa un intervalo cuyo tiempo de inicio es más bajo que el tiempo de llegada de la ráfaga y con un tiempo de final que es más alto que el de la cola de la ráfaga en cuestión. La ráfaga de datos se descarta si no hay disponible ningún intervalo.

1.4 Política de QoS: DiffServ over MPLS

Differentiated Services (DiffServ) se propuso por el IETF como una solución escalable de QoS para Internet de nueva generación. Desarrolla de una forma sencilla, métodos para suministrar diferentes niveles de servicio para el tráfico. DiffServ fija el número de posibles clases de servicio, siendo por tanto independiente del número de flujos o usuarios existentes y de una complejidad constante. Además otorga los recursos a un número pequeño de clases que agrupan varios flujos en lugar de a flujos individuales.

Multi-Protocol Labeling Switching surge del esfuerzo de llevar las características de los circuitos virtuales al mundo IP. Proporciona una administración del ancho de banda a través del control del enrutamiento empleando para ello las etiquetas que se encuentran encapsuladas en la cabecera de los paquetes.

Se pueden identificar dos similitudes entre los dos enfoques:

- Ambos enfoques aplican la complejidad en los nodos frontera de la red. Esta propiedad ayuda a la escalabilidad y hace que estas arquitecturas sean adecuadas para redes *backbone*.

- Ambas tecnologías usan etiquetas de un pequeño tamaño para implementar QoS.

Esto nos lleva a que la política de QoS que vamos a implementar sea una combinación de técnicas DiffServ junto con técnicas MPLS, adquiriendo lo necesario para nuestros propósitos de cada una de las tecnologías.

1.4.1 PHB Scheduling Class (PSC)

El tráfico que proviene de una red de usuarios y quiere acceder al dominio DiffServ, tiene que pasar a través de un nodo frontera. Este nodo realiza los trabajos de clasificación de paquetes y de acondicionamiento de tráfico, es decir, identifican a que clase pertenece un paquete y se monitoriza si un determinado flujo de datos cumple con un acuerdo de servicio. Este tráfico es dividido en un pequeño número de grupos denominado *forwarding class*.

Cada una de las clases representa un tratamiento predefinido en términos de asignación de ancho de banda y prioridad de descarte. La clase a la que un paquete pertenece, se codifica dentro de la cabecera del paquete de control (*Burst Header Packet*)

Entre las clases que han sido estandarizadas de mayor prioridad a menor tenemos, EF (*expedited forwarding*), AF (*assured forwarding*) y *best-effort*. La clase EF se empleará para el tráfico más prioritario.

1.4.2 Establecimiento de LSPs

En MPLS, un circuito virtual se le denomina LSP (*Label-Switched Path)*. MPLS emplea una etiqueta de tamaño fijo insertada en la cabecera de paquete para transmitir los paquetes. Cada nodo emplea la etiqueta de la cabecera del paquete como índice para encontrar el siguiente salto y la nueva etiqueta correspondiente. El paquete es enviado al siguiente salto después de que la etiqueta existente el paquete sea intercambiada con una nueva etiqueta para el siguiente salto. El camino que atraviesa el paquete a lo largo de la red es definido por las transiciones en los valores que toman las etiquetas. Ya que el mapeo entre etiquetas es fijo en cada nodo, un LSP queda determinado por el valor que tome la etiqueta en el nodo origen del LLSP [4].

En nuestro caso, implementamos la estrategia de aplicación de DiffServ indicado en la RFC 3270 [5], y denotada como L-LSP. Esto implica que un LSP transportará paquetes que tengan nodo de salida común, y que sean de la misma *packet schduling class*. La RFC distingue el concepto de *packet scheduling class* (PSC), definiéndolo como un conjunto de PHBs relacionados.

1.4.3 RSVP-TE

El protocolo RSVP-TE es una extensión del protocolo RSVP para el establecimiento de LSPs y para la distribución de las etiquetas en MPLS. Este protocolo soporta la creación de LSPs con una determinada ruta específica con o sin reserva de recursos. La ruta específica se determina mediante un campo ERO (*Expedited Route Object*) que se envía en el mensaje RSVP. RSVP-TE también permite crear rutas nuevas para un determinado LSP, y otras opciones como detecciones de bucles.

En el sistema que se ha implementado, los mensajes RSVP son generados al inicio de la simulación. Cada mensaje contiene una ruta específica para el LSP que viene indicada en su ERO por lo que al iniciar la simulación, van a quedar establecidos todos los LLSPs. Además, cada mensaje indica la tasa media en paquetes por microsegundo que son necesarios para el LLSP por lo que cada nodo de la ruta que recibe el mensaje, reserva la tasa necesaria de la capacidad que tenga a su salida para satisfacer los requerimientos de este LLSP. En esta implementación no se utilizan técnicas de reenrutado por lo que los LLSPs van a quedar fijos en el inicio de la simulación y no se van a variar durante ésta.

Los nodos destino al recibir un mensaje RSVP aceptan el LSP enviando hacia el nodo origen un mensaje RESV que sigue el mismo camino que el paquete RSVP pero en orden inverso. En la implementación que se ha realizado, no se generan mensajes RESV en respuesta a la llegada de un paquete RSVP al nodo destino del LSP. Si algún camino no puede crearse por falta de capacidad o por que no sea posible seguir la ruta explícita, se obtiene un mensaje de error por consola y finaliza el programa.

1.4.4 Etiquetas

Una etiqueta es un identificador con un tamaño fijo, de significado local que es empleada por el nodo para planificar los paquetes. Esta etiqueta se añade al paquete por lo que se dice que el paquete está etiquetado.

MPLS permite (pero no es un requerimiento) que la prioridad o clase de servicio sea completamente o parcialmente deducida de la etiqueta. Para un nodo que está conectado mediante un enlace punto a punto, la interfaz puede usar cualquier identificador para la etiqueta. Cuando el nodo tiene múltiples enlaces, las etiquetas tienen que ser asignadas de forma única entre todas las interfaces que tenga el nodo.

En la implementación realizada, la etiqueta que se le asocia a un determinado LSP en un nodo se decide cuando el nodo recibe el mensaje RSVP para la creación de un LSP. El espacio de etiquetas que se pueden emplear es único para cada fibra óptica de salida (no puede haber dos LSP con la misma fibra óptica de salida y misma etiqueta a la salida) e idéntico para el conjunto de fibras de salida (se puede emplear la misma etiqueta en fibras de salida distintas).

Existen protocolos para la distribución de las etiquetas como son LDP, CR-LDP…pero que no se han empleado en la implementación realizada ya que asumimos que las etiquetas se definen en la fase de establecimiento de los LSPs (fase inicial de la simulación donde se envían los mensajes RSVP) y permanecen inalteradas en el resto de la simulación.

1.4.5 Label Switching Table

La *Label Switching Table* también denominada *Incoming Label Map* mantiene el mapeo entre una etiqueta de entrada hacia una etiqueta de salida y una interfaz de salida. Su función es similar a la de una tabla de encaminamiento IP. La entrada de la tabla a la que una etiqueta de entrada apunta se le denomina NHLFE (*Next-Hop Label-Forwarding entry*). Cada etiqueta de entrada, típicamente apunta a una NHLFE [4].

Típicamente, la NHLFE contiene el siguiente salto en la red y la etiqueta de salida hacia el siguiente salto. Si un nodo es el nodo *ingress* o *egress* del LSP, la entrada NHLFE también especifica las acciones a realizar con el tráfico a través de un etiquetado especial.

En la implementación que se ha realizado, para el tráfico generado en el propio nodo, existe una tabla únicamente para este tipo de tráfico con entradas NHLFE cuya etiqueta de entrada y fibra óptica de entrada valen -1. La forma de acceder a ella es mediante una variable que identifica al LSP generado en el propio nodo.

Para los LSPs que atraviesan el nodo o los que su destino sea el propio nodo existe otra tabla con entradas NHLFE. Para el caso de ser el nodo *egress* el indicativo de la tabla para la etiqueta de salida y de la fibra óptica de salida valen -1.

1.4.6 Gestión de las colas ingress

Las ráfagas que son generadas por el módulo *assembler* son paquetes OBS en formato eléctrico con información relacionada con el flujo de datos que lo ha generado. La información referente al LSP es añadida por el módulo *ingress* cuando recibe Las ráfagas.

Una vez recibido el paquete por el módulo *ingress*, se produce la clasificación del paquete en colas eléctricas en función del LSP al que pertenece.

El elemento que decide cuando salen las ráfagas de las colas eléctricas es el *Scheduler* el cual también se encarga de controlar las colas ópticas (FDLs) empleadas para las ráfagas que son recibidos por las fibras ópticas de entrada y que permanecen en formato óptico (en el caso de ser un nodo *traversing*).

Este *ingress* informa al *scheduler* de que existe una ráfaga que se encuentra almacenada en una cola óptica y que tiene una determinada fibra de salida y un offset mínimo. El *scheduler* lo que hace es generar un BHP con el offset mínimo que le indica el módulo *ingress* y lo envía de manera inmediata (en el primer momento donde encuentre hueco a la salida) retardando la salida del *payload* de la cola electrónica el tiempo de offset mínimo.

1.4.7 Provisión de QoS en redes OBS

Existen varias estrategias que se han considerado en la literatura para suministrar resolución de contención con provisión de QoS en redes OBS entre las cuales están el adelantamiento de ráfaga y la diferenciación basada en offset. Estas opciones pueden ofrecer una mayor diferenciación entre clases.

- Adelantamiento de ráfaga (*Burst Preemption*): En este caso se permite sobrescribir recursos reservados para ráfagas de baja prioridad por reservas de alta prioridad en caso de conflictos de ráfaga. Vamos a considerar en este caso la arquitectura de red OBS con FDLs en los puertos de entrada de los nodos *traversing*. En esta arquitectura, los *edge nodes* no van a insertar ningún tipo de offset inicial entre el BHP y el *payload*. El paquete de control y el *payload* van a viajar de forma simultánea a través de la red. Cuando ambos paquetes alcanzan el nodo *traversing,* el paquete de control pasa directamente a la unidad de control mientras que la ráfaga se retarda en la FDL del puerto de entrada. Durante el tiempo en que el paquete de control es procesado la

unidad de control puede adelantar su reserva con una de mayor prioridad. La regla importante de este mecanismo es que el paquete de control, después de ser procesado, está esperando a su ráfaga en la memoria de la unidad de control hasta que ésta abandona la FDL de la entrada y ambos son enviados juntos al siguiente nodo (si la ráfaga no ha sido adelantada) o descartada (en caso de un adelantamiento con éxito). Después de que el paquete de control se haya enviado y la ráfaga esté siendo transmitida, ya no se permite el adelantamiento en el nodo.

- Diferenciación basada en offset (*Offset-time Differetiation*): En este caso se va a asignar un tiempo extra de offset a las ráfagas de mayor prioridad lo que resulta en una reserva más temprana con el fin de favorecerlas mientras la reserva de recursos se está realizando.

La principal desventaja de los mecanismos de adelantamiento en las redes OBS convencionales es que es necesaria una señalización adicional en caso de que se produzca un adelantamiento con éxito con el fin de liberar los recursos que ya se han reservado en el camino de salida.

1.5 Objetivos del proyecto

En este proyecto se va a estudiar el comportamiento de una red OBS asíncrona mediante la implementación de un simulador en Omnet.

La flexibilidad de este programa nos va a permitir implementar distintas soluciones para aspectos como son la planificación seguidas por los *Schedulers*, la topología de la red…

Entre los distintos puntos que se van a estudiar cabe destacar:

1. Adaptación del simlador oPASS existente para redes OPS, para el estudio de las prestaciones de las redes OBS.

2. Estudiar las distintas propuestas para los *schedulers* secuenciales OBS, y compararlas con una alternativa nueva, que es un *scheduler* paralelo e iterativo.

3. Hacer un estudio del límite de prestaciones que se puede alcanzar con *bufferless* OBS, planteando un escenario de test lo más favorable posible.

El resto de este documento se organiza de la siguiente manera:

- Capítulo 2 describirá los elementos que forman el simulador así como la estructura del código implementado en Omnet++.

- Capítulo 3 comparará las prestaciones del *scheduler* paralelo e iterativo frente a los *schedulers* secuenciales.

- Capítulo 4 estudia las prestaciones de la red OBS sin buffer.

- Finalmente el capítulo 5 da conclusión a este trabajo.

Capítulo 2
Extensión del simulador oPASS para redes OBS

2.1 Simulador oPASS

En este capítulo, vamos a empezar explicando las características más importantes de la herramienta oPASS desarrollada sobre OMNET++ e inicialmente empleada para simulación de redes OPS. En la siguiente sección nos centraremos en explicar los cambios realizados en la herramienta para implementar topologías de red OBS.

2.1.1 Descripción de la arquitectura global del sistema

En el sistema se han definido dos tipos de módulos simples Omnet++ que derivan de la clase *cSImpleModule:*

- *Switch General*: Es el módulo que simula el comportamiento de un nodo de la red (tanto *traversing* como *egress*). Existirá un módulo simple de este tipo por cada nodo que haya en la arquitectura que el usuario desee simular. Cada enlace de fibra óptica que une dos nodos se simula mediante una puerta a través de la cual los nodos se pueden enviar mensajes. El *Switch General* Se va a encargar de procesar todos los mensajes que se reciben del exterior o que se producen en el propio nodo. Los tipos de mensajes que se pueden enviar los nodos se explican en la siguiente sección. Cada uno de estos módulos está formado por un conjunto de interfaces interconectadas entre sí y a las cuales se les encarga una tarea específica. Cada *Switch General* contiene una interfaz de cada tipo (*Scheduler, Ingress, OpticalPacketAssembler y Egress Data Manager)*.

- *Global Stat Controler*: Es el módulo que se encarga de controlar el proceso de simulación con respecto al tiempo transitorio, el final de las muestras de simulación y el final de la simulación. Este módulo es el encargado de avisar a cada *Switch General* del estado de la simulación. Existe un único módulo de este tipo para todo el sistema. Este módulo avisa a cada uno de los nodos de la red del estado en el que se encuentra la simulación a través de la puerta que le une con cada uno de los nodos (módulos *Switch General).*

Para interconectar el módulo *Global Stat Controler* a cada uno de los módulos *Switch General* de los que está compuesto el sistema se emplean puertas a través de las cuales los módulos se intercambian mensajes. El tipo de puerta es unidireccional con salida en el módulo *Global Stat Controler* y entrada en el módulo *Switch General*. Es por esto por lo que cada módulo *Switch General* va a consumir los mensajes producidos por el módulo *Global Stat Controler. Estos mensajes* le van a indicar a cada nodo un cambio en el estado en el que se encuentra la simulación.

La figura 8 indica los módulos que hay en una simulación y su interconexión mediante las puertas:

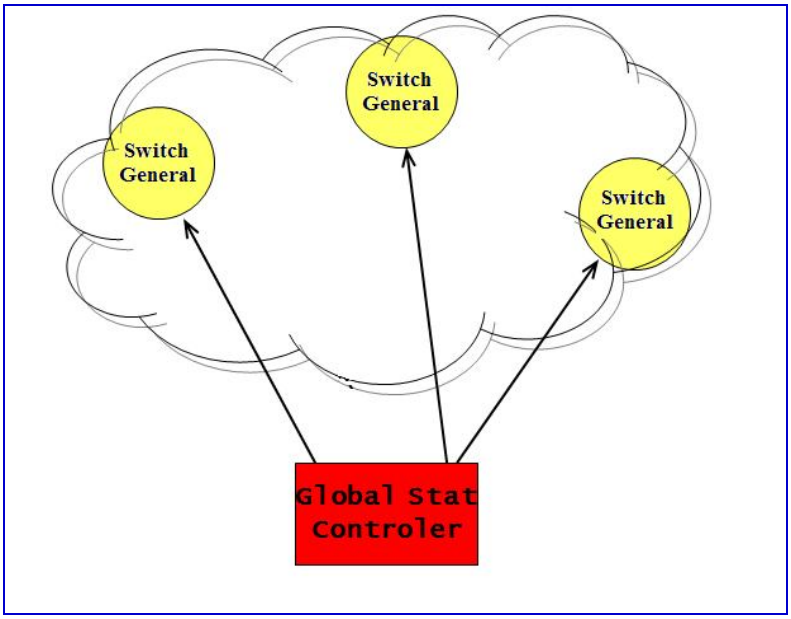

Fig.8: Esquema de la arquitectura

Además de los dos tipos de módulos Omnet, vamos a definir los tipos de mensajes que se pueden enviar los módulos simples. Para ello creamos un nuevo fichero denominado GMPLSOpticalPacket.msg donde se definen cada uno de los posibles mensajes y los campos que van a tener cada uno de ellos.

2.1.2 Mensajes definidos en la herramienta

En el fichero GMPLSOpticalPacket.msg se van a definir todos los tipos de mensajes que se van a enviar los módulos simples de Omnet definidos.

Los tipos de mensajes son los siguientes:

- BurstHeaderPacket

  Tipo de mensaje que simula el comportamiento de un paquete de control BHP. Entre sus campos se encuentran el *edge node* que lo generó, el *edge node* destino de la ráfaga de datos, la clase del paquete (PSC), el LLSP al que pertenece la ráfaga, la etiqueta MPLS que lleva, su longitud óptica (en microsegundos), el offset que hay entre el paquete de control y la llegada de la ráfaga, el identificador del flujo al que pertenece, la longitud de onda de transmisión en cada enlace y el ERO. Un campo importante que se le ha añadido es el contador del número de saltos que atraviesa para arquitecturas de red multinodo. Tiene en cuenta como salto el que realiza desde el *edge node* al primer *traversing node* y desde el último *traversing node* del LSP hasta el *edge node* destino.

- IntraAssemblerSignalPacket

  Mensaje planificado por el Assembler para generar una nueva ráfaga para un determinado flujo de datos de un LSP. El tipo de *assembler* simulado es el híbrido como se explica en el apartado 1.3.1.

- StatControllerSignal

  Mensaje que envía el módulo GlobalStatControler a cada uno de los nodos para indicarle el final de una determinada muestra o que ha terminado el tiempo transitorio de la simulación.

- GMPLSOpticalRSVPPacket

  Tipo de mensaje que simula el comportamiento de un mensaje RSVP de establecimiento de LLSP en la red. Este mensaje es generado en el proceso de inicialización del módulo GlobalStatControler y se envía cuando la simulación aún no ha comenzado. Los campos que contiene un mensaje RSVP son el nodo que generó el mensaje, el LLSP al que pertenece el mensaje, la clase del tráfico, la carga en paquetes por microsegundo que vamos a requerir de los enlaces por los que pase el LLSP, la duración óptica media del paquete en microsegundos, el ERO y la etiqueta correspondiente.

2.1.3 Ficheros Omnet.ini y Sistema.ned

La configuración de una simulación y los datos de entrada que ésta necesita se describen en un fichero de configuración normalmente denominado omnetpp.ini. La utilidad de este fichero es la de aportar parámetros a la simulación de manera que no haya que recompilar una simulación cada vez que queramos modificar un parámetro de la misma. Este fichero se encuentra agrupado en secciones denominadas [General], [Cmdenv] , [Parameters] y [Run] las cuales contienen diversas entradas.

La sección [General] contiene especificaciones generales que se aplican a todas las simulaciones que se realicen y a todas las interfaces de usuario.

La sección [Cmdenv] le indica al Cmdenv (línea de comandos de la interfaz de usuario para la ejecución por lotes) como tiene que ejecutar la simulación y las actualizaciones que tiene que imprimir sobre el progreso de la simulación.

La sección [Parameters] asigna valores a los parámetros que no han conseguido un valor dentro del los ficheros .NED.

La sección [Run] define los parámetros de la red a simular. Como los parámetros de las simulaciones son ***inabordables*** manualmente (debido a su gran número) se han generado unos *scripts* de Matlab que ayudan a su generación y facilitan su cambio (ver apartado 2.1.5).

La descripción de la topología de red viene dada en el lenguaje .NED. Este lenguaje soporta la descripción modular de una red. Esto significa que la descripción de una red consiste en la descripción de un número de componentes (canales, módulos simples o compuestos). Los canales, los módulos simples y los compuestos de una descripción de red pueden ser empleados en cualquier otra descripción de red.

Este fichero declara un módulo de sistema o de red, el cual integra al resto de módulos. Una descripción NED puede contener los siguientes componentes en distinto número y orden:

- Sentencias de importación
- Definición de canales

- Declaraciones de módulos simples y compuestos
- Declaración de un sistema modular

En el caso del sistema a simular, no importamos módulos simples sino que se definen en el propio fichero dos tipos de módulos simples como son el módulo Switch_General y el módulo GMPLS_GlobalStatController_v1. Para cada módulo simple se definen parámetros y puertas:

- Los parámetros son variables que pertenecen al módulo. Los parámetros de un módulo simple pueden ser consultados y usados por los algoritmos del módulo simple.
- Las puertas son los puntos de conexión para los módulos. Los puntos de comienzo y fin de las conexiones entre los módulos son las puertas.

Para cada uno de los dos módulos simples de los que está compuesta la red, los parámetros y puertas que emplean varían de una arquitectura de red a otra (solo con varia por ejemplo el número de fibras ópticas de entrada, necesitamos un .NED nuevo). Es por ello, que para ayudar a la automatización de la generación de los ficheros .NED, se han generado unos *scripts* en Matlab que ayudan a ello (ver apartado 2.1.5).

2.1.4 Control de la simulación

La razón para llevar a cabo la simulación de un sistema es hacer posible la evaluación de sus prestaciones. En cada una de las simulaciones que el usuario realiza, se obtienen un conjunto de datos estadísticos como resultado. Para que estos resultados se parezcan lo más posible a la realidad, se necesita realizar una simulación de tiempo infinito pero como cabe de esperar, realizar una simulación de este tipo es imposible.

Tener una simulación en tiempo infinito es equivalente a conocer el comportamiento del sistema en régimen permanente, por lo que se debe de eliminar el periodo inicial o transitorio para que no afecte a las estimaciones que se realicen. En nuestro caso, se elimina el efecto del transitorio, iniciando la toma de datos cuando el sistema lleva un periodo de tiempo en funcionamiento. El periodo de tiempo transitorio que se desea eliminar de la simulación del sistema, es un parámetro que introduce el usuario en el fichero Omnet.ini.

El módulo simple de Omnet encargado de controlar el tiempo transitorio de la simulación es GlobalStatControler. Este módulo se encarga de avisar a todos los nodos de la red de que se ha llegado al final del tiempo transitorio por lo que deben de inicializar las variables que servirán para la toma de estadísticos.

Una vez eliminado el transitorio, el siguiente paso es la toma de los estadísticos necesarios. Para ello se divide el tiempo de simulación en intervalos de tiempo idénticos o muestras con lo que transformamos un proceso continuo en el tiempo en otro discreto. Para cada una de las muestras, el sistema únicamente inicializa las variables que sirven para la toma de estadísticos pero no inicializa el sistema completo (los mensajes y paquetes siguen su curso normal). Al finalizar cada una de las muestras se almacenan las variables de interés en ficheros para su posterior procesado y evaluación.

El número de muestras tomadas y la duración en tiempo real de simulación de cada una de las muestras (medido en microsegundos de tiempo real) es un parámetro definido por el usuario en el fichero Omnet.ini. El módulo encargado de controlar el final de cada una de las muestras es GlobalStatControler. Este módulo envía un mensaje a cada uno de los nodos cuando se llega al final de una de las muestras para que éstos inicialicen sus variables.

Cuando se han realizado todas las muestras, el sistema llega al estado de final de simulación. En este caso, es de nuevo el módulo GlobalStatControler el que avisa a cada uno de los nodos de que la simulación ha terminado para que se realice un reiniciado del sistema. En este caso no solo se inicializan las variables sino que se borran todos los mensajes almacenados y eventos producidos y no consumidos, liberando la memoria.

Una vez que se ha reiniciado el sistema de forma completa, se puede pasar a ejecutar otra simulación de nuevo. En este caso se pueden modificar los parámetros de entrada, la semilla de los generadores de números pseudoaleatorios…

El número de simulaciones que se realizan, las define el usuario a través del fichero Omnet.ini. Cada una de las simulaciones almacena los ficheros generados en carpetas distintas para facilitar el procesado y la evaluación posterior.

2.1.5 Scripts de Matlab

Los Scripts de Matlab son archivos que contienen una sucesión de comandos análoga a la que se teclearía en una ventana de comandos.

En dos puntos del proceso de simulación del sistema, se han empleado dos scripts de Matlab, los *scripts* multiNodeWriteIniNed.m / monoNodeWriteIniNed.m y el readSimulationPoint.m.

Los pasos a seguir dentro de la simulación del sistema de forma gráfica son los siguientes:

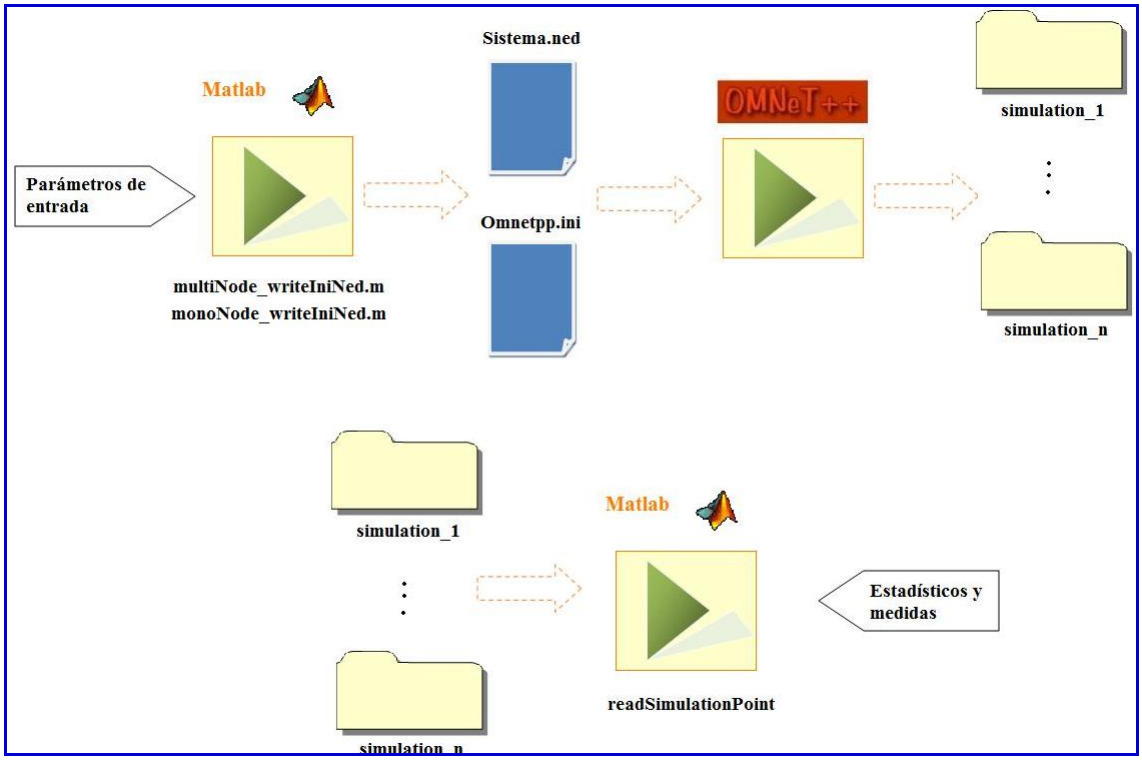

Fig.9: Esquema de los Scrits de Matlab

Vamos a ver cada uno de los pasos:

- Parámetros de entrada: Reciben todos los parámetros de entrada necesarios para generar los ficheros Omnet.ini y sistema.NED. Estos parámetros son los necesarios para generar las simulaciones mononodo o las multinodo. En las simulaciones multinodo, el script se va a encargar de temas como la optimización para el enrutamiento…

- multiNode_writeIniNed: Es el script en Matlab encargado de tomar los parámetros de entrada correspondientes al sistema a simular, para generar después los ficheros Omnetpp.ini y Sistema.ned. Estos ficheros generan en Omnet un módulo compuesto que simula el sistema y cuyos valores vienen dados en el fichero Omnetpp.ini. Lo empleamos en topologías multinodo.

- monoNode_writeIniNed: es similar al *script* multiNode_writeIniNed pero para topologías mononodo.

- Una vez generados los ficheros Omnetpp.ini y Sistema.ned, son ejecutados en Omnet++ junto con el resto de interfaces y de clases implementadas para el sistema. Como resultado se obtiene un número de carpetas en función de las simulaciones que se realicen en cuyo interior se encuentran los ficheros que almacenan los estadísticos deseados.

- Las carpetas generadas en las simulaciones de Omnet++ junto con los ficheros donde se encuentran los estadísticos, son pasadas como parámetros de entrada al script de Matlab readSimulationPoint. Este script trabaja con *struct* para presentar los datos de una forma ordenada y sencilla de utilizar en los posteriores *scripts* para la generación de gráficas y datos concluyentes.

2.2 Descripción de la extensión desarrollada

Vamos a ver en este apartado las variaciones más importantes que se han realizado al código original del oPASS en dos apartados:

- Nuevas implementaciones que heredan de las interfaces.
- Modificaciones a los scripts de Matlab para tener en cuenta las nuevas funcionalidades añadidas.

2.2.1 Descripción de las implementaciones

2.2.1.1 Implementación Assembler_ExponentialIATGaussianBurst_EdgeNode

Funcionalidad añadida a la interfaz

Esta clase hereda de la interfaz OpticalPacketAssembler. Es el *assembler* que van a implementar los *edge nodes*. Para esta implementación, la generación de ráfagas seguirá una tasa de tipo exponencial cuya media vendrá definida como un parámetro ajustable por el usuario en los datos incluidos en el fichero Omnet.ini. Para el tamaño medio de ráfaga, en esta implementación seguirá una distribución normal truncada donde los valores máximos y mínimos se obtienen del fichero Omnet.ini.

Parámetros que recibe del fichero Omnet.ini

minBurstSize	Tamaño mínimo de ráfaga en la distribución normal truncada
maxBurstSize	Tamaño máximo de la ráfaga en la distribución normal truncada
meanBurstSize	Tamaño medio de la ráfaga
varianceBurstSize	Varianza de la distribución normal truncada

Mensajes generados

ID_SIGNALTOGENERATEOPTICALPACKET	Mensaje que se planifica al generar un nuevo flujo de datos en el generador de ráfagas. Este mensaje es del tipo IntraAssemblerSignalPacket, el cual tiene un campo que almacena el identificador del flujo al que pertenece. Este mensaje indica que se debe de generar una nueva ráfaga con tasa exponencial. Además, cada vez que se genera una nueva ráfaga de datos, el mensaje se vuelve a planificar con una tasa de llegada

	exponencial. Cuando se saca el mensaje de la lista de eventos futuros (*FEL*), el mensaje es recibido por el módulo Switch General que lo reenvía hacia el OpticalPacketAssembler.

Mensajes consumidos

ID_SIGNALTOGENERATEOPTICALPACKET	El mensaje reenviado por el Switch General es consumido por el Assembler para generar una nueva ráfaga con una tasa exponencial. Una vez generada y enviada la ráfaga, el mensaje se vuelve a planificar con una tasa de llegada exponencial. Se almacena como un campo más dentro del vector que almacena la información de todos los flujos de datos cuyo nodo *ingress* es el propio nodo.

Implementación de las funciones declaradas en la interfaz

virtual void handleAssemblerMessage(cMessage*msg)

Genera una ráfaga siguiendo una exponencial con un tamaño medio de ráfaga que sigue una distribución normal truncada. Actualiza la información referente al flujo de datos que se almacena en un *struct* denominado *AssemblerFlowInfo* y planifica la generación de un nuevo paquete óptico con una tasa de llegada exponencial.

virtual int addTrafficFlow (double ppsRate, int egressNode, int packetPSC)

Almacena en un vector denominado *control*, la información relacionada con el nuevo flujo de datos. Después, planifica la generación de una ráfaga con una tasa de llegada exponencial en función de la carga asociada.

virtual void terminateTrafficFlow (int trafficFlowId)

Cancela la generación ráfagas para un determinado flujo de datos. Elimina el evento planificado para generar una nueva ráfaga de datos el cual se encuentra almacenado en el vector denominado *control*.

virtual void endSimulationSample (int sampleNumber)

Almacena los estadísticos que son de interés en un fichero de la forma simulation_xSy_Nodez_Assembler.dat.. Estos estadísticos son perFlowInfo_egressNode, perFlowInfo_packetsPerMicrosecRate, perFlowInfo_averageOpticalPacketDurationMicrosec, perFlowInfo_psc, perFlowInfo_associatedLlspIngressInternalIndex, perFlowInfo_numberOpticalPacketsAssembled. Una vez almacenados los estadísticos, se llama a la función endTransitoryTime( ).

virtual void *endTransitoryTime ( )*

Elimina los elementos almacenados en el vector *control* que contiene la información de cada uno de los flujos de datos cuyo nodo *ingress* es el propio nodo.

Funcionalidad particular de la clase Assembler_ExponentialAtNoBurst

Assembler_ExponentialIATNoBurst (Switch_General *fatherNode, const char*inputParameters)

Constructor de la clase *Assembler_ExponentialAtNoBurst*. Almacena una referencia a la clase Switch_General.

~Assembler_ExponentialIATNoBurst ( )

Destructor de la clase *Assembler_ExponentialAtNoBurst* . Elimina todos los elementos almacenados en el vector que contiene la información de cada uno de los flujos de datos cuyo nodo *ingress* es el propio nodo.

2.2.1.2 Implementación del IngressQueuesManager_EdgeNode

Funcionalidad añadida a la interfaz

Esta clase hereda de la interfaz IngressQueuesManager. Implementa un módulo *ingress* el cual inyecta las ráfagas generadas por el módulo *Assembler* en el módulo *Scheduler.* Es el módulo *ingress* que van a implementar los *edge nodes*. En él, se van a recibir las ráfagas generadas por el *assembler* a las que se les va a poner un offset en función del número de saltos que tengan que dar hasta el *edge node* destino. Una vez que se le pone el offset, se le avisa al *scheduler* de que hay una ráfaga esperando en cola electrónica para ser inyectada en la red OBS. El *scheduler* después será el encargado de solicitarle la ráfaga al *ingress.*

Parámetros que recibe del fichero Omnet.ini

NUMBER_TRAFFIC_FLOWS_PER_LLSP	Indica el número de flujos de datos que se tienen que generar para cada LLSP.
BHP_PROCESSINGTIME	Es el tiempo que tarda cada nodo de la red (nodos *traversing*) en procesar los paquetes de control BHPs.
FDLPERINPUTPORTLENGTH_EACHNODE	Tamaño en microsegundos de la FDL en los puertos de entrada de los nodos *traversing*.

Mensajes consumidos

BurstHeaderPacket	El *ingress* consume los mensajes generados por el *Assembler_ExponentialAtNoBurst* (ráfagas de datos en formato electrónico) .

Implementación de las funciones declaradas en la interfaz

virtual void *handleAssemblerMessage(cMessage*msg)*

Recibe las ráfagas generadas por el *Asssembler.* Realiza un proceso de marcado previo con el tráfico recibido (EXP bits). Etiquetamos el tráfico en función de la fibra óptica que se vaya a emplear para salir del nodo. Añadimos el paquete a la cola correspondiente al LLSP al que pertenece el flujo de datos. El *ingress* informa al *Scheduler* de que ha recibido un paquete de un determinado LLSP ,con destino a una determinada fibra óptica de salida y con un determinado offset temporal.

virtual int add_LLSP (int llspEgressNode, int llspPSC, int outputFiber, double ppmicrosecRate, double averageOpticalPacketDurationMicrosec, vector <int> ero)

Esta función es llamada por el módulo GMPLSSignalingModule cuando un nodo recibe un mensaje RSVP cuyo nodo *ingress* del mensaje LLSP es el propio nodo. Inicialmente, el *ingress* almacena la información correspondiente al LLSP en un vector. A continuación se establecen tantos flujos como indique la variable NUMBER_TRAFFIC_FLOWS_PER_LLSP con una tasa igual a la tasa total en ráfagas por microsegundo del LLSP dividida por el número de flujos por LLSP. Para ello se llama a la función *addTrafficFlow* del módulo *Assembler*. Una vez creados los flujos, se almacenan la información correspondiente a cada uno en un vector denominado *perTrafficFlowInfo*. También realiza el cálculo del offset que le añadirá a cada ráfaga cuando la reciba en función del ero correspondiente a ese LSP.

virtual void terminate_LLSP (int llspIngressInternalIndex)

Desactiva la generación de ráfagas para todos los flujos de datos de un determinado LLSP a través de la función del módulo *Assembler* terminateTrafficFlow (flowID).

virtual int getNumberOfPendingPacketsPerOutputFiberAndPSC (int outputFiber, int llspPSC)

Indica el número de ráfagas que hay en una determinada cola en el *ingress*. La cola queda determinada por la clase del tráfico del LLSP y por la fibra óptica de salida a la que va destinado el tráfico. Para ello tenemos un vector en el *ingress* encargado de almacenar información sobre el estado de las colas denominado perOutputFiberAndPSCInfo.

virtual GMPLSOpticalDataPacket * getPacketDirectedToOutputFiber (int outputFiber, int packetPSC)

Función invocada por el *Scheduler* para obtener una ráfaga de una determinada cola. En este caso la llamada la realiza el *scheduler* inmediatamente después de que el *ingress* le avise de

que hay una ráfaga en las colas eléctricas del *ingress* por lo que no se tienen que realizar ningún proceso de administración de colas en el *ingress* (como por ejemplo un round-robind).

virtual void endTransitoryTime ( )

Para cada uno de los LLSPs que se han creado en el *ingress*, ponemos a cero las variables que tienen en cuenta las ráfagas que se han generado.

virtual void endSimulationSample (int sampleNumber)

Se almacenan los estadísticos en un determinado fichero .

virtual void finish ( )

Elimina toda la información almacenada para todos los LLSPs.

Funcionalidad particular de la clase IngressQueuesManager_QoS1

IngressQueuesManager_QoS1(Switch_General *fatherNode , const char *inputParameters)

Obtiene los parámetros correspondientes del fichero Omnet.ini. Inicializa los vectores que contendrán la información de cada LLSP.

~IngressQueuesManager_QoS1 ( )

Elimina los vectores que almacenan información sobre cada LLSP y también las colas donde se almacenan las ráfagas.

2.2.1.3 Implementación Scheduler_EdgeNodeLAUC_VF

Funcionalidad añadida a la interfaz

Esta clase deriva de la interfaz Scheduler. La implementan los *edge* nodes de la red multinodo. Su función es la de recibir las ráfagas del módulo *ingress*, buscarles hueco en la fibra óptica de salida que tiene el nodo mediante la aplicación de una variación del algoritmo LAUC-VF y una vez que le encuentra hueco, envía de forma previa al envío de la ráfaga, un paquete de control BHP con el offset indicado por el módulo *ingress*. Esta variación del algoritmo LAUC-VF: selecciona el canal de transmisión que es capaz de acomodar el *payload* más pronto, pero no antes de offset inicial. También recibe los BHPs que ya han llegado al *edge node* destino y los envía al *egress* para que tome las estadísticas oportunas.

Parámetros que recibe del fichero Omnet.ini

BHP_PROCESSINGTIME	Es el tiempo que tarda cada nodo de la red (nodos *traversing*) en procesar los paquetes de control BHPs.
FDLINPUTPORT_ALLNODES	Tamaño en microsegundos de la FDL en los puertos de entrada de los nodos *traversing*.

Mensajes consumidos

BurstHeaderPacket	Recibe un BHP cuando es un *edge node* destino recibido por la fibra óptica de entrada y lo reenvía al módulo *egress* para la toma de estadísticos.

Implementación de las funciones declaradas en la interfaz

virtual void *handleMessage_BHP(BurstHeaderPacket *bhp)*

Esta función recibe los BHPs cuando el *edge node* hace de nodo *egress* del camino del LSP. Cuando recibe el BHP el *scheduler* reenvía el BHP al *egress* para que tome las estadísticas correspondientes.

virtual void newOpticalPacketInIngressQueue(int targetOutputFiber , int packetPSCdouble minOffsetTime)

Esta función es la que llama el *ingress* cuando recibe una ráfaga en una de sus colas eléctricas. Al llamar a esta función, el *scheduler* solicita al *ingress* que le envíe esa ráfaga para que le encuentre hueco a partir del algoritmo LAUC-VF modificado. Una vez que se ha encontrado hueco en la fibra óptica de salida, el *scheduler* envía un BHP previo al envío de la ráfaga (la ráfaga se envía un tiempo igual al indicado por el offset después del envío del BHP).

virtual void endTransitoryTime ( )

Ponemos a cero todos los estadísticos que después se van a almacenar cuando se llame a la función endSimulationSample.

virtual void endSimulationSample (int sampleNumber)

Se almacenan los estadísticos en un determinado fichero

virtual void finish ( )

En esta implementación del Scheduler no realiza nada.

2.2.1.4 Implementación Scheduler_SwitchOBAsynchLAUCVF_OBS

Funcionalidad añadida a la interfaz

Esta clase deriva de la interfaz Scheduler. La implementan los nodos *traversing*. Este *scheduler* es el encargado de planificar los BHPs que recibe de forma secuencial de sus fibras ópticas de entrada, mediante la aplicación del algoritmo LAUC-VF. No tiene ni parte de inyección de tráfico local ni envía tráfico al *egress.*

Parámetros que recibe del fichero Omnet.ini

NUMBER_FDLS	Número de FDLs a la salida para resolución de contención.
FDL_GRANULARITY	Granularidad de las FDLs a la salida
ELEMENTS_PAYLOAD_HISTOGRAM	Número de elementos que hay en los histogramas del *payload*.
minBurstSize	Tamaño mínimo de las ráfagas en la red.
maxBurstSize	Tamaño máximo de las ráfagas en la red.
BHP_PROCESSINGTIME	Es el tiempo que tarda cada nodo de la red (nodos *traversing*) en procesar los paquetes de control BHPs.
FDLINPUTPORT_ALLNODES	Tamaño en microsegundos de la FDL en los puertos de entrada de los nodos *traversing*.

Mensajes consumidos

BurstHeaderPacket	Recibe un BHP de las fibras ópticas de entrada y lo planifica para enviarlo al siguiente nodo en el camino del LSP tras aplicar el algoritmo LAUC-VF.

Implementación de las funciones declaradas en la interfaz

virtual void *handleMessage_BHP(BurstHeaderPacket *bhp)*

Esta función procesa los BHPs recibidos por las fibras ópticas de entrada. Una vez recibido el BHP se ejecuta el algoritmo LAUC-VF para encontrar hueco en una longitud de onda de salida

de la fibra óptica de salida objetivo. Tras encontrar hueco, envía al siguiente nodo en el camino del LSP el BHP modificando su campo de offset. Si el algoritmo no encuentra hueco, entonces descarta la ráfaga y genera un BurstHeaderPacket de tipo GMPLS_BURSTHEADERPACKET_PACKETLOSS que envía al siguiente nodo en el camino para informar al destino de donde se ha producido la pérdida de la ráfaga.

virtual void newOpticalPacketInIngressQueue(int targetOutputFiber , int packetPSCdouble minOffsetTime)

Esta función no se llama en esta implementación.

virtual void endTransitoryTime ( )

Ponemos a cero todos los estadísticos que después se van a almacenar cuando se llame a la función endSimulationSample.

virtual void endSimulationSample (int sampleNumber)

Se almacenan los estadísticos en un determinado fichero

updateStatistics_packetDecision(BurstHeaderPacket *bhp, int fdlIdAssigned)

Esta función se encarga de actualizar los histogramas tanto si se envía una ráfaga de forma correcta como si se descarta en el nodo.

virtual void finish ( )

En esta implementación del Scheduler no realiza nada.

2.2.1.5 Implementación Scheduler_SwitchOBParallelAsynchVarSize_QoS_NoPreemp_delayCycle1grant

Funcionalidad añadida a la interfaz

Esta clase deriva de la interfaz Scheduler. Es el *scheduler* que implementa el PI-OBS que se estudiará en el Capítulo 3. El algoritmo PI-OBS se ha diseñado como un algoritmo paralelo e iterativo, que es capaz de garantizar un límite superior al tiempo de respuesta del *scheduler*. Lo va a implementar un nodo *traversing* en simulaciones mononodo.

Parámetros que recibe del fichero Omnet.ini

NUMBER_FDLS	Número de FDLs a la salida para resolución de contención.
FDL_GRANULARITY	Granularidad de las FDLs a la salida
ALGORITHM_INTEREXECUTIONTIME	Tiempo entre dos ejecuciones consecutivas del algoritmo
ALGORITHM_RESPONSE_TIME	Tiempo que tarda en completarse una ejecución del algoritmo.
ACTIVEDEVICE_RESPONSE_TIME	Tiempo que tardan en configurarse las puertas lógicas de la matriz de conmutación óptica
NUMBEROFINPUTMODULESBLOCKS	Número de horizontes temporales del algoritmo (explicado más adelante en el apartado 3.2.2)
MAX_NUMBERITERATIONSALGORITHM	Número máximo de ciclos de retardo que se pueden realizar en una ejecución del algoritmo.
MINOFFSETTIME	Tiempo mínimo de offset permitido en la arquitectura de red.
MAXOFFSETTIME	Tiempo máximo de offset permitido en la arquitectura de red.
TIMEGAPLENGTHHISTOGRAMS	Longitud temporal entre dos intervalos consecutivos de los histogramas.
minBurstSize	Longitud temporal mínima de la ráfaga permitida en la arquitectura de red.
maxBurstSize	Longitud temporal máxima de la ráfaga permitida en la arquitectura de red.
checkOffsetConstant	Indica si vamos a emplear un offset constante que no varíe en la arquitectura de red o no.
numberIterationsPriorityHighClass	Número de ciclos de retardo por ejecución en el que solo entran en juego las ráfagas de mayor prioridad.
discretRegisterOccupation	Indica si los registros que almacenan la ocupación de la longitud de onda de salida a lo largo del tiempo se implementan como máscaras de bits o no.
microsecPerBitInRegisterOccupation	Indica el número de microsegundos que le corresponden a un bit en el caso de tener máscaras de bits en los registros de ocupación

Mensajes generados

ID_SIGNALTOTRIGGERALGORITHM	Es un evento que indica cuando tiene que ejecutarse la siguiente iteración del algoritmo PI-OBS.

Mensajes consumidos

BurstHeaderPacket	Recibe un BHP de las fibras ópticas de entrada y lo planifica para enviarlo al siguiente nodo en el camino del LSP tras aplicar el algoritmo PI-OBS.

Implementación de las funciones declaradas en la interfaz

virtual void *handleMessage_BHP(BurstHeaderPacket *bhp)*

Esta función recibe los BHPs que provienen de las fibras ópticas de entrada y los eventos encargados de despertar al *scheduler* para realizar una ejecución del algoritmo. Si lo que se recibe es un BHP, este se va a almacenar hasta que se inicie la siguiente ejecución del algoritmo. Si lo que se produce es un evento para que se inicie una nueva ejecución, enyonces se van a procesar todas las BHPs como se explica en el Capítulo 3.

int arrivalsAndAlgorithmExecutionTimeCheck()

Esta función se encarga de comprobar que se cumplen las restricciones temporales de llegada de BHPs al *scheduler*.

void algorithmInitializationInStartUp()

Lleva a cabo el proceso de inicialización previo a cada ejecución. En él, se actualizan los valores de las variables y de los punteros correspondientes.

Int testChanges()

Comprueba el número de cambios que se han producido entre cada dos ciclos de retardo consecutivos dentro de una misma ejecución con el fin de terminar la ejecución cuando no se produzcan cambios y no tener que llegar al límite de número máximo de ciclos de retardo por ejecución. Devuelve el número de cambios que se han realizado.

Int algorithmIteration()

Es la función encargada de realizar una iteración del algoritmo PI-OBS. Se encarga de realizar las fases *request*, *grant* y *accept*. Devuelve el número de cambios en la última iteración.

double checkOverlapLAUCVF(double packetHeadLeaveTime, double packetOpticalLength, multimap<double,GMPLSOpticalDataPacket *> *thisIteration_packetMapThisWavelengthPerTailLeaveTime, multimap<double,GMPLSOpticalDataPacket *> *packetMapThisWavelengthPerTailLeaveTime)

Comprueba si el paquete se solapa con los paquetes que ya se han asignado en las ejecuciones previas del algoritmo o en asignaciones previas en la misma iteración. Devuelve el hueco generado si no hay solapamiento.

void newOpticalPacketInIngressQueue(int targetOutputFiber, int packetPSC)

Obtiene del *ingress* la ráfaga cuya fibra óptica de salida es la indicada en targetOutputFiber y con la clase que determina packetPSC.

void request(int delayCycle, int numberPSC)

Realiza la fase *request* en el ciclo de retardo indicado.

void grant_accept(int delayCycle, list <InputModuleInformation *> *listOfInputModulesWithAcceptThisDelayCycle)

Realiza las fases de *grant* y *accept* en el ciclo de retardo que se le indica.

void update(int delayCycle, list <InputModuleInformation *> *listOfInputModulesWithAcceptThisDelayCycle)

Realiza el mapeo de los registros de ocupación a la salida a partir de los módulos de entrada que han recibido un *accept* en este ciclo de retardo.

void movePointers(int *inputBlock, int *inputFiber, int *inputWavelength, int inputFiberFirst, int inputWavelengthFirst, bool clockwise)

Actualiza el estado de los punteros que en cada iteración se comience por un bloque de entrada distinto.

void algorithmInitializationPriorDelayCycle()

Inicialización previa a la ejecución del ciclo de retardo. En ella se reinicia los valores almacenados en los módulos de salida.

void updateStatistics_packetDecision(GMPLSOpticalDataPacket *packet, int

fdlIdAssigned)

Se encarga de actualizar los histogramas tanto si se encuentra hueco a la ráfaga como si se descarta.

virtual void endTransitoryTime ( )

Ponemos a cero todos los estadísticos que después se van a almacenar cuando se llame a la función endSimulationSample.

virtual void endSimulationSample (int sampleNumber)

Se almacenan los estadísticos en un determinado fichero

updateStatistics_packetDecision(BurstHeaderPacket *bhp, int fdlIdAssigned)

Esta función se encarga de actualizar los histogramas tanto si se envía una ráfaga de forma correcta como si se descarta en el nodo.

virtual void finish ( )

En esta implementación del Scheduler no realiza nada.

2.2.2 Descripción de los Scripts

2.2.2.1 Script multiNode_writeIniNed_OBS

Funcionalidad

Este *script* de Matlab se encarga de generar los ficheros Omnet.ini y Sistema.NED para realizar simulaciones de red OBS multinodo para una topología de red NSFNET [8] (ver apartado 4.2.2). Va a calcular tanto el enrutamiento como el tráfico necesario en cada enlace para cumplir con la carga objetivo de los enlaces.

Funciones que emplea el script

libraryFA_integerRoutingMinimaxUtilization_v1

Esta función se encarga de resolver el enrutamiento de cada LSP y la proporción de tráfico que se incluye en cada LSP. Para ello se calcula el problema MILP (*Mixed Integer Linear Programming*) mostrado en el apartado 4.2.3. La formulación MILP se ha solucionado en una plataforma TOMLAB/CPLEX [9].

calculateLLSPs_SPF
Esta función se encarga de normalizar la utilización de toda la red para que la utilización en el enlace más cargado de la red coincida con un parámetro que indica la máxima utilización objetivo. Para ello, parte de los resultados obtenidos en el proceso de optimización, quedándose con las rutas que tienen al menos un nodo a lo largo de su ruta. Una vez que se obtiene la carga en todos los enlaces tras la optimización, se normaliza la matriz de tráfico por el factor que hace que la utilización en el enlace más cargado de la red no exceda un determinado valor.

generateTopology_NSFNET
Esta función genera las tablas que representan la topología de red NSFNET. A partir de la matriz de tráfico y la matriz de distancias de los enlaces genera una tabla donde cada entrada tiene un nodo origen, un nodo destino del enlace, la velocidad de propagación en microsegundos y el número de longitudes de onda de ese enlace.

recalculateNumberWavelengthsEdgeLinks
Esta función se encarga de suavizar que sale del *edge node* y que entra a la red OBS. Para ello, recalcula el número de longitudes de onda que tiene que tener el enlace que une el *edge node* con el primer nodo *traversing* de la red para que la utilización de ese enlace coincida con la utilización máxima objetivo de la red.

checkUtilizationPerLink
Se encarga de comprobar la utilización de cada uno de los enlaces de la red NSFNET después de haber realizado el proceso de optimización y después de normalizar la carga en el enlace más cargado al valor indicado por la carga objetivo de la red.

2.2.2.2 Script monoNode_writeIniNed_OBS

Funcionalidad

Este *script* de Matlab se encarga de generar los ficheros Omnet.ini y Sistema.NED para realizar simulaciones de red OBS mononodo para la topología de red que se muestra en el apartado 3.3. La idea es tener un marco común donde se pueda cambiar de forma sencilla el *scheduler* del nodo central para ver como varían los resultados.

Funciones que emplea el script

writeRun_SourcesPart
Esta función se encarga de escribir la parte de los nodos fuente encargados de inyectarle el tráfico al nodo *switch*.

writeRun_SwitchPart
Esta función se encarga de implementar el nodo *switch*. Se van a pasar al Omnet.ini todos los

parámetros que necesita la parte *switch*.

writeRun_SinkPart

Esta función se encarga de implementar los nodos sumidero. Se van a pasar al Omnet.ini todos los parámetros que necesita la parte encargada de recibir las ráfagas..

writeRun_GlobalStatPart

Esta función se encarga de calcular los tiempos de simulación y los transitorios para cada punto de simulación.

Capítulo 3
PI-OBS: Parallel Iterative Optical Burst Scheduler

Este capítulo, describe el trabajo realizado dentro del proyecto, para la definición y evaluación del algoritmo PI-OBS (*Parallel Iterative Optical Burst Scheduler*). Este trabajo ha dado lugar a la publicación [11].

3.1 Introducción

En *Optical Burst Switching* (OBS) [1] [13], el tráfico en formato electrónico se ensambla en ráfagas ópticas de longitud variable, las cuales se inyectan en la red OBS a través de los *edge nodes* y de forma transparente, se enrrutan cruzando la red OBS hasta el *edge node* destino de la ráfaga.

Como se explicó en el apartado 1.2.2, existe un protocolo de reserva que se denomina *Just Enough Time* (JET). Este protocolo selecciona el offset lo suficientemente grande para tener en cuenta el tiempo de procesamiento de un paquete de control en los nodos intermedios y en el nodo destino. Cada paquete de control contiene información sobre el tamaño de su offset y la duración del *payload.* La cabecera al llegar a un nodo intenta reservar recursos para el tiempo correspondiente entre el comienzo y el final del *payload*. Si no existen suficientes recursos, entonces la ráfaga se descarta.

Debido a las restricciones temporales del protocolo de señalización JET usado para la reserva, los algoritmos de *scheduling* en OBS están afectados por una inevitable restricción en la respuesta temporal. El camino interno en el OSF debe de estar preparado en el momento de la llegada del *payload*. Garantizar un límite superior a la respuesta temporal del *scheduling* es un deber incluso en condiciones de tráfico de pico. Los enfoques secuenciales procesan las cabeceras de entrada una a una, sufren de una peor respuesta temporal la cual se incrementa al aumentar el número de puertos de entrada. Es por ello que estos enfoques secuenciales no son factibles para OSFs de media o gran escala. Además, estos enfoques son inherentemente codiciosos. Como regla general, se prefiere realizar asignaciones de recursos conjuntas involucrando más de una ráfaga buscando un mejor rendimiento medio.

Este trabajo propone un algoritmo de *scheduling* para nodos OBS, denominado PI-OBS (*Parallel Matching Optical Burst Scheduler*). PI-OBS es adecuado para una implementación rápida, paralela e iterativa con un tiempo de respuesta que es independiente del tamaño del *switch*. En este trabajo se aplican conceptos de *scheduling* presentes en el iSLIP-*like parallel-iterative scheduling algorithm* [4] diseñado para arquitecturas *Virtual Output Queuing* (VOQ). PI-OBS de forma conjunta procesa las cabeceras de las ráfagas que llegan en una determinada ventana, usando las técnicas de *void filling*. También tiene capacidades de diferenciación en la pérdida de ráfaga de acuerdo con la información en la cabecera de la ráfaga.

3.2 Descripción del algoritmo PI-OBS

3.2.1 Vista general y restricciones temporales

El algoritmo PI-OBS se ha diseñado como un algoritmo paralelo e iterativo, que es capaz de garantizar un límite superior al tiempo de respuesta del *scheduler*. Vamos a denotar este tiempo de respuesta como T_A µs. El algoritmo se ejecuta periódicamente cada T_I µs. La restricción de que $T_I \geq T_A$ asegura que una ejecución del algoritmo empieza estrictamente después de que la ejecución anterior haya finalizado. La ejecución del algoritmo que comienza en un tiempo $t = t_0$, es responsable de realizar de forma conjuntamente el procesamiento de todas las BHPs recividas de forma asíncrona durante el intervalo $[t_0\text{-}T_I,t_0]$. Este intervalo se denomina *header arrival time window* de la ejecución del algoritmo. Después de la ejecución del algoritmo, las decisiones de *scheduling* hechas para todas las cabeceras procesadas son almacenadas en el sistema para la correcta reconfiguración del OSF para el momento en el que las ráfagas de datos lleguen al OSF. En resumen, el tiempo de operación ranurado está relacionado únicamente con el funcionamiento del *scheduler*, mientras que el *data plane* opera de forma asíncrona. Esto es posible gracias al esquema de reserva avanzado de OBS (explicados en la sección 1.2.2): las decisiones tomadas sobre las cabeceras, solo se despliegan, en términos de reconfiguración de *switch*, cuando el correspondiente *payload* llega a la matriz de conmutación.

El sistema se diseña para garantizar que un camino en el OSF está listo cuando llega el *payload*. Denotamos como T_{WC} al tiempo en el peor caso (µs) entre el instante de la recepción del BHP y el momento en el cual el camino se encuentra disponible para la llegada del *payload*. El T_{WC} es la suma de tres tiempos: (i) T_I como el tiempo del peor caso desde la llegada del BHP hasta la ejecución del algoritmo (correspondiente a un BHP recibido justo en el comienzo de la ventana de llegada de cabeceras), (ii) el tiempo de respuesta del algoritmo T_A y (iii) el tiempo de reconfiguración de los componentes ópticos del OSF T_0.

$$T_{WC} = T_I + T_A + T_0 \qquad \text{Ec 1}$$

Para una BHP que llega en un tiempo $t=t_0$, se debe satisfacer que el *payload* entre en el OSF al menos en un tiempo $t > t_0 + T_{wc}$.Para cumplir con esta restricción, existen dos parámetros que se pueden sintoniza: (i) tiempo de offset mínimo δ_m entre el BHP y el *payload* y (ii) es un tiempo extra de retardo D_p añadido en el camino del *payload*, implementado por FDLs de duración fija en los puertos de entrada (ver Fig. 1). Hay que tener en cuenta que aunque este enfoque es más usado comúnmente en arquitecturas OPS (*Optical Packet Switching*), también es factible en este caso. Como conclusión, la siguiente restricción se tiene que mantener:

$$\delta_m + D_p \geq T_{WC} = T_I + T_A + T_O \qquad \text{Ec 2}$$

Existe una implicación adicional en la operación de *scheduling* al no tener un tiempo de reconfiguración de los componentes ópticos despreciable. Durante el tiempo de reconfiguración de un camino interno en el OSF, los componentes ópticos involucrados no se pueden emplear. Es por ello que si el tiempo ocioso entre la llegada de dos *payloads* consecutivos al mismo puerto de entrada fuera menor que T_O entonces la segunda ráfaga sería destruida por la reconfiguración de los componentes ópticos.

Como conclusión hay que destacar que se necesita un acuerdo de toda la red para definir lo que se denomina mínimo *inter-burst gap* (IBG) de la misma forma que esto ocurre con el *inter-frame gap* en redes Ethernet. El tiempo de reconfiguración óptica de los equipos OBS debe de ser menor (no está estandarizado aún) que el tiempo IBG. Los *schedulers* OBS deben garantizar que dos *payloads* asignados a una longitud de onda de salida en cualquier enlace se encuentran separadas la cabeza de la cola por un tiempo de cómo mínimo el IBG.

Después de que el algoritmo se haya completado, las ráfagas a las cuales se les ha asignado un retardo y una longitud de onda, propagan sus correspondientes BHP con la información actualizada de offset. De forma contraria, una ráfaga que se ha descartado, no propaga su BHP. Algunas propuestas se han presentado para aplicar estrategias OBS preventivas las cuales pueden permitir que las cabeceras recibidas en el futuro, adelanten a ráfagas que ya se han planificado previamente (normalmente asociadas a ráfagas de tráfico de una clase mayor). Este tipo de estrategias también se pueden aplicar al algoritmo PI-OBS. Sin embargo, estos enfoques involucran una carga en la señalización si la ráfaga adelantada ya ha propagado su cabecera ya que el nodo siguiente debe ser actualizado con el adelantamiento. Es por esto por lo que estas técnicas no se van a estudiar en este trabajo fin de máster.

3.2.2 Arquitectura del Scheduler

La Fig. 10 muestra los principales bloques de la arquitectura propuesta para el algoritmo. Este está basado en la interconexión de *nNH* módulos de entrada (lado izquierdo de la figura), y *nN* módulos de salida (lado derecho de la imagen) conectados por medio de una matriz de interconexión para realizar la señalización entre los módulos de entrada y de salida.

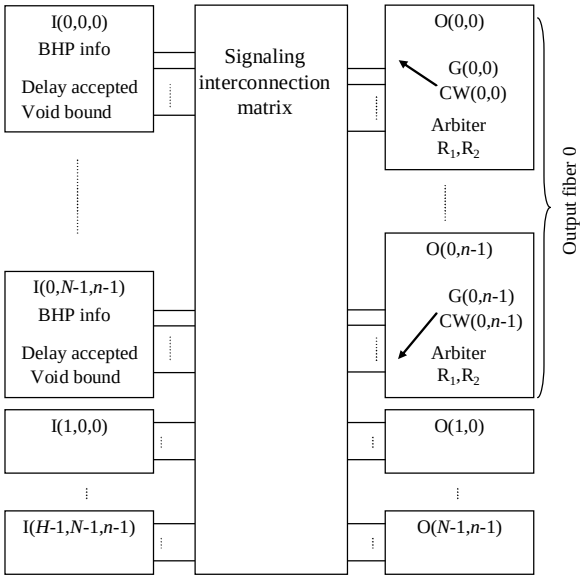

Fig. 10. Arquitectura del scheduler

Vamos a explicar las funcionalidades de los módulos de entrada y de salida.

1. Descripción de los módulos de entrada

   Un módulo de entrada I_{hfw} existe por cada bloque de horizonte temporal *(h=0,…H-1)* , cada fibra de entrada (*f=0,…,N-1*), y cada longitud de onda de entrada (*w=0,…,n-1*). Los horizontes en este contexto son los intervalos consecutivos de duración T_I en los cuales organizamos las futuras llegadas de *payloads*. Durante la ejecución del algoritmo que comienza en un tiempo $t=t_0$, un módulo de entrada I_{hfw} contiene la información sobre el *payload* cuyo primer bit llegará al OSF a través de la fibra de entrada *f,* la longitud de onda de entrada w, dentro del intervalo de tiempo $[t_0-T_I+T_{WC}+hT_I, t_0-T_I+T_{WC}+(h-1)T_I]$. El jorizonte más cercano en tiempo corresponde con los *payloads* que llegan al OSF en el rango $[t_0-T_I+T_{WC},\ t_0+T_{WC})$. El tiempo $t_0-T_I+T_{WC}$ es el tiempo más temprano de llegada de un payload cuya cabecera fue recibida en el comienzo de la actual ventana de llegada de cabeceras, $t=t_0-T_I$. El número de horizontes temporales a considerar *H*, depende de la diferencia entre el offset máximo y mínimo permitidos en el sistema. Se debe de garantizar que como máximo un *payload* se asocia a cada módulo de entrada en una ejecución del algoritmo. La razón

es que cada módulo de entrada del *scheduler* es capaz de manejar como mucho una llegada de un *payload*. Esto implica que la longitud mínima permitida para el *payload* (L_{min}) más el IBG debe ser mayor que el periodo de ejecución del algoritmo T_l .

$L_{min} + IBG > T_l$ Ec 3

La información almacenada en cada módulo de entrada es (i) la fibra óptica de salida asociada a la ráfaga, el offset de la ráfaga, la longitud de la ráfaga y la clase de QoS, (ii) la información sobre la asignación de la ráfaga que se actualiza durante la las iteraciones del algoritmo: FDL y longitud de onda de salida asignada, y un límite superior a la longitud del hueco generado desde la cabecera de la ráfaga a la cola de la ráfaga previa si la asignación se realiza. La naturaleza del cálculo de la longitud de este hueco se describe más adelante en esta sección.

2. Descripción de los módulos de salida

   Existe un módulo de salida O_{fw} por cada fibra de salida $f=0,\ldots,N-1$, y por cada longitud de onda de salida $w=0,\ldots,n-1$. Cada módulo de salida contiene cuantro registros de control: (i e ii) dos registros que almacenan la ocupación de la longitud de onda de salida w a lo largo del tiempo: $R_1(f,w)$ y $R_2(f,w)$, (iii) un registro $G(f,w)$ que almacena un puntero *grant*, de longitud $\log_2(Nnh)$ bits, y (iv) un registro de un bit $CW(f,w)$ que establece la dirección de exploración del puntero *grant*. La utilización de estos registros se explicará más claramente a continuación.

3.2.3 Descripción del algoritmo de Scheduler

Como se ha mencionado en los apartados anteriores, una ejecución del algoritmo comienza periódicamente cada T_l µs. Vamos a suponer que la ejecución del algoritmo actual comienza en un tiempo $t=t_0$. En este momento, los módulos de entrada I_{hfw}, con h=0,..,H-1, f=0,...,N-1, w=0,...,n-1, contienen la información de control de los BHPs que han llegado dentro de la ventana $[t_0-T_l, t_0)$. En cada módulo de salida, los registros R_1 y R_2 contienen la misma información: la ocupación de las longitudes de onda de salida con los *payloads* planificados en las iteraciones previas del algoritmo, las cuales se pueden solapar aún con los *payloads* que han llegado. R_1 actúa como copia de *backup* de R_2 durante la ejecución del algoritmo.

Cada ejecución del algoritmo se compone de una secuencia de CI iteraciones. Cada iteración se compone de una secuencia de D ciclos de retardo, uno por cada FDL en el OSF. Cada ciclo de retardo i se dedica a encontrar una longitud de onda para asignarla a los *payloads* que llegan si estos se retardasen mediante la correspondiente FDL i, $i=0,\ldots,D-1$. Cada ciclo de retardo consiste en una secuencia de cuatro pasos: (i) *request* , (ii) *grant*, (iii) *accept*, (iv) *update*.

Las operaciones envueltas en los ciclos de retardo de la primera iteración del algoritmo son:

- Paso *Request* (para los ciclos de retardo $0,\ldots,D-1$): Se ejecuta en paralelo en cada uno de los nNH módulos de entrada. Para cada módulo i con un *payload* destinado a la fibra de salida f, una señal de *request* es enviada a los módulos de salida asociados con todas las longitudes de onda de salida de la fibra óptica de salida objetivo: O_{fw}, w=0,...,n-1. Después de la señal *request*, la información sobre el tiempo de llegada del *payload*, duración del *payload*, y QoS del *payload* se transmite también a través de la señalización de la matriz de interconexión.
- Paso *Grant* (para los ciclos de retardo $0,\ldots,D-1$). Esto se ejecuta en paralelo, en cada uno de los módulos de salida nN. Las señales de *request* recibidas desde los módulos de entrada se escanean, empezando por el módulo de entrada perteneciente al horizonte h, fibra de salida f y longitud de onda de entrada w apuntada por el puntero *grant* $G(f,w)$ del módulo de salida. Internamente, el orden de escaneo del resto de las 3-tuplas (h,f,w) continúa de forma lexicográfica: un módulo de salida (h_1,f_1,w_1) se escanea antes que un módulo de entrada (h_2,f_2,w_2) si h_1 es más cercano a h que h_2 en el sentido de las agujas del

reloj dependiendo del estado del registro de bit *CW(f,w)*. Si $h_1=h_2$, las 3-tuplas son ordenadas de acuerdo al mismo tipo de distancia desde f_1 y f_2 hasta *f*. Si f_1 y f_2 son iguales entonces los módulos de entrada son ordenados de acuerdo a la distancia desde w_1 y w_2 hasta *w*. Aunque es difícil de describir, estas operaciones se pueden realizar por medio de árbitros implementados con circuitos combinacionales. Se envía un *grant* al primer módulo de entrada encontrado cuya ráfaga no se solape ni con las ráfagas existentes planificadas en las iteraciones previas del algoritmo ni con las ráfagas asignadas en los ciclos de retardo previos de la misma iteración del algoritmo. El árbitro da prioridad a los módulos de entrada con mayor clase de QoS, antes que a la posición del módulo de entrada. La información para el control del solape en la asignación se almacena en el registro R_2 . Las diferentes disposiciones de la información en los registros R_1 y R_2 puede conducir a una compensación entre tiempo de respuesta y complejidad en la implementación electrónica. Por ejemplo, si los registros R_1 y R_2 se implementan como máscaras de bits, cada bit representa un pequeño intervalo de tiempo de la duración de la ocupación, por lo que la comprobación del solape se simplifica a una sencilla y rápida operación AND en paralelo. Además, la comprobación del solape en distintos ciclos de retardo, se realiza de forma sencilla desplazando el registro R_2 antes de la comprobación. La señal *grant* (si es que hay alguna) se transmite a través de la matriz de interconexión. Después de esto, la información sobre el hueco generado por este *grant*, se transmite también: la distancia temporal entre la cabeza del *payload* al que se le está buscando hueco y la cola del *payload* anterior, de acuerdo a la actual asignación. Esto se calcula de forma sencilla como un producto de la operación de comprobación de solape. Hay que darse cuenta de que el hueco verdadero, se puede disminuir si en los siguientes ciclos de retardo, a una ráfaga se le asigna llenar el hueco entre la ráfaga actual que está buscando hueco y la cola de la ráfaga que le precede.

- Paso *Accept* (para los ciclos de retardo *0,…, D-1*). Este paso se ejecuta en paralelo en cada uno de los *nNH* módulos de entrada. De todas las señales *grant* que se reciben, se selecciona aquella que tiene un hueco menor. Si esto lleva a tener más de un *grant* posible, aquel con el menor índice asociado a la longitud de onda es el que se elige (first-fit). Una vez realizado esto, se envía una señal *accept* al módulo de salida asociado. Hay que destacar que (i) solo los módulos de entrada que envían un *request* pueden recibir un *grant*, (ii) después de enviar una señal de *accept*, el módulo de entrada no entra en juego en los siguientes ciclos de retardo de la misma iteración del algoritmo.
- Paso *Update* (para los ciclos de retardo *0,…,D-2*): Se ejecuta en paralelo, en cada uno de los nN módulos de salida. El registro interno R_2 que almacena la ocupación de la longitud de onda de salida a lo largo del tiempo, se actualiza con la información de las nuevas señales *accept*, asi que las futuras asignaciones en diferentes retardos de la misma iteración, no se van a solapar con la asignación que ya se ha aceptado.

Una vez que una iteración ha terminado, todas las asignaciones realizadas se borran. La ocupación en el registro R_2 se pone de nuevo a ser R_1 : el sistema es reiniciado al estado previo a la primera iteración en esta ejecución del algoritmo. La única información que permanece en el sistema de una iteración a la siguiente, se almacena en los módulos de entrada: cada módulo de entrada recuerda el retardo y el hueco asociado a la asignación aceptada en las iteraciones previas, si es que hay alguna. Esta información se usará en los pasos *request* y *grant* de la siguiente iteración.

Las acciones que se realizan durante los pasos *request* de las iteraciones 2,…, C_I se modifican de la siguiente manera. Vamos a suponer un módulo de entrada *(h.f.w)* el cual aceptó una asignación para el retardo D_1 en la iteración previa, con un límite de hueco V_1. En los siguientes ciclos de retardo de la iteración $d=0,\ldots,D_1-1$, se realiza la operación normal. Si una señal *accept* es enviada, el módulo de entrada rechaza enviar más señales *request* en los siguientes ciclos de retardo. Durante el ciclo de retardo D_1 (si el módulo no ha recibido un *grant* aún), la señal *request* se acompaña con el tamaño V_1 de la información del límite del hueco almacenado en el módulo de entrada, el cual es enviado a los módulos de salida a los que se les va a hacer un *request*.

El paso *grant* se modifica de la siguiente manera. Cada módulo de salida envía un *grant* el primer módulo de entrada siguiendo el siguiente orden de escaneo, para los que (i) un *request* se ha recibido, y (ii) el hueco generado por esta asignación (comprobado en el registro R_2) es estrictamente menor que el hueco V_1 publicado por dicho módulo de entrada. De nuevo,

esta funcionalidad se puede implementar de forma rápida por comparaciones binarias realizadas en paralelo en todos los módulos de salida. Hay que tener en cuenta que la comparación de los huecos implica que los *grants* enviados en la iteración *i* hacia un módulo de entrada, puede ser enviada hacia otros módulos en la iteración *i+1*.

Al final de la última iteración, las asignaciones aceptadas por los módulos de entrada son consideradas finales. En los módulos de entrada, el registro R_2 en cada módulo de salida contiene la ocupación actualizada. Esta información es copiada al registro R_1. Antes de que la siguiente iteración del algoritmo comience, los registros R_1 se modifican para reflejar la propagación de un paquete de T_l µs. Si R_1 y R_2 son los registros de máscaras de bit, que se pueden implementar por medio de operaciones de desplazamiento.

3.2.4 Operación de los punteros Grant e inicialización del sistema

Como pasa en el *scheduler* iSLIP para conmutadores VOQ, la operación de los punteros *grant* afecta de forma importante al rendimiento del sistema. Si un módulo de entrada entra dentro de un paso *request* , de forma simultánea envía una señal *request* a los *n* módulos de salida, una por cada longitud de onda de salida de la fibra de salida objetivo. Esto es interesante para reducir el número de *grants* simultáneos que un módulo de entrada recibe, ya que como mucho se puede aceptar un *grant*. Los *grants* que no se han aceptado corresponden a asignaciones de retardo que no han enviado *grant* a otros módulos. Estas asignaciones no entrarán en juego hasta la siguiente iteración. Es por esto que los punteros *grant* de los módulos de salida correspondientes a la misma fibra de salida deben ser desincronizados, en el sentido de que ellos apuntan a los módulos de entrada tan separados como sea posible los unos de los otros en el orden lexicográfico. Por ello, incrementamos las posibilidades de que los *grants* se distribuyan de forma más uniforme entre los módulos de entrada. Esto se puede obtener por medio de lo siguiente: (i) la inicialización de un puntero *grant* durante la puesta en marcha del sistema el cual maximiza la mínima distancia lexicográfica entre punteros, (ii) el bit *CW* se cambia después de cada ejecución del algoritmo, conmutando la dirección de barrido de los puertos de entrada apuntados por los punteros *grant*. Esta acción tiene como objetivo el mejorar la imparcialidad del sistema cuando las llegadas entre paquetes no son uniformes entre las fibras de entrada, (iii) cada dos ejecuciones del algoritmo (con direcciones de barrido opuestas) todos los punteros incrementan su valor por uno, módulo *nNH*. Como todos los punteros actualizan sus valores de forma simultánea, la desincronización de los punteros se mantiene.

3.2.5 Convergencia del algoritmo

Algunas de las propiedades de las que se va a hablar en este apartado, serán probadas de forma intuitiva.

El hecho de que los módulos de entrada recuerden en la iteración *i* la FDL aceptada y el hueco generado en la iteración *i-1* , es la idea principal del método, el cual permite mejorar las asignaciones en las siguientes iteraciones. Al mismo tiempo, se debe de comprobar que el algoritmo converge y que las variaciones en la asignación no permanecen eternamente. Vamos a definir el tiempo de convergencia del algoritmo como el número de iteraciones necesarias para que el sistema alcance una asignación estable para todas las ráfagas procesadas, la cual no cambiaría si se realizasen un mayor número de iteraciones.

Vamos a ver dos propiedades de la convergencia del algoritmo con sus correspondientes demostraciones:

Propiedad 1: El tiempo de convergencia está limitado por un límite finito.

Demostración: Vamos a considerar un módulo de entrada I_{hfw} el cuál ha recibido un *grant* para un retardo D_1 en la iteración *i*. En la iteración *i+1*, el mismo módulo de entrada puede recibir una asignación diferente de retardo y longitud de onda, solo cuando (i) las asignaciones en los ciclos de retardo $0,\ldots,D_1-1$ han cambiado desde la iteración previa o (ii) el límite del hueco generado anunciado en el ciclo de retardo D_1 cambia en cualquier módulo de entrada.

Para el ciclo de retardo 0, solo la condición (ii) se puede mantener, y una variación en (ii) puede suceder únicamente cuando un módulo de entrada ha mejorado su estimación del hueco. Por tanto, esto solo puede suceder durante un número finito de iteraciones. Después de esto, las asignaciones no cambian en el ciclo de retardo 0. En este momento, aplicando el mismo principio al ciclo de retardo 1, después al 2, etc, la convergencia está garantizada en un número finito de iteraciones.

El algoritmo PI-OBS dirige el problema de optimización multiobjetivo de asignar retardos y longitudes de onda de salida a las ráfagas de entrada de modo que: (i) el número de ráfagas que reciben un retardo se maximiza para cada clase de QoS, priorizando el tráfico de mayor clase, (ii) el retardo medio de la asignación se minimiza, (iii) el tamaño medio de los huecos generados se minimiza.

Propiedad 2: Las asignaciones estables del algoritmo PI-OBS son soluciones locales de distancia 1 al problema previo.

Demostración: Vamos a dar una demostración intuitiva. Una solución local de distancia 1 mínima significa que la asignación no se mejora por soluciones vecinas las cuales difieren como mucho en una asignación. Vamos a asumir una solución en la cual (i) una ráfaga más puede recibir un retardo y longitud de onda en lugar de ser descartado, o (ii) recibir un mejor retardo, (iii) o recibir una asignación con el mismo retardo pero implicando un menor hueco. Claramente, esta solución no sería una asignación estable, la convergencia no se ha alcanzado y el algoritmo cambiaría la solución en una iteración más lejana.

3.3 Resultados

La Fig 11 muestra el escenario de pruebas. El *switch* bajo evaluación S_E recibe tráfico desde los N nodos vecinos ($I_0,\ldots,I_{N-1}$) y es responsable de conmutar las ráfagas a los *N* nodos de salida objetivo ($T_0,\ldots,T_{N-1}$). Las fibras tienen *n* longitudes de onda $\lambda_1,\ldots,\lambda_n$ y una longitude de onda de control separada λ_0.

Vamos a evaluar tres algoritmos de *scheduling* diferentes en el nodo S_E : LAUC, LAUC-VF y PI-OBS. LAUC y LAUC-VF son algoritmos secuenciales que se emplean comúnmente como comparación para los límites de rendimiento.

El tiempo de reconfiguración de los equipos ópticos y el tiempo IBG se asume que es igual a 0.03 µs (T_O=IBG=0.03 µs). Los nodos de entrada y el nodo S_E bajo test respetan este tiempo IBG en sus asignaciones. Los algoritmos de *scheduling* pueden hacer que de forma artificial añadan el valor IBG la duración del *payload*. Es por esto que el *scheduler* garantiza que cada *payload* es seguido por un tiempo sin actividad de IBG µs en la longitud de onda de salida.

Cada nodo fuente ensambla ráfagas cuya duración sigue una distribución normal truncada. La longitud mínima de la ráfaga se establece en 10 µs, y la duración máxima de la ráfaga es de 100 µs. La duración media de la ráfaga se establece en 55 µs. El tiempo entre el ensamblado de dos ráfagas consecutivas están distribuidas exponencialmente. Esta media se calcula para que coincida con el valor de carga deseado. Después de que una ráfaga sea ensamblada, la longitud de onda de transmisión y el tiempo de inyección en la fibra de conexión son seleccionados como si el nodo de estrada fuera un nodo donde se implementa el *scheduler* LAUC-VF, con un número infinito de FDLs. Usar los nodos fuente con LAUC-VF, se considera un escenario más real, el cual intenta reproducir la correlación en la llegada de ráfagas que aparece en diferentes longitudes de onda de la misma fibra en una red OBS. La granularidad de las FDLs de los nodos fuente y del nodo S_E se hace igual a 55.03 µs, que es

igual al tamaño medio de ráfaga más el tiempo IBG. Vamos a denotar este tiempo como el tamaño medio de ráfaga percibido, como si este fuera el tamaño medio de ráfaga percibido por el *scheduler*. Se han realizado trabajos previos en los que se muestra que una granularidad de las FDLs cercana a la duración media del *payload* optimiza el rendimiento del sistema.

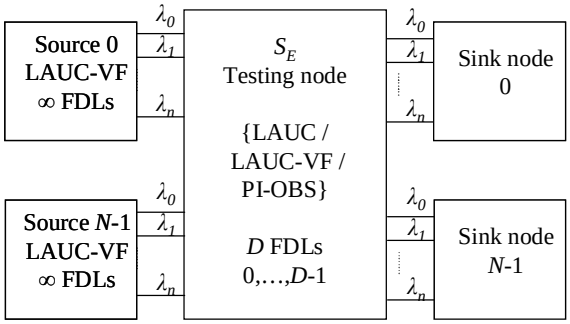

Fig. 11. Escenario de Prueba

El tiempo entre dos ejecuciones del algoritmo se establece a T_I = 10 µs. El tiempo de respuesta del algoritmo se asume que es también T_A=10 µs (con esto se cumple la restricción de que $T_I \geq T_A$). El tiempo de offset mínimo de las ráfagas generadas por el *edge node* se calcula asumiendo que el *switch* bajo test tiene D_P=0 µs extra de retardo de *payload.* Entonces δ_m=20.03 µs. Las ráfagas son generadas con un offset aleatorio distribuido uniformemente en el rango [20.03, 80.03] µs. Esto implica que el algoritmo PI-OBS emplea 7 horizontes de 10 µs cada uno. Para una ejecución del algoritmo que comienza en un tiempo $t=t_0$, el primer horizonte contiene los *payloads* que llegan al OSF en el intervalo temporal [t_0+10.03, t_0+20.03]. Hay que tener en cuenta que en el diseño de redes OBS, el tener un tiempo de offset constante provoca que el número de horizontes en el PI-OBS se reduciría a uno, resultando en un considerable ahorro de complejidad de implementación.

El nodo de salida objetivo se selecciona aleatoriamente con una distribución uniforme. Se han definido dos clases de servicio en los test: 10% de las ráfagas son de alta prioridad (*Hi*), y el 90% son de baja prioridad o tráfico *best-effort* (*BE*). Solo el algoritmo PI-OBS implementa diferenciación de tráfico.

La evaluación del rendimiento se ha realizado mediante una simulación por eventos discretos. La herramienta de simulación se ha construido sobre la plataforma OMNeT++. Todos los test que se han realizado consisten en 5 muestras independientes, con 10^7 ráfagas generadas en cada una. Los intervalos de confianza se calculan para tener un 95% de calidad, usando el método *t-Student*. Los intervalos de confianza obtenidos validan los resultados, pero no se muestran en las figuras por claridad.

El primer paso en nuestro estudio está dirigido al dimensionamiento del *buffering* en el *switch* S_E requerido para garantizar una probabilidad de pérdida de bit inferior a 10^{-5} para una carga de entrada del 80%. Medir la probabilidad de pérdida de bit significa que la pérdida de una ráfaga se pondera por su duración. La varianza de la duración del *payload* se hace igual a la longitud de ráfaga actualizada 55.03 µs. Esto corresponde a un coeficiente de variación del *payload* igual a 1, *CV=1*.

Los resultados se muestran en la Tabla I. Cabe destacar que los algoritmos PI-OBS y LAUC-VF tienen un resultado similar en cuanto a requerimiento de *buffering*. En el escenario DWDM (n=64), con 2 FDLs es suficiente para garantizar las pérdidas objetivo. También, los resultados confirman que el algoritmo LAUC incrementa de forma importante los requerimientos de *buffering* debido a su ineficiencia en el uso de los recursos. Los mismos requerimientos de *buffering*, que no se han mostrado en la tabla, se han obtenido para los coeficientes de variación *CV*=0.5 y *CV* = 1.5 con la única diferencia para el caso de n =16, *CV*=0.5, donde los *schedulers* LAUC-VF y PI-OBS requieren de una FDL extra.

Los siguientes test incluidos en este trabajo intentan suministrar una profunda comprensión de cómo los *schedulers* LAUC-VF y PI-OBS reaccionan a intervalos de sobrecarga. El *scheduler* LAUC se elimina de esta figura debido a su peor rendimiento.

TABLA I
NÚMERO DE FDLS PARA UNA PROBABILIDAD DE PÉRDIDA DE BIT 10-5, BAJO 80% DE CARGA. SCHEDULERS{PI-OBS/LAUC-VF/LAUC}

N \ λ	*n*= 16	*n* = 32	*n* = 64
N =2	4 / 4 / >10	3 / 2 / >10	2 / 2 / 4
N =4	4 / 4 / >10	3 / 3 / >10	2 / 2 / 4
N =8	4 / 4 / > 10	3 / 3 / >10	2 / 2 / 4

Las figuras Fig.8 y Fig.9 estudian la probabilidad de pérdida de ráfaga de un *switch* con *N*=4 fibras de entrada y de salida, y *n=16* longitudes de onda por fibra, bajo una carga de entrada del 95 %, pero con un dimensionamiento de *bufferin* de *D*=4 FDLs (calculado en la Tabla I para una probabilidad de pérdida de bit de 10^{-5} bajo un 80 % de carga). Naturalmente, este escenario sobrecargado se supone que es transitorio en un sistema actual, ya que la probabilidad de pérdida que se obtiene es inaceptable. Se han realizado tests para *switchs* de tamaño *N* = {2, 4, 8}, *n* = { 16, 32, 64 }, los cuales no se han mostrado en este trabajo pero que conducen a las mismas conclusiones que las expuestas a continuación.

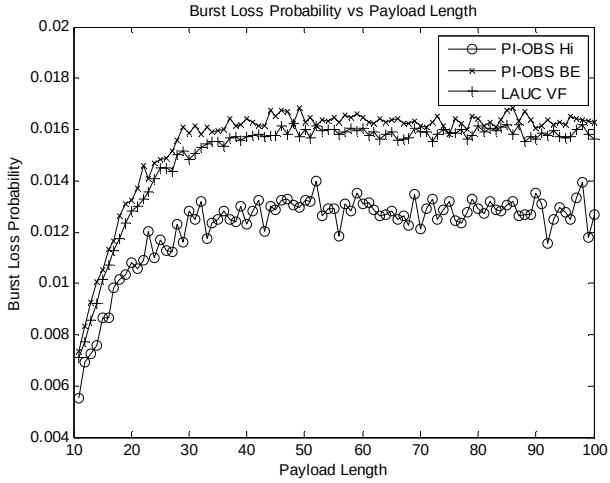

Fig. 12. Distribución de la probabilidad de pérdida de ráfaga dependiendo de la longitud del *payload*. Carga de entrada 95%, N=4, n=16, D=4

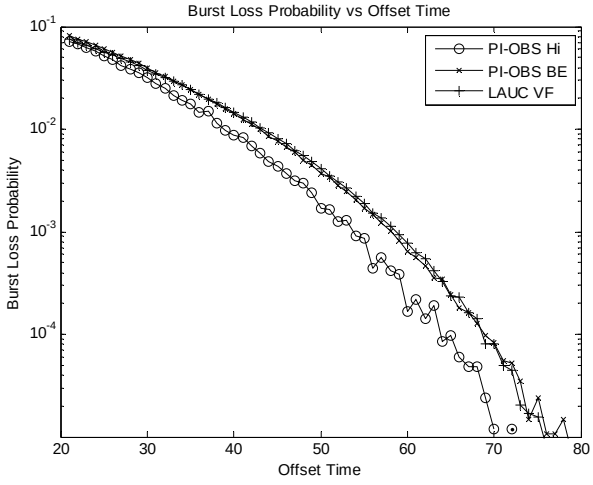

Fig. 13. Distribución de la probabilidad de pérdida de ráfaga dependiendo del offset del *payload*. Carga de entrada 95%, N=4, n=16, D=4

La figura Fig.12 muestra la distribución de la probabilidad de pérdida de ráfaga, dependiendo del tamaño del *payload*. Los resultados muestran las siguientes conclusiones: (i) Las ráfagas con un tamaño menor tienen una mayor oportunidad de que se le asignen los recursos. Todas las ráfagas de tamaño mayor que la mitad del tamaño medio de ráfaga son tratadas de igual forma por el sistema. Estos efectos aparecen por igual en los algoritmos PI-OBS y LAUC-VF. (ii) PI-OBS diferencia de forma correcta la probabilidad de pérdida de ráfaga para las dos clases de servicio probadas: clase alta (*Hi*), y *best effort* (*BE*). Sin embargo, hay margen en la investigación de variantes en los *scheduler* que obtienen una mayor diferenciación del tráfico (iii) las pérdidas del tráfico *best-effort* en el algoritmo PI-OBS son similares a las obtenidas con el algoritmo LAUC-VF. El rendimiento del tráfico de clase alta en el *scheduler* PI-OBS mejora los resultados de LAUC-VF.

La figura Fig.13 muestra la distribución de la probabilidad de pérdida de ráfaga como función del offset de la ráfaga. De nuevo, LAUC-VF y PI-OBS son similares en el impacto del offset de ráfaga en la probabilidad de pérdida para un escenario sobrecargado, incrementando la probabilidad de descarte para ráfagas que tienen un offset menor.

Los resultados observados para el algoritmo LAUC-VF son comparables a aquellos resultados obtenidos en otros trabajos los cuales han estudiado el rendimiento del *scheduler*, aunque bajo diferentes escenarios de prueba. Las conclusiones más importantes en nuestro estudio son que PI-OBS, el cual se diseñó para minimizar la misma función objetivo sin un enfoque codicioso, conlleva a un comportamiento similar bajo condiciones de sobrecarga.

Aunque la convergencia de PI-OBS ha sido probada en un número finito de iteraciones, es obvio que tiempos más cortos de convergencia conducirán a una simplificación hardware. La Tabla II resume la información de convergencia del algoritmo para PI-OBS en los mismos tests mostrados en la Tabla I. Para cada N={2,4,8}, n ={16,32,64}, incluimos el número de iteraciones del algoritmo para que se obtenga la solución optima en el 99% de los time slots, para los tres factores de *CV*, *CV*={0.5, 1,1.5}. Podemos concluir que el tiempo de convergencia no depende ni del número de fibras, ni del factor de *CV* que tengamos en la distribución de la longitud del *payload*, pero sí que es ligeramente mayor para el caso de tener un mayor número de longitudes de onda por fibra. Sin embargo, el número de iteraciones para la convergencia se puede considerar razonablemente bajo.

TABLA II
NÚMERO DE ITERACIONES EN PI-OBS QUE IMPLICA CONVERGENCIA EN EL 99% DE LAS EJECUCIONES DEL ALGORITMO. *CV*={0.5/1/1.5

N \ λ	n= 16	n = 32	n = 64
N =2	6 / 6 / 6	8 / 8 / 8	10 / 10 / 10
N =4	6 / 6 / 6	8 / 8 / 8	11 / 11 / 11
N =8	6 / 6 / 6	8 / 9 / 8	11 / 11 / 11

3.4 Implementación electrónica

La implementación electrónica de algoritmos paralelos e iterativos ideados para *schedulers* OBS no se ha abordado en la literatura abierta. El algoritmo PI-OBS en el primer enfoque en esta dirección. La idea principal está detrás de la implementación yace en el hecho en que PI-OBS procesa todas las BHPs en la ventana de llegada de forma simultánea. Es por esto que es posible concebir una implementación electrónica dirigida a completar una ejecución en el orden de el tamaño mínimo de ráfaga más el tiempo IBG (por ejemplo 10 µs puede ser un valor realístico). Por el contrario, los algoritmos que procesan BHPs una a una, suelen dirigir su tiempo de respuesta del algoritmo sobre dos órdenes de magnitud menor (alrededor del tiempo previo dividido por el número de puertos de entrada).

Los algoritmos paralelos e iterativos en *switches* VOQ, de los cuales se han obtenido los principales conceptos del algoritmo PI-OBS, obtienen un tiempo de ejecución inferior a los 10 ns. Es por esto que algunas de las operaciones que se realizan dentro del *scheduler* PI-OBS se pueden implementar mediante circuitos electrónicos secuenciales en lugar de mediante diseños combinacionales, explorando la relación entre el tiempo de respuesta y la complejidad de la implementación. Un paso del algoritmo donde este tiempo extra se puede intercambiar por una simplificación hardware es en la comprobación de solape en los módulos de salida. La investigación se está conduciendo a dimensionar los registros R_1 y R_2 de forma efectiva (implementados como registros de máscara de bits), explorando la relación entre la longitud de los registros y el rendimiento del sistema. Hay que tener en cuenta también que la complejidad de implementación del algoritmo PI-OBS se reduce en las redes OBS donde se emplea un offset determinístico por lo que el parámetro *H* se iguala a uno, disminuyendo el número de módulos de entrada en el sistema (*nNh*).

El estudio de viabilidad sobre la implementación del algoritmo PI-OBS, está fuera del objetivo de este proyecto. Esta tarea está siendo abordada actualmente por una colaboración entre el Grupo de Ingeniería Telemática de la UPCT y el DEIS-UNIBO de la Universidad de Bologna

3.5 Conclusiones y trabajos futuros

PI-OBS es la primera propuesta de un *scheduler* paralelo e iterativo para conmutadores OBS. En contra de los enfoques codiciosos convencionales, todas las cabeceras recibidas en una ventana temporal dada, son procesadas de forma conjunta. Esto abre un campo para una ganancia de rendimiento, cuando se compara con otros enfoques codiciosos como el enfoque LAUC-VF. También, la convergencia estudiada del algoritmo muestra un tiempo de respuesta que es aproximadamente independiente del tamaño del *switch*. Observando las similitudes con los *schedulers* VOQ, los autores de este trabajo están trabajando en una implementación práctica del *scheduler*, explorando la relación entre la complejidad de implementación y el tiempo de respuesta del algoritmo.

Los autores consideran el algoritmo PI-OBS como el primer paso. Se pueden explorar variaciones del algoritmo PI-OBS para aplicar nuevas estrategias de asignación de recursos de forma conjunta, consiguiendo mejoras en el rendimiento y/o simplificaciones hardware.

Capítulo 4
Evaluación del límite de prestaciones de *bufferless* OBS

Como parte del trabajo asociado a este Trabajo Fin de Máster, se ha abordado un estudio de evaluación, sobre las limitaciones intrínsecas del paradigma *bufferless* OBS, que penalizan esta opción como solución viable de conmutación en redes ópticas. Este estudio de evaluación ha dado lugar a una publicación que se encuentra en proceso de revisión [14].

4.1 Introducción

Desde sus orígenes hace una década, el paradigma *Optical Burst Switching* (OBS) ha atraído un enorme interés en el campo de la investigación. Los sucesivos refinamientos y estudios han sido conducidos en numerosos campos que crecieron en relación con la tecnología OBS. Para nombrar algunos: técnicas de ensamblado de ráfagas y su efecto en el rendimiento del sistema y en los patrones de tráfico resultantes, los protocolos de reserva y la variación del offset inicial y del offset salto a salto, o los algoritmos de *scheduling* para los nodos de conmutación OBS.

Este trabajo no es ni una propuesta de un nuevo algoritmo de *scheduling* para un nodo OBS, ni una nueva arquitectura de *switching*, ni un nuevo protocolo de reserva, etc. Lo que se ha intentado es de establecer un escenario de pruebas razonable donde la mayoría de parámetros que se pueden manipular en una red OBS se tienen en cuenta, buscando la combinación que optimiza el *throughput* de la red, garantizando unas pérdidas objetivo razonable. Con ello intentamos responder a la pregunta de ¿las redes *backbone* OBS sin *buffer* son una opción factible, desde el punto de vista de la solución de contenciones?

La habilidad de solucionar la contención sin *buffers* ópticos se ha reivindicado como una de las mejores ventajas del paradigma OBS antes que otros enfoques como las redes *Optical Packet Switching* (OPS). No obstante, los estudios existentes que apoyan esta afirmación, no son exhaustivos, y por lo tanto no son concluyentes. Probablemente, esto es debido al hecho de que construir un escenario de pruebas es una tarea que conlleva tomar decisiones no triviales. Por supuesto, no es posible probar la interacción de todas las combinaciones de *assemblers*, con todos los *schedulers* y arquitecturas de *switching*, topologías de red, decisiones de ingeniería de tráfico, etc. Hay que identificar con extremo cuidado el conjunto de parámetros y técnicas que van a tomar parte en el estudio de prestaciones. Este trabajo está dirigido en esta línea. Nuestro escenario de pruebas tiene el objetivo de encontrar un límite superior al rendimiento de las pérdidas de ráfagas que se puede alcanzar con unas suposiciones razonables en el paradigma OBS sin *buffer*. En otras palabras, las opciones de diseño relevantes en la redes han tomado a favor del rendimiento de la resolución de la contención en los nodos ópticos.

Desde el punto de vista de los autores, la contribución de este trabajo va enfocada en tres direcciones. Primero, la selección meticulosa de las suposiciones consideradas en el escenario de pruebas puede ser un marco de trabajo para futuros tests, donde algunas de las hipótesis se revisarán, o más patrones de topologías de tráfico etc serán incluídos. Segundo, en algunos aspectos, nuestros resultados muestran desviaciones de estudios previos, que merecen un análisis cuidadoso. Tercero, esperamos que hayamos suministrado al menos, una pequeña parte de la respuesta de nuestra cuestión objetivo.

4.2 Escenario de pruebas

Los siguientes subapartados detallan y justifican el escenario de pruebas. Cada esfuerzo se ha puesto en definir de forma precisa la red bajo test, para permitir que los resultados se puedan repetir.

4.2.1 Topología de red

Estamos interesados en una red *backbone* óptica, compuesta de nodos OBS de interconexión y de *edge nodes* OBS. Los nodos de interconexión están situados a lo largo de los centros de tráfico, en ciudades más o menos pobladas. Éstos se encuentran conectados en una topología *mesh* por medio de enlaces WDM (*Wavelength Division Multiplexing*) que pueden atravesar cientos de Km. Cada nodo de interconexión se encuentra conectado a un *edge node*. Los *edge nodes* tienen un equipamiento específico, encargados de ensamblar las ráfagas a partir de la demanda de tráfico electrónico, y inyectarlo en la red a través de los nodos de interconexión. Los *edge nodes* se supone que tienen las mismas instalaciones que sus nodos de interconexión asociados, y por tanto conectados a ello por medio de enlaces de corto tamaño WDM.

Los enlaces entre dos nodos de interconexión tienen una longitud de onda de control para transmitir los paquetes de control (*Burst Header Packet*), y un número dado de longitudes de onda de datos para propagar los *payloads* (igual en todos los nodos de interconexión). Esto corresponde con un mallado de longitudes de onda único, empleado a lo largo de la red, lo cual es un supuesto razonable en el *backbone*. Cada evaluación estudiada en redes OBS ha observado un gran impacto del número de longitudes de onda en los enlaces en el rendimiento de las pérdidas en la red. Es por ello, por lo que vamos a elegir el número de longitudes de onda en los enlaces de interconexión *W* como un parámetro sintonizable en nuestro escenario de pruebas.

Los enlaces entre los *edge nodes* y los nodos de interconexión tienen también una longitud de onda de control, por donde se transmiten los BHPs. El número de canales de datos usados en estos enlaces se prevé diferente de los enlaces entre los nodos de interconexión como se describirá en las siguientes subsecciones.

En este trabajo solo se emplea una topología de red de referencia: la bien conocida red NSFNET [8] compuesta por 14 nodos de interconexión (14 nodos *traversing*), donde cada uno de ellos se encuentra unido a un *edge node*. Esto hace que tengamos un total de 20 nodos en la red. Las distancias en Km entre dos nodos de interconexión se toman a partir de la distancia geográfica en la red NSFNET [8], y se muestra en la Tabla I en la posición triangular inferior. Los enlaces entre un *edge node* y su correspondiente nodo de interconexión se suponen que son de longitud 0 Km. La velocidad de propagación en las fibras se supone que es de 200,000 Km/s.

4.2.2 Demanda de tráfico

Con el fin de asumir un tráfico realista en la red, la demanda de tráfico considerada es la que obtenemos en los *papers* para la red NSFNET. Es una matriz de tráfico simétrica pero no uniforme cuyos valores se muestran en la matriz triangular superior de la Tabla III. El valor (*i,j*), *i,j*=1,…,14 corresponde con el tráfico en Erlangs generado por el *edge node i* y dirigido al *edge node j*., y viceversa, normalizado para que el valor máximo sea igual a 100. Esta matriz de tráfico se empleará como referencia para todos los tests. Por supuesto, para observar el rendimiento de la red bajo diferentes intensidades de tráfico, esta matriz de referencia se multiplicará por diferentes factores de normalización como se verá en las siguientes subsecciones.

TABLA III: MATRIZ DE TRÁFICO (ERLANGS) Y LONGITUD DE LOS ENLACES (KM)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0	71.66	28.49	74.56	77.62	43.82	57.08	61.53	45.46	51.93	39.57	52.37	39.93	55.48
2	1100	0	30.81	71.13	37.77	44.47	62.69	41.76	36.05	28.03	46.03	44.68	27.14	40.91
3	1600	0	0	73.26	40.76	62.43	24.98	58.81	29.40	62.54	55.13	37.96	41.08	30.13
4	0	600	0	0	35.02	36.22	35.10	48.11	52.98	50.39	40.68	26.43	22.57	59.05
5	0	1000	0	600	0	34.45	84.44	54.61	65.38	60.54	48.05	55.48	48.85	48.96
6	0	0	2000	0	1100	0	55.87	51.44	44.10	40.57	53.07	39.16	53.11	59.55
7	0	0	0	0	800	0	0	60.42	30.94	63.00	62.27	34.24	75.09	50.50
8	2800	0	0	0	0	0	700	0	63.35	87.83	79.15	63.66	71.34	87.22
9	0	0	0	0	0	0	0	700	0	49.38	70.37	86.08	76.37	35.23
10	0	0	0	0	0	1200	0	0	900	0	70.35	62.80	50.62	66.21
11	0	0	0	2400	0	0	0	0	0	0	0	100.0	68.83	45.78
12	0	0	0	0	0	0	0	0	500	0	800	0	87.52	80.54
13	0	0	0	0	0	2000	0	0	0	0	0	300	0	52.52
14	0	0	0	0	0	0	0	0	500	0	800	0	300	0

4.2.3 Ingeniería del tráfico para una utilización minimax

Se asume que tenemos un *control plane* GMPLS y para facilitar la lectura de este trabajo, vamos a emplear una terminología GMPLS. De acuerdo con esto, (i) las conexiones de tráfico de las capas superiores toman la forma de LSPs (*Label Switched Path*), los cuales se pueden establecer entre dos *edge nodes*, y (ii) los *edge nodes* actúan como OBS-LSRs (*Label Switched Router*).

Una consecuencia de asumir que el control de la red es GMPLS, es que todas las ráfagas que pertenecen al mismo LSP deben seguir la misma secuencia de saltos desde el nodo *ingress* hasta el *egress edge node.* Es por esto que las técnicas de *deflection routing* no se pueden considerar como una opción para solucionar la contención y por ello no se incluyen en este trabajo. Hay que tener en cuenta que haciendo esto, el orden de las ráfagas *end-to-end* se conserva en las redes OBS sin *buffer*.

Como en toda tecnología de conmutación de paquetes o de ráfagas, la forma en la que enrrutemos el tráfico a través de la red, afecta de forma importante a su rendimiento. Por ejemplo, usando el enrrutado *shortest path* en una topología *mesh* con una matriz de tráfico arbitraria como la que tenemos, puede conducir fácilmente a tener enlaces congestionados y enlaces infra utilizados. En nuestro estudio, estamos interesados en encontrar el mayor *throughput* de la red, que mantenga la probabilidad de pérdida en un valor aceptable. Esto coincide con el objetivo estándar de los operadores de red. El rendimiento de pérdidas en la red está dominado por la utilización media de los enlaces. De acuerdo a esto, una manera más favorable desde el punto de vista de la optimización de la resolución de la contención es encontrar el enrutamiento de cada LSP, para que la máxima utilización a través de los enlaces se minimice. La optimización minimax de la utilización de los enlaces es un hecho comúnmente empleado como técnica de ingeniería de tráfico en redes de conmutación de paquetes y de ráfagas. Una mayor mejora en la utilización minimax se obtienen al permitir que el tráfico entre dos *edge nodes* se pueda dividir en rutas diferentes, para obtener un mayor balanceamiento del tráfico. Pero el dividir los flujos va en contra del concepto de LSP. Así que con el fin de permitir un mayor *throughput* de la red, asumimos que se establecen dos LSPs entre cada par de *ege nodes*. El enrutamiento de cada LSP y la proporción de tráfico que se incluye en cada LSP se calcula solucionando el problema MILP (*Mixed Integer Linear Programming*) mostrado a continuación.

Dado:

- Topología de la red : Nodos y enlaces en la red
- *W:* Número de longitudes de onda en los enlaces de interconexión (supone que son iguales en todos los enlaces)
- T_{sd}: Matriz de tráfico base. Tráfico en Erlangs desde el *edge node s* hacia el

edge node d.

Encontrar:

- r_{sdie}: 1 si el *i-th* LSP (*i*={1,2}) entre el *edge node s* y el *edge node d*, atravesando el enlace *e* y 0 en otro caso. (1a)
- f_{sdie}: Erlangs de tráfico del *i-th* LSP (*i*={1,2}) entre el *edge node s* y el *edge node d*, que atraviesa el enlace *e*. (1b)
- u: Variable para la utilización minimax del enlace. (1c)

La función objetivo: min *u*. (1d)

Sujeto a las siguientes restricciones:

$$f_{sdie} \le T_{sd} \cdot r_{sdie}, \ \forall s,d \text{ edge node}, \ \forall i=\{1,2\}, \ \forall \text{enlace } e \ . \tag{1e}$$

$$\sum_{\substack{e \text{ outgoing} \\ \text{from node } n}} r_{sdie} - \sum_{\substack{e \text{ entering} \\ \text{node } n}} r_{sdie} = \begin{cases} 1 \ si \ s = n \\ -1 \ si \ d = n \\ 0 \ Otro \ caso \end{cases}, \tag{1f}$$

$\forall s,d$ *edge nodes,* $\forall n$ *edge* o *interconnection node,* $\forall i=\{1,2\}$

$$f_{sd1e} + f_{sd2e} = T_{sd}, \tag{1g}$$

$\forall s,d$ *edge nodes, e* link de salida del *edge node s*

$$\frac{1}{W} \sum_{\substack{s,d \text{ edge node} \\ i=\{1,2\}}} f_{sdie} \le u, \ \forall e \text{ enlace de interconexión} \tag{1h}$$

Las variables de decisión (1a) y (1b) definen los enlaces para cada LSP, y cómo el tráfico de cada par entrada-salida se divide entre los dos LSPs asociados. La variable de decisión *u* se requiere para implementar la optimización minimax. Esta variable contiene la utilización en el enlace de interconexión más cargado en la red el cuál es la cifra para minimizar en (1d).

Las restricciones (1e) fuerzan que el tráfico transportado por un LSP en un enlace sea cero, si ese enlace no está incluido en la ruta del LSP. Las restricciones (1f) son restricciones de conservación de flujo en un enlace para el problema de enrutamiento. Las restricciones (1g) fuerzan que todo el tráfico entre cada par salida-entrada sea transportado conjuntamente por los dos LSPs asociados. Finalmente, las restricciones (1h) fuerzan, junto con la función objetivo, que la variable *u* almacene el máximo valor de utilización en cualquier enlace de interconexión. Hay que tener en cuenta que la utilización en los enlaces *edge* no restringen el problema: las capacidades de los enlaces *edge* se calcularán en una fase post-procesado, descrita en la siguiente subsección.

El resultado de esta formulación MILP es un conjunto de LSPs el cuál transporta el tráfico de la matriz base, balanceando el tráfico para que la máxima utilización se minimice. Hay que tener en cuenta que al multiplicar la matriz de tráfico original Tsd por un factor de normalización f, también multiplicaría el tráfico en cada LSP por el mismo factor f. Pero esto ni cambiaría las rutas de los LSPs ni tampoco cambiaría la proporción de tráfico dividido entre los dos LSPs asiciados entre el mismo par de nodos de entrada-salida.

La formulación (1) se ha solucionado en una plataforma TOMLAB/CPLEX [9]. El enrutamiento del tráfico planificado resulta en un número de enlaces medio atravesado de 3.0008. Para la matriz de tráfico dada, la diferencia en en la máxima utilización de un enlace entre usar dos LSPs por cada par de nodos y un número infinito de LSPs es inexistente debido a la precisión numérica del solver.

4.2.4 Edge node y parámetros de ensamblado

La figura Fig. 14 muestra la estructura de un *edge node*. El tráfico eléctrico de entrada de cada LSP se supone que está compuesto por paquetes IP, los cuales son ensamblados por el correspondiente módulo *assembler*. En el escenario de pruebas, cada *edge node* contiene 26 LSPs, dos LSPs dirigidos a cada uno de los otros 13 nodos en la red.

Una plétora de técnicas de ensamblado se han propuesto para los *edge nodes* OBS. Esto es ampliamente aceptado en la actualidad, ya que el ensamblado de ráfagas debe seguir un enfoque híbrido entre el ensamblado basado en tiempo y el ensamblado basado en volumen. En una simulación por eventos discretos de una red extensa, el modelado de cada proceso de ensamblado en cada LSP penaliza de forma extrema el tiempo de ejecución de la simulación. Afortunadamente, se han propuesto varios modelos para estimar el proceso de salida de un *assembler*, los cuales han mostrado resultados exactos. En este trabajo, la salida de cada *asembler* se modela como una secuencia de ráfagas ensambladas con una duración de *payload* aleatoria siguiendo una distribución normal truncada, con una duración media de 55 µs (lo que implica 68,750 bytes a una tasa de transmisión de 10 Gbps). La duración mínima y máxima de la ráfaga se fija a 10 µs y a 100 µs respectivamente. Estos valores corresponden a un volumen mínimo de ráfaga de 12,500 bytes y un máximo de 125,000 bytes con un transmisor de 10 Gbps. Algunos estudios han mostrado una relación más o menos fuerte entre las pérdidas de la red OBS, y la variabilidad observada en la longitud del *payload* [11]. Nosotros incluímos este aspecto en nuestro estudio, evaluando diferentes valores del coeficiente de variación (*CV*) de la longitud del *payload* en una distribución normal truncada.

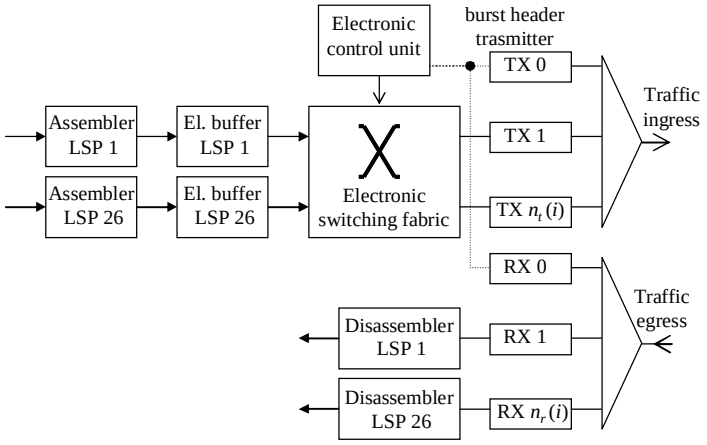

Fig. 14: Modelo de un *edge node*

Para cada *assembler*, el tiempo entre el ensamblado de dos ráfagas es aleatorio, con una distribución exponencial negativa. Se ha elegido entra distribución sin memoria debido a sus efectos beneficiosos esperados en el rendimiento de las pérdidas. El tiempo medio entre dos ráfagas consecutivas se establece en cada simulación para que el tráfico medio generado en Erlangs coincida con el calculado para cada LSP.

Las ráfagas que dejan el *assembler* son almacenadas en una cola eléctrica, separadas para cada LSP, esperando su turno para ser transmitidas. El tiempo de transmisión de cada ráfaga ensamblada y el transmisor que se va a emplear se calcula por medio de la unidad de control, cuando la ráfaga deja el *assembler*. Cada *payload b* tiene un tiempo mínimo de espera en memoria electrónica, dado por el offset inicial de la ráfaga, calculado para cada LSP como sigue:

$$off\_init(b) = \max(0, H \cdot \Delta off) \quad (2)$$

H : number of hops in the LSP of burst b

El número de saltos en cada LSP depende de su ruta. En nuestro caso tenemos en cuenta como salto, el salto inicial desde el *edge node* fuente, y el último salto que llega al *edge node* destino. Se asume que offset de la ráfaga se reduce una cantidad de Δ*off* en cada salto. Si el parámetro Δ*off* es negativo, significa que el offset se incrementa salto a salto. En este caso el offset inicial se pone a cero. La importancia del parámetro Δ*off* se explicará en una subsección posterior.

Para planificar el tiempo de transmisión y el canal de transmisión de una ráfaga *b*, el edge node ejecuta una variación del algoritmo LAUC-VF [12], para el *edge node*: selecciona el canal de transmisión que es capaz de acomodar el *payload* más pronto, pero no antes de *off_init(b)* μs. El sistema debe garantizar que para cada transmisor, la cola del *payload* se separa un mínimo de tiempo de la cabeza de la siguiente. Este mínimo *interburst-gap* se requiere entre dos ráfagas consecutivas para que los dispositivos de conmutación óptica se reconfiguren en el siguiente nodo. El mismo concepto se aplica entre cada dos ráfagas consecutivas en cualquier enlace en una red OBS. Vamos a asumir que este tiempo de reconfiguración óptico es igual a 0.03 μs.

Las consecuencias del requerimiento de un mínimo *interburst-gap* son que el tiempo de slot requerido para acomodar una ráfaga en una determinada longitud de onda en cada salto en la red debe de ser igual a la longitud del *payload* más 0.03 μs. Como este tiempo extra se necesita para cada ráfaga en cada salto en la red, se puede modelar fácilmente incrementando la longitud del *payload* por 0.03 μs en los tests.

Para cada *edge node* i=1,…,14, el número de transmisores $n_t(i)$ debe ser lo suficientemente elevado para llevar el tráfico *egress* desde el nodo. Como la inyección del tráfico es conservativo (ningún transmisor está parado si una ráfaga está preparada para ser transmitida), este número mínimo es igual al número de Erlangs inyectados, redondeando superiormente. En este punto surge una compensación. Por un lado, este mínimo número de transmisores implica un beneficioso *shaping* del tráfico, ya que esto prohíbe lleguen un excesivo número de ráfagas al nodo de interconexión. Por otro lado, los paquetes se encuentran esperando en memoria electrónica, y una utilización cercana a 1 en el enlace del *edge node,* multiplicaría los requerimientos de *buffering* electrónico y el retardo en cola electrónica. Como compromiso, hemos calculado el número de transmisores para que la utilización media coincida con la máxima utilización calculada en los tests para los enlaces de interconexión. Si tenemos un número fraccional de transmisores, se redondea al entero superior.

El número de receptores del *edge node* $n_r(i),i=1,…,14$, se calcula de forma diferente. Incrementando este número, podemos reducir las pérdidas de ráfagas en el último salto, con un moderado incremento en el coste. Esto se mantiene debido a que los *edge nodes* se supone que se encuentran en las mismas instalaciones que los nodos de interconexión, separados por una distancia corta. Por lo tanto, el coste de operar y mantener los enlaces *edge* se supone que es despreciable. Sin embargo, un alto número de receptores implicaría un alto coste de los *edge nodes*, y un alto número de puertos en la matriz de conmutación óptica del nodo de interconexión asociado. Por lo que se debe de tomar una decisión respecto a esto. De nuevo, nuestro enfoque es el de garantizar el mejor escenario posible para el paradigma OBS sin buffer cuando sea posible. Por lo tanto, vamos a asumir que el número de receptores es lo suficientemente elevado para eliminar cualquier posible pérdida de ráfaga en el último salto de la red.

4.2.5 Nodos de interconexión

La figura Fig. 15 muestra el esquema de un nodo de interconexión con *N* fibras de interconexión de salida y de entrada y un enlace de salida a su *edge node* asociado. El número de longitudes de onda en los enlaces de interconexión *W* es un parámetro en los tests, mientras que el canales activos en el enlace al *edge node i* es igual a su número de transmisores/receptores, $n_t(i)$ y $n_r(i)$ respectivamente. Esta cantidad se calcula nodo a nodo como se describió en la sección previa.

La matriz de conmutación óptica (OSF) conmuta de forma transparente las ráfagas ópticas desde los puertos de entrada a los puertos de salida. En este trabajo, vamos a

considerar OSFs con una conversión completa de longitudes de onda. Esto significa que una ráfaga que llega a través de cualquier longitud de onda y de cualquier fibra de entrada se puede conmutar a cualquier fibra de salida y longitud de onda de salida. Este caso es extremadamente favorable desde el punto de vista de la contención.

Las cabeceras que llegan al nodo *traversing* son convertidas a formato eléctrico y procesadas de forma secuencial (aunque de forma paralela para diferentes fibras ópticas de salida) por una unidad de control electrónica, la cual implementa el algoritmo LAUC-VF [12]. El emplear LAUC-VF como único *scheduler* en los tests es debido a que en numerosos estudios donde este algoritmo muestra un límite superior para el rendimiento de la probabilidad de pérdida de ráfagas.

Vamos a considerar que el tiempo requerido para el procesamiento de las cabeceras T_{proc} es constante en cada nodo y ráfaga, e igual a 10 µs. Los canales de datos en cada fibra de entrada pasan a través de una FDL de longitud *D*. Por tanto, un *payload b* llega a la matriz de conmutación *off(b)+D* µs después de la llegada de la cabecera de la ráfaga, siendo *off(b)* el offset de la ráfaga *b*. Siguiendo este sencillo modelo, la reducción en el tiempo de offset de la ráfaga después de atravesar un nodo viene dada por:

$$\Delta off = Tproc - D \qquad (3)$$

De esta forma, incrementar el retardo a la entrada *D* que sufre el *payload* en un nodo, implica que la disminución del offset que este nodo provoca se reduce, y consecuentemente el offset inicial de los LSPs que lo atraviesan se puede reducir. En el límite, si el valor del retardo *D* coincide con el tiempo de procesamiento, entonces el offset es constante salto a salto, y el offset inicial se hace cero. Si el valor del retardo *D* es mayor que el tiempo de procesamiento, el offset se incrementa salto a salto. Esto se puede emplear como mecanismo para compensar las pérdidas de las ráfagas para los LSPs que atraviesan más nodos. Este tema se estudiará en la siguiente subsección.

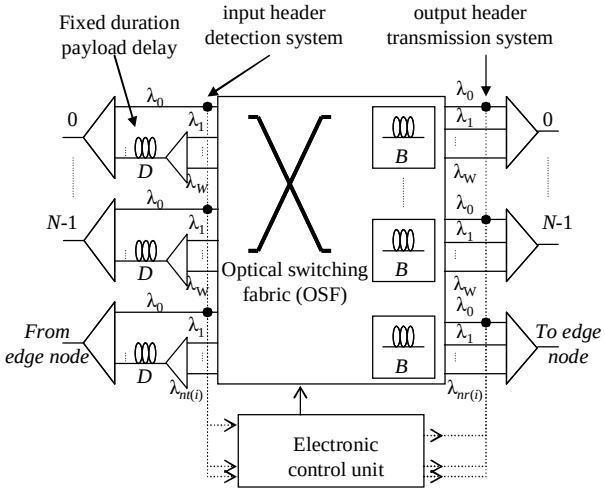

Fig. 15 Modelo de un nodo de interconexión

4.2.6 Protocolo de reserva

Un ámplio rango de protocolos de reserva se han propuesto para las redes OBS (algunos de ellos vistos en la sección 1.2.2). El protocolo de reserva elegido, determina dos procesos diferentes en la red: como se calcula el tiempo de offset de la ráfaga, y como el offset varía salto a salto. Esto es importante desde el punto de vista de la resolución de la contención ya que el offset de una ráfaga impacta en la probabilidad de pérdidas. La razón es que para un *payload* que llegue en un tiempo *t*, cuanto más anticipada sea la decisión tomada con respecto

al resto de *payloads* que llegan en tiempos similares (esto es, cuanto mayor sea el offset con respecto al resto de *payloads* que compiten), mayor son las oportunidades de encontrar recursos de *buffering*.

Como punto inicial, el protocolo JET (*Just-Enough-Time*) explicado en el apartado 1.2.2 asigna un offset inicial fijo a las ráfagas, el cual se reduce salto a salto debido al tiempo de procesamiento de las cabeceras en los nodos. Este tema ha sido usado para suministrar un tipo de diferenciación de QoS, asignando mayores offsets iniciales a las ráfagas de mayor prioridad. Sin embargo, este tema se ha mostrado difícil de lograr y no es robusto en redes *mesh* ya que las ráfagas de menor prioridad de los nodos más cercanos pueden perder la prioridad frente a ráfagas de mayor prioridad que vienen de nodos más lejanos. Para tratar este problema se han propuesto en otros trabajos la opción de establecer un valor aleatorio para el offset inicial de las ráfagas inyectadas, con el objetivo de evitar posibles efectos de sincronización.

Con el fin de simplificar el diseño del protocolo de señalización y la implementación del sistema, se ha propuesto un protocolo denominado *Only Destination Delay* (ODD), el cuál asocia a cada ráfaga un tiempo de offset constante y usa los retardos por fibra de entrada de longitud *D* (ver Fig. 11) en cada nodo intermedio para retardar la ráfaga un tiempo igual al tiempo exacto de procesamiento de las cabeceras. Un tiempo constante de offset simplifica la ingeniería de la red, eliminando la variación en las pérdidas observadas en las ráfagas con diferentes offsets. También simplifica la implementación de los *schedulers*: con un offset constante, el orden de llegada de los BHPs es igual al orden de llegada de los *payloads*. Un offset constante significa por ejemplo que el algoritmo LAUC-VF puede ser equivalente a su versión simplificada denominada LAUC (ver sección 1.3.3)

Otro enfoque que se comentó en el apartado 1.2.2 es el HPI (*Hop-by-hop Priority Increasing*). En este casi se propone que todas las ráfagas tengan un offset inicial de cero pero la fibra de retardo a la entrada *D* en cada nodo de interconexión se diseña para que tenga un valor mayor que el tiempo de procesamiento. Como consecuencia, el offset crece salto a salto. La intención de este enfoque es la de compensar las altas pérdidas *end-to-end* observadas en los LSPs más largos, incrementando de forma lineal su offset en cada salto.

Con el fin de concentrar los enfoques JET,ODD y HPI en nuestro estudio, incluimos la variación de offset de la ráfaga salto a salto (*Δoff*), definido en (3), como un parámetro de testeo. Los valores positivos de *Δoff* implican que los offsets se reducen salto a salto siguiendo el enfoque JET. Esto es debido a que el retardo de entrada *D* no compensa completamente el tiempo de procesamiento de la cabecera (T_{PROC}=10 μs).El caso de *Δoff*=0 sucede cuando *D*=10 μs, y corresponde con el protocolo de reserva ODD. Un valor negativo de *Δoff* corresponde con el enfoque HPI.

Cabe señalar que predecir un valor constante de *Δoff*, requiere de un tiempo de procesamiento que sea constante y predecible para cada ráfaga, y de un canal de control para transmitir los BHPs que esté libre de contención (como los BHPs están almacenados de forma electrónica esperando su turno para usar el canal de control). Como ambos requerimientos no son posibles de obtener, el efecto generado puede ser el de un pequeño *jitter* en el tiempo de offset de la ráfaga. Como buscamos un escenario de pruebas que sea favorable para el caso de red OBS sin buffer, vamos a considerar el impacto de este *jitter* en la probabilidad de pérdida de ráfaga como un valor despreciable, y por tanto no incluimos este efecto en el modelo.

4.2.7 Diferenciación QoS

Existen diferentes técnicas que se han propuesto para suministrar diferenciación de QoS en las redes OBS, permitiendo la existencia de ráfagas con diferentes prioridades. Aunque la diferenciación de QoS es un tema relevante e importante, éste no es el principal objetivo de este trabajo. Las leyes de conservación de colas (por ejemplo las leyes de conservación de Kleinrock's) aplicadas a los nodos de conmutación ópticos, y las colas electrónicas en los *ingress*, y la mejora del rendimiento de las pérdidas/retardo de un subconjunto de ráfgas podría ser a un coste de disminuir las pérdidas/retardo del resto. Nosotros estamos interesados en la evaluación de la habilidad del paradigma OBS sin buffer para suministrar un rendimiento medio

de pérdidas aceptable. Esto puede ser más conveniente probarlo asumiendo una única clase de tráfico. Por ello, esta va a ser la opción tomada en este trabajo.

4.3 Resultados

Esta sección presenta los resultados testeados. Los resultados se han obtenido por medio de simulación, usando una extensión de la herramienta oPASS, implementada mediante OMNET++. Los *edge nodes*, *interconection nodes* y todos los elementos de la red se han modelado siguiendo la descripción suministrada en la sección previa.

En la sección 4.2, se han identificado cuatro parámetros, que van a ser testeados en nuestro estudio para investigar su impacto en el rendimiento de la red. A continuación, vamos a enumerar los parámetros bajo test y sus correspondientes valores.

- *W={20,40,80}*, número de longitudes de onda por enlace entre los nodos de interconexión.
- *u={10%,20%,30%,40%,50%,60%,70%,80%,90%}*, la máxima utilización permitida en un enlace entre los nodos de interconexión.
- *CV={0,0.5,1,1.5}*, el coeficiente de variación en la distribución del tamaño del *payload*.
- $\Delta off = \{10,8,5,2,0,-2,-4\}$, es la reducción salto a salto en el offset de la ráfaga, como se explicó en la sección 4.2, los valores negativos implican que el offset actual se incrementa salto a salto.

Para cada combinación de valores, se realiza un punto de simulación. Cada simulación se compone de un tiempo transitorio, seguido por cinco intervalos de tiempo consecutivos. La duración de cada intervalo es la requerida para garantizar que la parte *egress* de cada LSP en la red ha recibido al menos $2 \cdot 10^5$ paquetes. Los resultados estadísticos extraídos de cada intervalo son considerados muestras independientes, y el intervalo de confianza se calcula en cada caso, aplicando el método *t-student*. El tiempo transitorio inicial fue calculado para ser el 10% de la duración esperada de un intervalo.

4.3.1 Máxima utilización dentro de las pérdidas objetivo

En nuestro caso, hemos fijado la probabilidad de pérdida de bit a un valor de 10^{-4} (la probabilidad de pérdida de bit significa que una pérdida de ráfaga se pondera por la longitud de la ráfaga). En este caso, buscamos el enlace con una utilización más alta *u* para el cual las pérdidas objetivo se cumplen (i) como una probabilidad de pérdida *end-to-end* (ii) para todos los LSPs. Centrarse en el rendimiento *end-to-end* es una elección importante, ya que representa el rendimiento percibido por el usuario de la red.

Hay que tener en cuenta que el valor *u* es directamente proporcional, y por tanto representativo, de la demanda máxima de tráfico que puede ser llevada dentro de las pérdidas objetivo. Los resultados han sido sorprendentes en varios aspectos. Primero, la máxima utilización *u* es la misma para cada valor de Δoff y *CV*, y solo es dependiente del número de longitudes de onda por fibra *W*. Los valores de utilización obtenidos son de *u=0.3* y *u=0.4* para *W=20* y *W=40* longitudes de onda por fibra respectivamente. Para el caso de *W=80* se obtiene una utilización más satisfactoria de *u=0.6*. Los tres casos corresponden a un total de tráfico transportado en la red de 0.92 Tbps, 2.45 Tbps y 7.35 Tbps respectivamente. Si las pérdidas *end-to-end* objetivo para el peor LSP se estableciesen en 10^{-5} , la máxima utilización sería en este caso de 0.2, 0.4 y 0.5 para *W={20, 40, 80}* respectivamente. Sin embargo, se necesitan simulaciones más largas para validar los resultados para unas pérdidas objetivo de 10^{-5}.

Hay que realizar un comentario sobre los resultados obtenidos. Algunos estudios previos habían sugerido que la probabilidad de bloqueo de un nodo OBS se podría dimensionar, en algunos casis, aplicando la fórmula Erlang-B, asumiendo llegadas Poissonianas en cada longitud de onda y en cada fibra óptica de entrada. Nosotros hemos verificado que esta afirmación va en contra de los resultados simulados. Hemos observado que, en todos los escenarios bajo test, las probabilidades de pérdidas obtenidas se encuentran entre uno y dos órdenes de magnitud peores que aquellas predichas por el modelo Erlang-B.

4.3.2 Retardo en el ingress y coste electrónico

El restardo en el *ingress* es el tiempo desde que una ráfaga deja el módulo *assembler* hasta que su último bit abandona el buffer electrónico por LSP. Estamos interesados en medir este tiempo, el cual está estrechamente relacionado con la cantidad de memoria electrónica que se tiene que asignar por buffer electrónico. La Tabla IV muestra, para cada punto de prueba, el tamaño máximo alcanzado por cualquiera de los buffers electrónicos en cualquier LSP, asumiendo transmisores a 10 Gbps. Este valor suministra un límite al mayor retardo *ingress* sufrido por cualquier ráfaga (esto es, no son valores medios a lo largo del tiempo sino valores máximos a los largo del tiempo).

Algunas conclusiones interesantes se pueden extraer de estos resultados. Primero, los requerimientos de *buffering* son relativamente pequeños y caen muy bien dentro de las capacidades actuales de los bufferes electrónicos (3.27 MB de buffer, o 2.6 ms de retardo en el peor de los casos). Naturalmente, los valores menores se obtienen para los tiempos de offset iniciales menores. Sin embargo, es interesante ver que en estos tests, los requerimientos de buffering/retardo en el *ingress* son bastante similares para los protocolos de reserva JET/ODD/HPI (Retardos JET son entre un 5% y un 35% peores que los retardos en ODD o HPI).

También se observa que para mayores valores de *CV* y mayor número de longitudes de onda llevan a un mayor retardo en el peor caso. Esto es también lógico ya que el tiempo de espera en las colas del *ingress* se degrada por la variabilidad en el tráfico *ingress* y de su volumen total (dependiente del número de longitudes de onda).

Tabla IV: Máximo tamaño de memoria por LSP

λ \ Δ *off* / *CV*		10	8	5	2	0	-2	-4
20	0	1.24	1.18	1.11	1.11	1.04	1.11	1.04
	0.5	1.54	1.38	1.37	1.27	1.19	1.28	1.32
	1	1.49	1.38	1.37	1.33	1.36	1.30	1.30
	1.5	1.51	1.39	1.39	1.29	1.28	1.27	1.28
40	0	1.77	1.77	1.64	1.50	1.44	1.57	1.44
	0.5	1.97	2.00	1.84	1.82	1.69	1.69	1.83
	1	2.20	2.04	1.93	1.92	1.76	1.75	1.91
	1.5	2.20	1.95	1.92	1.94	1.70	1.78	1.80
80	0	2.69	2.69	2.36	2.23	2.16	2.16	2.16
	0.5	2.99	3.05	2.75	2.57	2.37	2.61	2.49
	1	3.16	3.19	2.86	2.68	2.52	2.60	2.82
	1.5	3.12	2.98	2.92	2.70	2.58	2.58	2.57

4.3.3 Longitud del payload vs distribución de probabilidad de pérdida

Varios trabajos han estudiado la dependencia entre la probabilidad de pérdida de ráfaga y la longitud de la ráfaga. En cada test, nosotros hemos calculado la probabilidad de pérdida de ráfaga condicional al tamaño del *payload*, particionando la longitud del *payload* desde 10 µs hasta 100 µs en slots de 3 µs. Los resultados han revelado que las pérdidas se distribuyen de manera uniforme. Hemos calculado de medida de Jain para la justicia para la distribución condicional. Todos los valores son extremadamente altos (>0.99) lo que está bastante cercano al valor máximo de 1 que significa una uniformidad perfecta. Entonces, podemos concluir para

este escenario que la probabilidad de pérdida de ráfaga no es significativamente dependiente del tamaño de ráfaga en cualquier caso. Esto es lógico en nuestro caso, ya que las oportunidades de crear huecos en el sustema son escasas. Los huecos son la fuente de la diferenciación de pérdidas dependiendo del tamaño de la ráfaga para las siguientes llegadas de ráfagas: pequeñas ráfagas tendrán mejores oportunidades de ser asignadas en huecos, que las que son mayores. Si no se generan huecos en el sistema, una ráfaga se asigna a un canal dependiendo del solape potencial de la cabecera de la ráfaga con las asignaciones previas, sin influencias de la longitud de la ráfaga en la probabilidad de pérdida.

Cabe destacar que una probabilidad de pérdida independiente de la longitud del *payload*, en realidad favorece a aquellos *assemblers* que tienden a crear ráfagas más largas, los cuales tienen la misma probabilidad de pérdida que las ráfagas más pequeñas pero pueden llevar más tráfico.

4.4 Discusión de resultados

¿Es la red OBS sin buffer una factible desde el punto de vista de la resolución de contención?. No se puede dar una respuesta definitiva ni tampoco es el objetivo de este trabajo.

De acuerdo con los resultados obtenidos, modestamente pensamos que OBS sin buffer presenta una manera pobre de utilización de la red cuando el número de longitudes de onda no es demasiado elevado. Un 30% y un 40% de utilización de la red (correspondiente a los casos de *W=20, W=40*) para una probabilidad de pérdida de 10^{-4} no puede competir con otros enfoques como por ejemplo *Optical Circuit Switching (OCS)* los cuales tienen mucho menos requerimientos hardware. Especialmente, si comparamos con el coste del hardware de la conversión completa de las longitudes de onda asumidos en los nodos de interconexión de la red OBS.

Por otro lado, el caso de *W=80* permite tener una aceptable utilización de *u=0.6.* Sin embargo, los malos resultados obtenidos para los casos *W={20,40},* penalizan enormemente la migración de escenarios en los que OBS y OCS comparten la red, y algunas longitudes de onda pero no todo lo dedicado para la parte OBS de la red.

Se pueden extraer otras conclusiones del estudio realizado. Primero, con respecto a la precisión del modelo Erlang-B para dimensionar el sistema, los resultados obtenidos muestran un importante hueco entre la simulación y la probabilidad de pérdida predicha por el modelo. Segundo, con respecto a el protocolo de reserva, hemos observado que la utilización de la red alcanzable en todos los casos es la misma. Además, las diferencias en el *buffering* del *edge node* y en el retardo del *ingress* no parecen significativas, y están perfectamente dentro de los límites tecnológicos actuales de almacenamiento electrónico. Es por esto que desde el punto de vista de los autores, la selección de un protocolo de reserva debe mantener un equilibrio entre dos opciones. Por un lado, escoger el protocolo ODD puede resultar en una simplificación considerable de los *schedulers* OBS: un offset constante en una red OBS sin buffer significa que el oden de llegada de los BHPs coincide con el orden de llegada de los *payloads* lo que lleva a una simplificación del hardware de la unidad de control. Por otro lado, la señalización ODD puede ser difícil de implementar, y todavía mantiene un *jitter* en el offset causado por la contención de los BHPs y por los tiempos de procesamiento no constantes. Esto puede justificar el uso de una propuesta simple como la JET.

4.5 Conclusiones y trabajos futuros

En las páginas anteriores, hemos construido un escenario de pruebas, con el objetivo de encontrar un caso límite razonable para la probabilidad de pérdidas para una red OBS sin buffer de referencia. Los resultados se han analizado. Aunque no se pueden tomar conclusiones que sean claras y estrictas, los resultados sugieren que el paradigma OBS sin buffer es penalizado por una pobre resolución de la contención que lleva a unas figuras de

utilización de la red pobres. Esta degradación del rendimiento se obtuvo con asunciones favorables como son la conversión completa de longitudes de onda en los nodos de interconexión, y el número infinito de receptores en los *edge nodes*. Por lo tanto, esto muestra que existe un pequeño margen para la mejora del rendimiento del paradigma OBS.

Los trabajos futuros están siendo conducidos en este tema, con la extensión de los test en diferentes aspectos (por ejemplo topologías de red) con el fin de suministrar una conclusión de mayor alcance.

Capítulo 5
Conclusiones y líneas futuras

Este trabajo fin de Máster se centra en las redes de conmutación óptica de ráfagas, y su evaluación mediante la herramienta de simulación modificada oPASS.

Para esta evaluación se han estudiado tanto topologías de red multinodo (como es el caso de la topología de red NSFNET) como en casos mononodo que delimitan un escenario de pruebas de interés.

De los resultados obtenidos podemos sacar las siguientes conclusiones:

- La extensión a la herramienta de simulación oPASS nos permite realizar simulaciones tanto mononodo como multinodo y con topologías de red tanto *Optical Packet Switching* como *Optical Burst Switching*. Además de esto, podemos estudiar tanto los casos síncronos como los asíncronos los que nos permite evaluar un sinfín de posibilidades.
- PI-OBS es la primera propuesta de un *scheduler* paralelo e iterativo para conmutadores OBS. En contra de los enfoques codiciosos convencionales, todas las cabeceras recibidas en una ventana temporal dada, son procesadas de forma conjunta. Esto abre un campo para una ganancia de rendimiento, cuando se compara con otros enfoques codiciosos como el enfoque LAUC-VF. También, la convergencia estudiada del algoritmo muestra un tiempo de respuesta que es aproximadamente independiente del tamaño del *switch*.
- Para redes OBS sin buffer desde el punto de vista de los autores, la selección de un protocolo de reserva debe mantener un equilibrio entre dos opciones. Por un lado, escoger el protocolo ODD puede resultar en una simplificación considerable de los *schedulers* OBS: un offset constante en una red OBS sin buffer significa que el oden de llegada de los BHPs coincide con el orden de llegada de los *payloads* lo que lleva a una simplificación del hardware de la unidad de control. Por otro lado, la señalización ODD puede ser difícil de implementar, y todavía mantiene un *jitter* en el offset causado por la contención de los BHPs y por los tiempos de procesamiento no constantes. Esto puede justificar el uso de una propuesta simple como la JET.

Las líneas futuras que pueden continuar con este proyecto, son las basadas en la introducción de distintas implementaciones sobre el simulador como son:

- Observando las similitudes del PI-OBS con los *schedulers* VOQ, los autores de este trabajo están trabajando en una implementación práctica del *scheduler*, explorando la relación entre la complejidad de implementación y el tiempo de respuesta del algoritmo.
- Los autores consideran el algoritmo PI-OBS como el primer paso. Se pueden explorar variaciones del algoritmo PI-OBS para aplicar nuevas estrategias de asignación de recursos de forma conjunta, consiguiendo mejoras en el rendimiento y/o simplificaciones hardware.
- En colaboración con la universidad de Bolonia se requiere ampliar los resultados del PI-OBS. Para ello vamos a ahondar en el estudio de la viabilidad de la

implementación que se ha realizado teóricamente. Además también se va a estudiar el número de bits que se necesitan para implementar los módulos de entrada y de salida. Otro de los temas a tratar es el de estudiar el desorden que introduce el algoritmo PI-OBS para redes multinodo.

- Para el caso de redes OBS sin buffer, la idea es la de hacer más exhaustivos los estudios realizados. Para ello, se van a implementar más topologías y a la vez se van a hacer más sólidos los resultados.

Bibliografía

[1] Y. Chen, C. Qiao, X. Yu, “Optical Burst Switching: A New Area in Optical Networking Research”, *IEEE Network*, vol. 18, no. 3, May/June 2004.

[2] S. L. Danielsen, C. Joergensen, B. Mikkelsen and K. Stubkjaer, "Analysis of a WDM packet switch with improved performance under bursty traffic conditions due to tunable wavelength converters", *IEEE Journal of Lightwave Technology*, vol. 16, no. 5, pp. 729-735, May 1998.

[3] M. Yoo and C. Qiao, “Just-enough-time (JET): A High Speed Protocol for Bursty Traffic in Optical Networks,” *IEEE/LEOS Tech. Global Info. Infra.*, pp. 26–27, Aug. 1997.

[4] N. McKeown, “iSLIP: A Scheduling Algorithm for Input-Queued Switches”, *IEEE/ACM Transactions on Networking*, vol. 7, no. 2, pp. 188-201, April 1999.

[5] F. Callegati, W. Cerroni, G. S. Pavani, “Key Parameters for Contention Resolution in Multi-Fiber Optical Burst/Packet Switching Nodes” (Invited Paper), *Proc. of IEEE Broadnets 2007*, Raleigh, NC, Sept. 2007.

[6] J. Turner, “Terabit Burst Switching”, *Journal of High-Speed Networks*, vol. 8, no. 1, 1999.

[7] C. Qiao and M.Yoo, “Optical burst switching (OBS)—A new paradigm for an optical internet,” J. High Speed Netw., vol. 8, pp. 69–84, 1999.

[8] Mills, D. L. and Braun, H., “The NSFNET backbone network. In Proceedings of the ACM Workshop on Frontiers in Computer Communications Technology” (Stowe, Vermont, United States, August 11 - 13, 1987). J. J. Garcia-Luna-Aceves, Ed. SIGCOMM '87. ACM, New York, NY,

[9] http://tomopt.com/tomlab/. Last accessed 31th May. 2009.

[10] A. Rostami and A. Wolisz, ”Modeling and fast emulation of burst traffic in optical burst-switched networks,” in Proceedings of IEEE ICC, vol. 6, pp. 2776-2781, 2006.

[11] P. Pavon, J. Veiga, A. Ortuño, W. Cerroni, J. Garcia, “PI-OBS: a Parallel Iterative Optical Burst Scheduler for OBS networks”, submitted for publication in IEEE International Workshop on High Performance Switching and Routing 2009, HPSR'09, June 2009, Paris, France.

[12] Y. Xiong, M. Vandenhoute, H. C. Cankaya, “Control Architecture in Optical Burst-Switched WDM Networks”, IEEE Journal on Selected Areas in Communications, vol. 18, no. 10, Oct. 2000.

[13] J. Turner, “Terabit burst switching,”Journal of High Speed Networks, vol.8, pp. 3-16, 1999.

[14] P. Pavon-Mariño, A. Ortuño-Manzanera, "Bufferless Optical Burst Switching?", submitted for publication in International Workshop in Optical Burst/Packet Switching, Madrid, September 2009.