Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO FINAL DE CARRERA

TÍTULO DEL TFC: Estudio y simulación del DBA OBS-aware para EPONs

TITULACIÓN: Ingeniería de Telecomunicaciones, especialidad en Telemática

AUTORES: Rubén Herráiz Buendía
Albert Nogueira Ventura

DIRECTORA: Anna Agustí Torra

FECHA: 17 de mayo de 2012

Título: Estudio y simulación del DBA OBS-aware para EPONs

Autores: Rubén Herráiz Buendía
Albert Nogueira Ventura

Directora: Anna Agustí Torra

Fecha: 17 de mayo de 2012

Resumen

El desarrollo de tecnologías de red de acceso basadas en el uso de la fibra óptica como medio de transmisión ha permitido, en comparación con las redes de acceso de cable, incrementar en varios órdenes de magnitud la capacidad de transmisión de datos en el segmento más cercano al usuario.

Las redes de acceso ópticas pasivas (*Passive Optical Networks*, PONs) se caracterizan por no tener componentes activos intermedios entre el usuario y la central. Entre ellas se definen las redes Ethernet PON (EPON), cuya arquitectura se basa en el transporte de tráfico Ethernet, manteniendo las características de la especificación 802.3 sobre fibra óptica.

En la red troncal la fibra óptica es el medio de transmisión predominante, siendo la conmutación de circuitos la técnica más utilizada debido, principalmente, a la falta de un equivalente real a las memorias de acceso aleatorio (*Random Access Memory*, RAM) que permita soportar conmutación de paquetes en el dominio óptico. Para poder beneficiarse de las ventajas del multiplexado estadístico de usuarios que proporciona la conmutación de paquetes sin los requisitos tecnológicos que supone su implementación en redes ópticas, se definió la conmutación óptica de ráfagas (*Optical Burst Switching*, OBS). En OBS, los datos de los usuarios se agregan en los nodos frontera formando unidades de datos de longitud superior (ráfagas). Antes de la transmisión de cada ráfaga, se envía un paquete de control con el objetivo de configurar los dispositivos intermedios, permitiendo así que el envío de los datos se pueda realizar íntegramente en el dominio óptico.

El presente estudio desarrolla el escenario de simulación que interconecta una red de acceso EPON con una red troncal basada en la tecnología de conmutación OBS.

Title: Study and simulation of DBA OBS-aware for EPONs

Author: Rubén Herráiz Buendía
Albert Nogueira Ventura

Director: Anna Agustí Torra

Date: 17 de mayo de 2012

Overview

The development of technologies based in Access network using optical fiber as medium of transmission is allowed increase by several orders of magnitude the data transmission capacity in the segment closest to the user compared with wire Access networks.

The passive optical Access networks (PONs) are characterized by having no active components intermediate between the user and the data center. Including Ethernet PON networks are defined, whose architecture is base don the transport of Ethernet traffic, maintaining the characteristics of the specification 802.3, over optical fiber.

In the optical fiber backbone networks is predominant mode of transmission, switching circuits being the most used technique, mainly due to the lack of a real equivalent to the random Access memory (RAM) to allow support packet switching in the optical domain. To qualify for the benefits of statistical multiplexing of uses providing packets switching without technological requirements resulting from their implementation in optical networks, was defined the optical burst switching (OBS). In OBS, the user data are added in units forming the boundary nodes of a length of data (burst). Before transmission of each burst, sending a control packet in order to configure intermediate devices, allowing the sending of data can be made entirely in the optical domain.

This paper develops the simulation scenario that interconnects an EPON Access network with a backbone based on the OBS switching technology.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Las redes de acceso predominantes hasta hoy se basan en reutilizar el par de cobre del bucle de abonado mediante tecnologías DSL (*Digital Subscriber Line*) junto con un multiplexor de acceso ATM (*Asynchronous Transfer Mode*). Sin embargo, cada vez más operadoras optan por remplazarlas por redes de fibra óptica que permiten ofrecer mayores velocidades de transmisión.

Las redes de acceso ópticas se pueden clasificar en función de hasta donde se instala la fibra:

- FTTC (*Fiber To The Cabinet/Curb*).
- FTTB (*Fiber To The Building*).
- FTTH (*Fiber To The Home*).

El principal problema de las redes de acceso ópticas es su coste, sobretodo cuando se utilizan elementos de interconexión activos. Por este motivo, se han desarrollado tecnologías que permiten prescindir de componentes activos intermedios, son las denominadas redes ópticas pasivas (*Passive Optical Networks*, PONs).

Las redes PON se componen de 3 elementos principales:

- ONU (*Optical Network Unit*): Es el elemento más cercano al usuario.
- OLT (*Optical Line Terminator*): Es el elemento situado en la central del operador.
- *Splitter*: Es un divisor de potencia óptico que permite interconectar las ONUs al OLT.

Existen varias tecnologías PON que difieren tanto en la topología como en los protocolos utilizados para la transmisión de información. Una de ellas es la Ethernet PON (EPON), cuya arquitectura se basa en el transporte de tráfico Ethernet, manteniendo las características de la especificación 802.3, sobre fibra óptica. La velocidad de transmisión que permiten las redes EPON es de 1Gb/s (a repartir entre los usuarios) o de 10Gb/s, en función de la implementación.

En muchas PONs, las ONUs comparten el canal de subida (*uplink*) entre el *splitter* y el OLT, de manera que se requieren mecanismos de reparto de ancho de banda (*Dynamic Bandwidth Assignment*, DBA) para evitar las colisiones.

La fibra óptica es el medio de transmisión predominante en las redes troncales, siendo la conmutación óptica de circuitos, basada en la reserva de un canal extremo a extremo (*lightpath*), la técnica de conmutación más común debido a la dificultad tecnológica de implementar la conmutación óptica de paquetes (por falta de un equivalente real a las memorias de acceso aleatorio, RAMs, en el dominio óptico). Con el fin de aprovechar las ventajas del multiplexado estadístico de datos pero sin los requerimientos tecnológicos de la conmutación

óptica de paquetes, se definió la conmutación óptica de ráfagas OBS (*Optical Burst Switching*). En una red OBS, los datos de los usuarios se agregan en los nodos frontera formando bloques de datos de longitud superior (denominadas ráfagas). Previamente a la transmisión de cada ráfaga, se envía un paquete de control que es procesado en el dominio electrónico en cada nodo intermediario de la red, con la finalidad de reservar los recursos necesarios para que el envío de la ráfaga se pueda realizar de forma transparente en el dominio óptico.

En este trabajo se estudia, vía simulación, la integración de las redes EPON y OBS con el objetivo de reducir el retardo extremo a extremo en la transmisión de información. Para conseguirlo se propone utilizar un mecanismo de distribución de ancho de banda (DBA) en el OLT de la red EPON que tenga en cuenta el criterio de agregación de ráfagas que se realiza a la entrada de la red OBS. El uso de este DBA, denominado aquí DBA OBS-aware, supone tener que realizar cambios en el almacenamiento de información en las ONUs, en el contenido de los mensajes de control y en el procesado de la información de control en el OLT para realizar el reparto del ancho de banda del enlace compartido entre las distintas ONUs.

La memoria de este trabajo se organiza de la siguiente manera. La primera parte describe las principales características de las tecnologías PON y OBS. La segunda parte introduce los fundamentos del funcionamiento del simulador OPNET y detalla las modificaciones realizadas con el objetivo de realizar la interconexión de las redes EPON y OBS. Por último, se exponen las conclusiones y se resumen las líneas futuras de trabajo.

CAPITULO 1. INTRODUCCIÓN A LAS TECNOLOGIAS UTILIZADAS

1.1. Introducción a las redes PON

Las redes ópticas pasivas (PON) se desplegaron entorno al año 1988, como alternativa a las redes de acceso de cable, proporcionando una mayor capacidad de transmisión (ancho de banda o BW) y un incremento del alcance (hasta los 20km) al introducir menos atenuación.

Las redes PON son redes de fibra óptica punto a multipunto sin elementos activos intermedios, lo cual reduce el número de transceptores ópticos y de centrales locales requeridas, así como los metros de fibra óptica necesarios para dar conectividad a los usuarios. Además, ofrecen una mejor calidad de servicio que las redes de cable debido a la inmunidad de la fibra óptica a los ruidos electromagnéticos.

Existen varios estándares distintos:

IEEE 802.3ah
- EPON (*Ethernet Passive Optical Network*).

ITU-T G.983
- APON (*Asynchronous Transfer Mode Passive Optical Network*).
- BPON (*Broadband Passive Optical Network*).

ITU-T G.984
- GPON (*Gigabit Passive Optical Network*).

Este trabajo final de carrera se centra en las redes EPON.

1.1.1. Estructura y funcionamiento de una red PON

Una red PON consta de tres elementos principales:

- OLT (*Optical Line Terminal*): Es el nodo central que interconecta la red PON con una red metropolitana o troncal.
- ONU (*Optical Network Unit*): Es el elemento que interconecta a los usuarios con la red PON, siendo el elemento más próximo a los usuarios finales.
- *Splitter* (divisor óptico): Es un divisor de potencia que divide la señal recibida por el OLT para distribuirla hacia las ONUs y viceversa.

La siguiente figura muestra un esquema de la interconexión de los distintos elementos que componen la red PON.

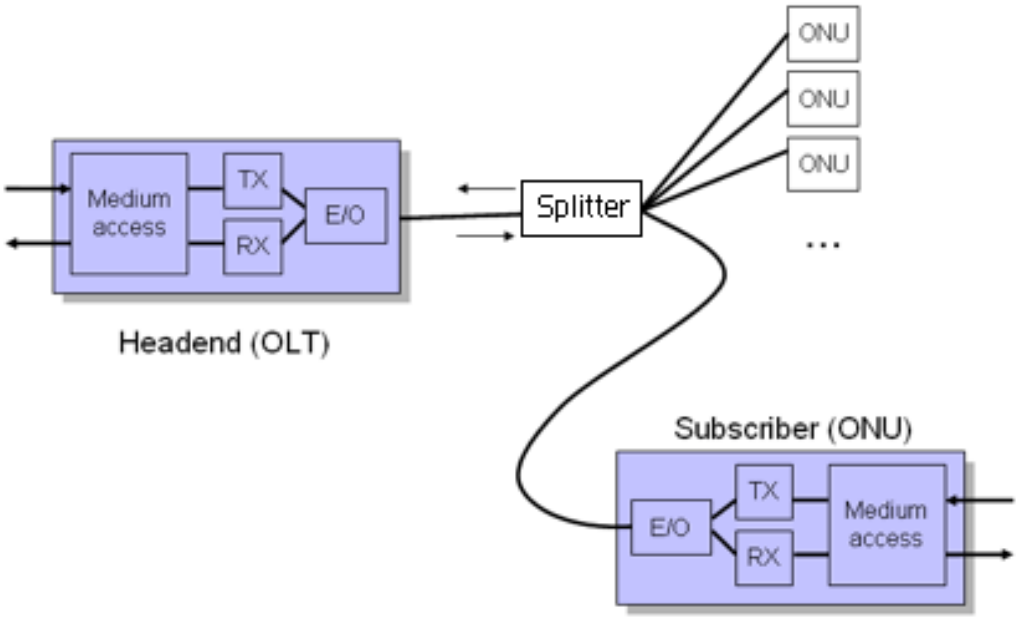

Fig. 1.1. Elementos de las redes PON [13].

La transmisión entre el OLT y las distintas ONUs se realiza a través del *splitter*, donde se transporta la información de los canales de subida y bajada.

- Canal ascendente (*uplink*): Este canal es compartido por todas las ONUs, de modo que, para evitar colisiones, debe establecerse un mecanismo de control de acceso al medio o DBA (*Dynamic Bandwidth Assignment*). Las características del DBA utilizado en la EPON determinan en gran medida el rendimiento de la red, la eficiencia en el uso de los recursos y la calidad de servicio ofrecida a cada ONU. Para distribuir el ancho de banda del canal de subida, el OLT y las ONUs intercambian dos tipos de mensajes: *report* y *gate*. Las ONUs utilizan los mensajes *report* para ir comunicando al OLT la cantidad de datos que desean transmitir. El OLT, a partir de las peticiones que va recibiendo de las ONUs y aplicando el algoritmo de reparto de ancho de banda que corresponda, distribuye el tiempo de uso del enlace ascendente y se lo comunica a las ONUs mediante mensajes *gate*.

- Canal descendente (*downlink*): Es el canal de envío de información desde el OLT hacia todas las ONUs.

Aunque las redes PON puedan tener distintas topologías (árbol, bus, anillo, etc.) la más común es la topología de árbol. La arquitectura depende del alcance de la fibra óptica:

- FTTH (*Fiber To The Home*): En este caso la fibra óptica llega hasta la casa del cliente, utiliza una arquitectura del tipo punto a punto.
- FTTB (*Fiber To The Building*): En este caso la fibra óptica llega hasta el edificio, y posteriormente se distribuye la señal dividiéndola entre los suscriptores.
- FTTC (*Fiber To The Curb/Cabinet*): En este caso, por lo general, la fibra óptica llega hasta un nodo establecido en una calle que da servicio a toda una manzana.

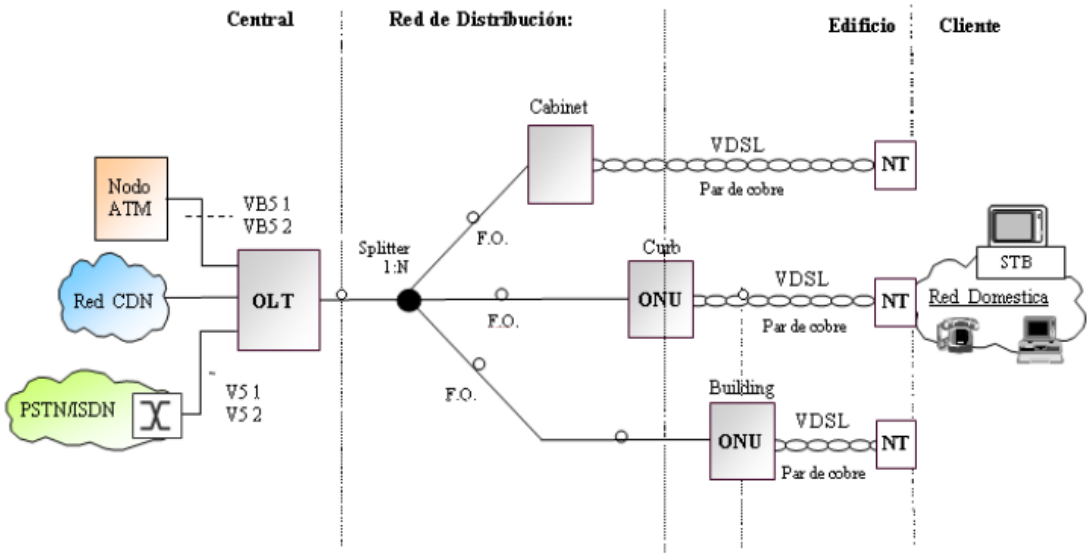

Fig. 1.2. Distribución de las distintas arquitecturas.

1.2. Introducción a las redes OBS

En los últimos años, la definición de nuevos servicios y aplicaciones ha incrementado de manera exponencial la demanda de ancho de banda en las redes troncales de comunicaciones. Actualmente, la fibra óptica es el medio de transmisión predominante en la red troncal y, para hacer frente a la creciente demanda de capacidad, ya son muchas las redes ópticas que incorporan la técnica del multiplexado por división de longitud de onda (*Wavelength Division Multiplexing,* WDM) que permite la transmisión simultánea sobre diferentes longitudes de onda dentro de una misma fibra.

En la mayoría de redes ópticas la fibra se utiliza como medio de transmisión, pero el procesado en los nodos intermedios se realiza en el dominio electrónico.

1.2.1. Conmutación óptica de circuitos

La conmutación óptica de circuitos (*Optical Circuit Switching*, OCS) consiste en la reserva de recursos extremo a extremo, creando un circuito entre el nodo transmisor y el nodo receptor, denominado *ligthpath*, por donde se realiza la transmisión de los datos de forma transparente en el dominio óptico [2].

Tal y como ocurre en el dominio eléctrico, en OCS se desaprovechan recursos cuando, estando reservados, no se realiza transmisión de información.

1.2.2. Conmutación óptica de paquetes

La conmutación óptica de paquetes (*Optical Packet Switching*, *OPS*), se caracteriza por la transmisión de las unidades de datos de los usuarios (paquetes) como unidades independientes. Así, cada paquete contiene una cabecera con la información necesaria para que los elementos intermedios puedan decidir hacia dónde hay que dirigir el paquete.

La conmutación de paquetes permite aumentar la eficiencia de utilización del canal respecto a la conmutación de circuitos, gracias al multiplexado estadístico de los datos. Sin embargo, existen inconvenientes particulares a dicho paradigma de conmutación en el dominio óptico. Actualmente todavía no es posible procesar la información de la cabecera de los paquetes a la velocidad necesaria para tomar las decisiones de enrutamiento requeridas sin tener que almacenar la información del campo de datos de los paquetes**.** Además, tampoco existe actualmente un sistema de almacenamiento temporal (*buffer*) para poder guardar datos en el dominio óptico.

Como alternativa se plantea la posibilidad de implementar retardadores con fibras de retardo (*Fiber Delay Lines*, FDL), pero solo permiten almacenar la luz durante períodos de tiempo directamente proporcionales a la longitud de la fibra utilizada. Así que son necesarias bobinas de fibra muy largas para introducir un pequeño retardo (una fibra de 1Km de longitud sólo permite retardar la transmisión durante 5 μs), por lo que el inconveniente principal es el elevado coste que representa implementar este tipo de técnica.

1.2.3. Conmutación óptica de ráfagas

La conmutación óptica de ráfagas (*Optical Burst Switching,* OBS) permite disponer de un grado de multiplexado estadístico de usuarios superior al de OCS pero sin los elevados requerimientos tecnológicos que exige OPS [3].

Con la conmutación de ráfagas, los datos de los usuarios se agregan en la frontera de la red formando unidades de datos más largas, denominadas ráfagas. Previamente a la transmisión de cada ráfaga se transmite un paquete

de control con el objetivo de configurar los dispositivos intermedios para que la transmisión de la ráfaga se pueda realizar íntegramente en el dominio óptico.

Aunque hay distintas definiciones y variantes de la tecnología OBS, la mayoría de ellas tienen en común dos características principales [4]:

- Los datos del usuario se agrupan y desagrupan en los nodos frontera (*edge nodes*) de la red OBS, formando ráfagas de longitud variable.

- Los canales de datos y de control son independientes. La información de control se envía por un canal (entendiendo canal como longitud de onda) distinto a los usados para la transmisión de la ráfaga, procesándose a nivel electrónico en todos los nodos de la red (tanto los nodos frontera como los nodos de la red troncal). En cambio las ráfagas de datos se transmiten de forma transparente en el dominio óptico.

El funcionamiento básico de las redes OBS se caracteriza por agregar los datos de los usuarios en los nodos frontera, siendo transmitidos como ráfagas a través de la red troncal y siendo desagregados de nuevo en el nodo frontera que conecta la red troncal con los usuarios finales (*egress node*) a los que se entregan los datos. A diferencia de la conmutación de paquetes, las ráfagas no están formadas por una parte de datos con cabecera, ya que el paquete de control realiza la función de cabecera de la ráfaga (compuesta por varios paquetes).

El paquete de control se transmite por un canal de control dedicado, y tiene una longitud muy inferior a la de la ráfaga, de manera que un canal de control es capaz de señalizar varias decenas de canales de datos. El paquete de control se procesa electrónicamente en cada nodo intermedio de la red. Si el nodo dispone de los recursos solicitados, el paquete se transmite hacia el siguiente nodo en dirección al destinatario. Así se crea un camino extremo a extremo para que la ráfaga se pueda transmitir a nivel óptico sin tener que ser procesada en los nodos intermedios.

1.2.3.1. Generación de ráfagas

En los nodos de entrada a la red OBS es donde se generan las ráfagas. Los algoritmos de agregación de ráfagas se clasifican de la siguiente manera:

- Algoritmos basados en tiempo: Agrupan los paquetes que han llegado durante un intervalo de tiempo, establecido por un temporizador.
- Algoritmos basados en la longitud de la ráfaga: Se establece un umbral que define la longitud mínima de la ráfaga.
- Algoritmos híbridos: Combinan las dos técnicas anteriores: tiempo y longitud. La ráfaga se crea cuando se supera un umbral de longitud mínima o cuando se excede el tiempo establecido por el temporizador.
- Algoritmos adaptativos: El umbral mínimo y/o el temporizador se modifican dinámicamente en función de las condiciones instantáneas de la red. Proporcionan un rendimiento más elevado y complejo.

CAPITULO 2. INTERCONEXIÓN DE LA RED EPON CON LA RED OBS

2.1. Introducción

La figura 2.1 muestra la interconexión de una red EPON con una red OBS. En la red EPON, los paquetes de información se envían hacia el OLT, el cual, en caso de determinar que el destino está fuera de la red EPON, los envía hacia la red OBS.

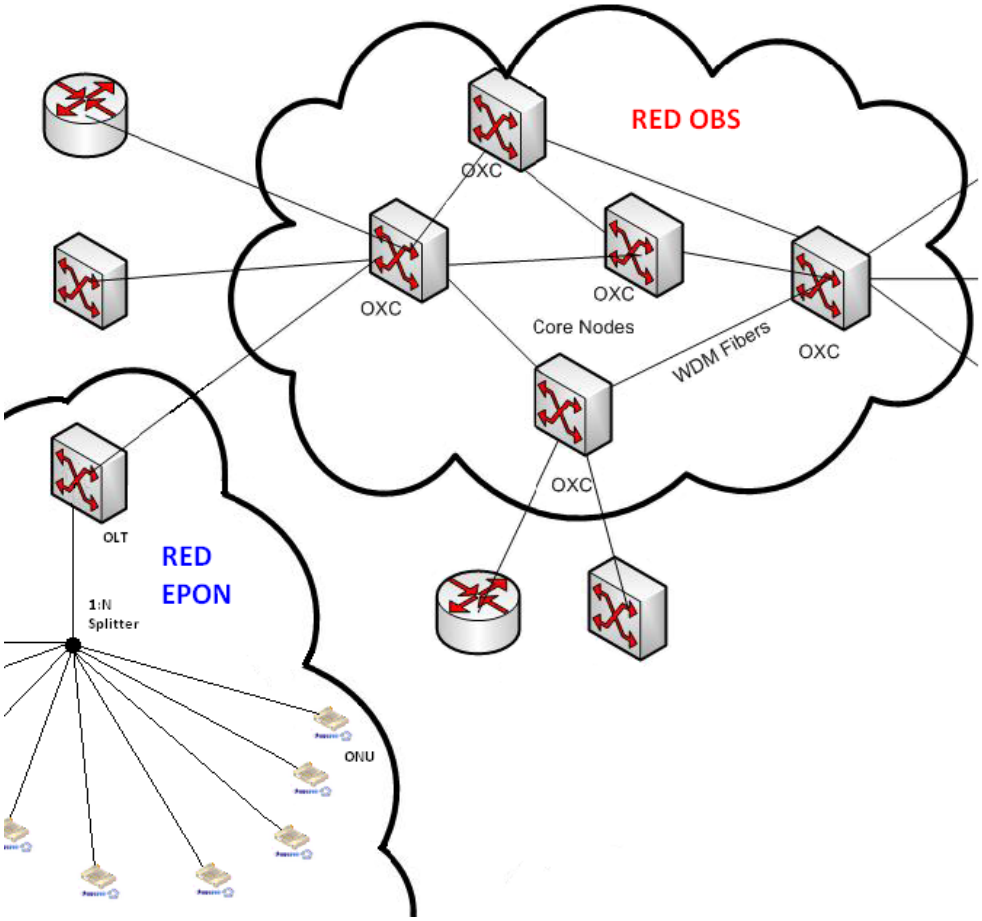

Fig. 2.1. Interconexión de redes.

En una red EPON, el algoritmo DBA es el encargado de repartir el ancho de banda del enlace ascendente (compartido por todas las ONUs). Existen diferentes tipos de algoritmos DBA. La tabla 1 resume las principales características de algunos de los más utilizados [1].

Tabla 2.1. Comparativa de algoritmos DBA

DBA	Tipo	Planificador Intra-ONU	Planificador Inter-ONU	QoS	SLA
IPACT	Centralizado		X		
DBA multimedia	Centralizado		X	X	
DBA con QoS	Centralizado		X	X	X
BGP	Centralizado		X	X	X
DDSPON	Distribuido	X		X	X
DBA descentralizado	Distribuido		X	X	
Cyclicpolling	Centralizado		X	X	X
DBAM	Centralizado	X	X	X	X

Para seguir con el estándar IEEE 802.3ah, el intercambio de información de control entre las ONUs y el OLT para poder realizar el reparto de ancho de banda y comunicar la información a las ONUs debe realizarse utilizando los mensajes *gate* y *report* estándar.

2.2. DBA OBS-aware

En [5] se propone un DBA OBS-aware cuyo objetivo es minimizar el retardo extremo a extremo cuando la red EPON se conecta a una red OBS.

El DBA OBS-aware se ejecuta en el OLT y requiere que las ONUs especifiquen en sus mensajes *report* la cantidad de bytes que quieren transmitir hacia cada destino OBS. Para que esto sea posible, cada nodo de la red OBS se identifica mediante un número de 5 bits denominado APID (*Access Provider IDentifier*). Según el estándar 802.3ah, cada ONU puede definir hasta 8 colas de 8 prioridades distintas. En una red EPON con DBA OBS-aware, cada una de las 8 colas de una ONU se subdivide a su vez en un máximo de 32 subcolas (que es el máximo número de APIDs que se pueden definir con 5 bits). La figura 2.2 muestra la estructura de colas de una ONU cuando se quiere utilizar el DBA OBS-aware.

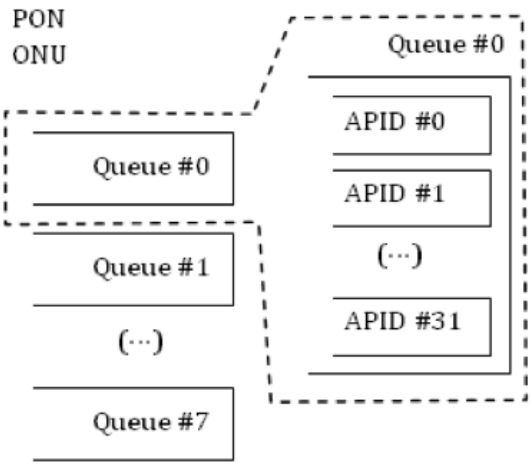

Fig. 2.2. Estructura de colas en una ONU.

Para no modificar el estándar EPON en el formato de paquetes de control, el DBA OBS-aware se adapta al formato de los mensajes *report* y *gate* estándar y sólo modifica el significado del contenido de algunos campos:

- Report: Mensaje que envían las ONUs al OLT para solicitar el envío de una determinada cantidad de datos. En este paquete se indica la cantidad de bytes para cada una de las colas, para ello se especifica el campo *number of queues set*, asociado a las distintas prioridades de un mismo APID, además del campo *report bitmap* que indica para cada *queue set* de que colas se hace la solicitud de envío de datos.

- Gate: Mensaje de control enviado desde el OLT a las ONU. Existen 2 tipos de mensajes *gate*:

    - *Normal gate*: Sirve para que el OLT informe de la cantidad de bytes de información que puede enviar cada ONU y en qué instante de tiempo se asigna esa transmisión. Este mensaje se genera a partir de toda la información recibida por el OLT de los *reports* recibidos. Cada *gate* puede contener hasta 4 *grants*, especificando 4 intervalos de transmisión distintos.

    - *Discovery gate*: Sirve para sincronizar el OLT y las ONUs.

El proceso para realizar la asignación de ancho de banda en el DBA OBS-aware es el siguiente:

Cada ONU especifica en sus mensajes *report* la cantidad de información que tiene almacenada en sus colas para cada APID de cada prioridad (sin superar la longitud máxima de los mensajes *report* que fija el estándar). Con la información de todos los *reports* recibidos, el OLT completa la siguiente tabla:

Tabla 2.2. Recopilación de *reports* en el OLT

	APID1			APID2			…	APID32			RESTO		
	Q0	Q1	Q2	Q0	Q1	Q2		Q0	Q1	Q2	Q0	Q1	Q2
ONU1	$R_1^{0.1}$	$R_1^{1.1}$	$R_1^{2.1}$	$R_1^{0.2}$	$R_1^{1.2}$	$R_1^{2.2}$		$R1^{0.32}$	$R_1^{1.32}$	$R_1^{2.32}$	$R_1^{0.T}$	$R_1^{1.T}$	$R_1^{2.T}$
ONU2	$R_2^{0.1}$	$R_2^{1.1}$	$R_2^{1.1}$	$R_2^{0.1}$	$R_2^{1.2}$	$R_2^{2.2}$		$R2^{0.32}$	$R_2^{1.32}$	$R_2^{2.32}$	$R_2^{0.T}$	$R_2^{1.T}$	$R_2^{2.T}$
…													
ONUn	$R_N^{0.1}$	$R_N^{1.1}$	$R_N^{2.1}$	$R_N^{0.2}$	$R_N^{1.2}$	$R_N^{2.2}$		$R_N^{0.32}$	$R_N^{1.32}$	$R_N^{2.32}$	$R_N^{0.T}$	$R_N^{1.T}$	$R_N^{2.T}$

El cálculo de la ventana máxima para realizar la transmisión se realiza mediante la definición de un tiempo de ciclo máximo, extrayendo la ventana máxima asignable a una ONU.

$$T^{max} = N\left(T_g + \frac{W^{max}}{R_{EPON}}\right) \rightarrow W^{max} = R_{EPON}\left(\frac{T^{max}}{N} - T_g\right) \quad \textbf{(2.1)}$$

T^{max}: Tiempo de ciclo máximo
W^{max}: Ventana máxima (por ONU)
R_{EPON}: Tasa EPON (bps)
N: Número de ONUs
T_g: Tiempo de guarda

El OLT calcula el número de bytes totales que quiere transmitir cada ONU y el número total de bytes que se quieren transmitir hacia cada APID.

R_i: Suma de todos los bytes solicitados por la ONU i.
R_{APID}: Suma de todos los bytes con destino el APID i.

Tabla 2.3. Bytes solicitados por ONU y APID

	APID1			APID2			…	APID32			RESTA			
	Q0	Q1	Q2	Q0	Q1	Q2		Q0	Q1	Q2	Q0	Q1	Q2	
ONU 1	$R_1^{0.1}$	$R_1^{1.1}$	$R_1^{2.1}$	$R_1^{0.2}$	$R_1^{1.2}$	$R_1^{2.2}$		$R1^{0.32}$	$R_1^{1.32}$	$R_1^{2.32}$	$R_1^{0.T}$	$R_1^{1.T}$	$R_1^{2.T}$	R_1
ONU 2	$R_2^{0.1}$	$R_2^{1.1}$	$R_2^{1.1}$	$R_2^{0.1}$	$R_2^{1.2}$	$R_2^{2.2}$		$R2^{0.32}$	$R_2^{1.32}$	$R_2^{2.32}$	$R_2^{0.T}$	$R_2^{1.T}$	$R_2^{2.T}$	R_2
…														
ONU N	$R_N^{0.1}$	$R_N^{1.1}$	$R_N^{2.1}$	$R_N^{0.2}$	$R_N^{1.2}$	$R_N^{2.2}$		$R_N^{0.32}$	$R_N^{1.32}$	$R_N^{2.32}$	$R_N^{0.T}$	$R_N^{1.T}$	$R_N^{2.T}$	R_N
	R_{APID1}			R_{APID2}				R_{APID32}			R_{RESTO}			

Entonces, el OLT clasifica las ONUs como *overloaded* (las que demandan un exceso de datos con respecto a la media) o *underloaded* (las que solicitan la transmisión de un número de datos inferior a la media).

Overloaded (O) → $R_i > W^{max}$
Underloaded (U) → $R_i \leq W^{max}$

Se calcula el exceso total

$$E_{total} = \sum_{j \in U} W^{max} - R_j \quad \textbf{(2.2)}$$

Se realiza la asignación de bytes de exceso a las ONUs *overloaded* (M ONUs)

$$E_i = \frac{R_i}{\sum R_j} \cdot E_{total} \quad \textbf{(2.3)}$$

$$E_i = \frac{E_{total}}{M} \quad \textbf{(2.4)}$$

Y se calcula la ventana total asignada a cada ONU, siendo:

Underloaded (U) → $W_i = R_i$
Overloaded (O) → $W_i = R_i + E_i$

Para informar a cada ONU sobre el número de bytes que puede transmitir para cada APID, se ordenan los APIDs en función del volumen de datos solicitados por todas las ONUs (en orden decreciente los valores R_{APIDi}) y, a cada ONU, en función de su W_i, se le asignan como máximo 3 ventanas para los 3 primeros APID de la lista anterior para los cuales ha solicitado enviar datos. La cuarta ventana corresponde al resto de bytes restantes del W_i asignado inicialmente.

$$W_i^{APID} = \frac{W_i}{\text{Cantidad solicitada}} \quad \textbf{(2.5)}$$

W_i^{APID}: Ventana asignable a un APID

La gestión de las colas se puede definir poniendo un ejemplo en el que una ONU quiere especificar que tiene la siguiente información para transmitir:

- 36000 bytes (3 paquetes) de prioridad 0 + 24000 (2 paquetes) bytes de prioridad 1 del APID #1.
- 48000 bytes (4 paquetes) de prioridad 0 del APID #2.
- 12000 bytes (1 paquete) de prioridad 0 + 24000 (2 paquetes) bytes de prioridad 2 del APID #3.
- 36000 bytes (3 paquetes) de prioridad 1 del APID #4.
- 24000 bytes (2 paquetes) de prioridad 0 del APID#5.

El *report* generado contendrá 5 *queue sets*, uno para cada APID, y especificará 7 tipos de tráfico distinto. El OLT generará un *gate* (con un máximo de 4 *grants*), donde se asignarán 3 *grants* con ancho de banda para 3 tipos de tráfico distinto (diferenciados por APID y prioridad), y asignará en el último *grant* el APID # 31 para que la ONU reparta la capacidad del *grant* entre el resto de tipo de tráfico no asignado en los 3 primeros *grants* del *gate*.

CAPITULO 3. FUNCIONAMIENTO GLOBAL DEL SIMULADOR OPNET

3.1. Introducción a OPNET

Para implementar el escenario de interconexión de las redes EPON y OBS se ha empleado el simulador de redes OPNET Modeler 15.0. Este simulador tiene una interfaz gráfica muy potente y dispone de una extensa librería de modelos. Además, se puede acceder al código de los modelos, modificarlo e incluso desarrollar el código de nuevos módulos.

OPNET se basa en la jerarquía de tres niveles, especificados en la figura 3.1.

- Modelo de red: Define la red y subred del escenario a simular.
- Modelo de nodos: Define la estructura interna de cada elemento de la red o subred.
- Modelo de procesos: define el comportamiento de cada nodo en base a un diagrama de estados (forzados y no forzados).

Fig. 3.1. Jerarquía de OPNET.

Con tal de editar el modelo de nodos, OPNET ofrece un editor y un grupo de nodos predefinidos que son los siguientes:

Tabla 3.1. Módulos del editor de nodos.

	Módulo de procesado.
	Cola (pudiendo tener subcolas albergadas en ella).
	Transmisor punto a punto.
	Receptor punto a punto.
	Transmisor en un bus.
	Receptor en un bus.
	Transmisor por radiofrecuencia.
	Receptor por radiofrecuencia.
	Nodo de comunicaciones por satélite.
	Módulo esys.

También existe un editor de enlaces para interconectar los distintos nodos, los que podemos encontrar predefinidos son:

Tabla 3.2. Enlaces del editor de nodos.

	*Packet Stream*. Son conexiones que llevan paquetes desde un nodo origen a uno destinatario.
	*Static Wire*. Enlaces por los que se envían datos desde un origen a un destino.
	*Logical Associations*. Su función es indicar la relación que tienen dos nodos en la simulación.

Dentro de cada nodo se definen los estados que especifican el funcionamiento del módulo. Este funcionamiento interno de cada uno de los estados así como las transiciones entre ellos, se definen mediante código de programación en C/C++.

De forma análoga al editor de nodos, en OPNET también existe un editor de estados, con las opciones siguientes:

Tabla 3.3. Módulos del editor de estados.

	*CreateState* . Crea un estado dentro del *process model*.
	*Create Transition*. Crea transiciones entre los diferentes estados.
	*Set Initial State*. Establece el estado inicial del diagrama de estados.
	*Edit State Variables*. Define las variables de estado de simulación.
	*Edit Temporary Variables*. Define las variables temporales del *process model*.
	*Edit Header Block*. Permite declarar variables, macros y variables globales que usaran los distintos estados del *process model*.
	*Edit Function Block*. Permite programar las funciones que se use en el process model. Pueden ser llamadas desde cualquier estado.
	*Edit Diagnostic Block*. Contiene las funciones sobre el estado de las simulaciones.
	*Edit Termination Block*. Permite especificar el conjunto de funciones que se ejecutan antes de acabar la simulación.
	Compile Process Model. Permite compilar el código del módulo.

En el diagrama de estados de los nodos se definen dos tipos de estados: los forzados y los no forzados. La diferencia entre ellos es la manera de salir del estado. Los forzados (de color rojo) se quedan en espera de que ocurra algún evento, como por ejemplo que una variable tenga el valor 1 o que llegue una interrupción determinada. En cambio los estados no forzados (color verde) cuando acaban de ejecutar su código salen de su estado para ir al siguiente. La manera que tiene OPNET para establecer un punto de parada en el caso de los estados forzados es a través de 2 tiempos de ejecución en cada estado, denominados:

- *Enter executives*: Parte de código que se ejecuta cuando el proceso entra en ese estado.
- *Exit executives*: Parte del código que se ejecuta en el caso de los estados no forzados cuando se ha acabado de ejecutar el *enter executives*, y en el caso de los estados forzados cuando ocurra algún evento, como por ejemplo el cambio de valor de una variable o la llegada de una interrupción.

Hay distintos sitios donde guardar los ficheros que forman parte de un proyecto en OPNET. Cuando se crea un proyecto, los procesos que se usan se guardan en la carpeta donde se ha generado el proyecto. A excepción de los procesos que se usan en las colas que guardan los ficheros de la capa MAC en *C:\Archivos de programa\OPNET\15.0.A\models\std\ethernet*, y los ficheros de las colas donde se clasifica el tráfico en *C:\Archivos de programa\OPNET\15.0.A\ models\std\base*.

CAPITULO 4. IMPLEMENTACIÓN DE LA INTERCONEXIÓN EN OPNET

4.1. Objetivos

El objetivo inicial del proyecto era evaluar el retardo en la generación de ráfagas a la entrada de una red OBS cuando se interconecta la red OBS con una red EPON en la cual se utilizaba un DBA IPACT y un DBA OBS-aware. Para ello ha sido necesario revisar a fondo la implementación y el funcionamiento del DBA OBS-aware iniciado en proyectos anteriores.

4.2. Escenario

La figura 4.1 muestra el esquema de la red utilizada. La red EPON se compone de 16 ONUs interconectadas al OLT mediante el *splitter*. Cada ONU tiene dos fuentes de tráfico conectadas. El OLT tiene, a su vez, cinco sumideros de tráfico que actúan de destino de los paquetes.

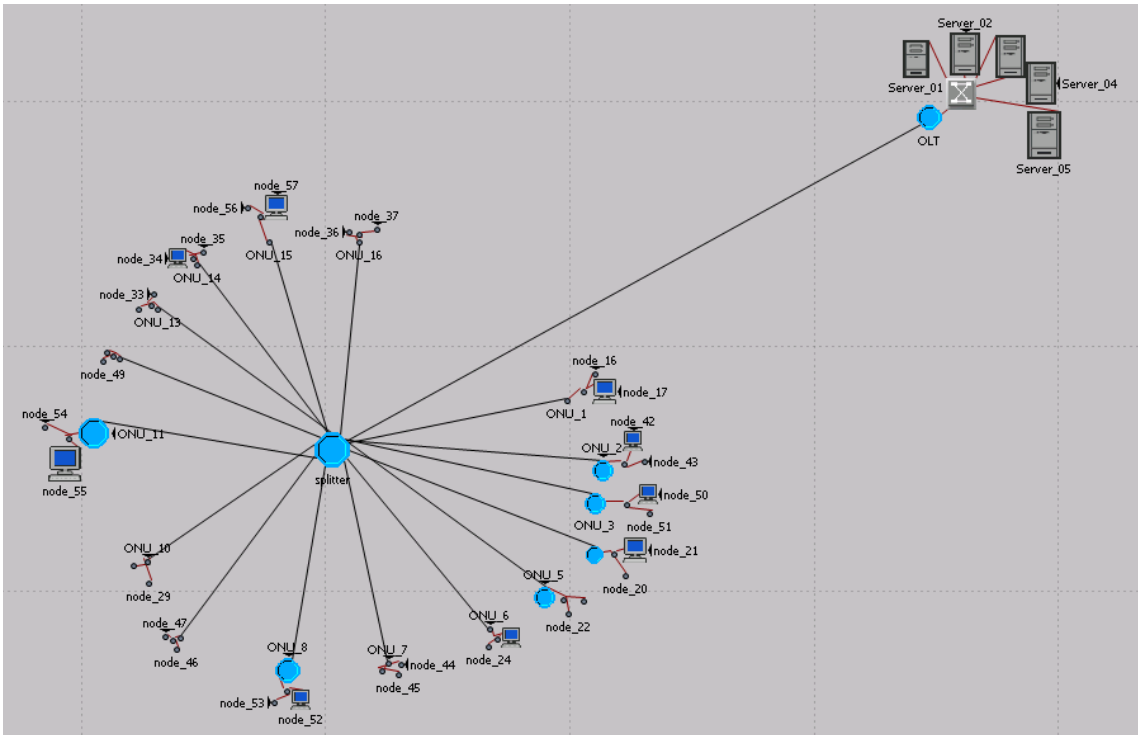

Fig. 4.1. Escenario completo de la red.

4.3. Nodos

La figura 4.2 muestra el diagrama de procesos de una ONU (las 16 son iguales).

Cada mensaje *gate* enviado por el OLT se recibe a través del módulo **rcv** y se envía al módulo **control_parser_discovery** para determinar el tipo de *gate*:

- Si es un *discovery gate* (mensaje de sincronización entre OLT y ONU) se envía al módulo **discovery_process_discovery**.
- Si es un *normal gate* (mensaje con el número de bytes que la ONU puede transmitir y los instantes de transmisión) se envía al módulo **gate_rx_discovery**.

El módulo **gate_rx_discovery** interacciona con el módulo **gate_activation_discovery** para extraer y enviar la información del *gate* al módulo **control_multiplexer_discovery**.

El módulo **control_multiplexer_discovery** interactúa con el **reporting_proc_discovery** para generar cada *report* y con las colas donde se almacenan los paquetes generados por las fuentes conectadas a la ONU.

Los paquetes generados por las fuentes se reciben a través del módulo **rx_0** y se envían al módulo **clasificador** para determinar la cola y subcola a la que pertenece el paquete.

Todos los módulos tienen relación con los módulos **mpcp_clock** y **dba_agent_discovery**. El primero controla los tiempos correspondientes a todos los procesos dentro de la ONU y el segundo permite regular interrupciones de ejecución para pasar de un módulo a otro.

Una vez realizados los procesos en la ONU, la salida de paquetes, tanto de información como de control, se realiza mediante el módulo **tx**.

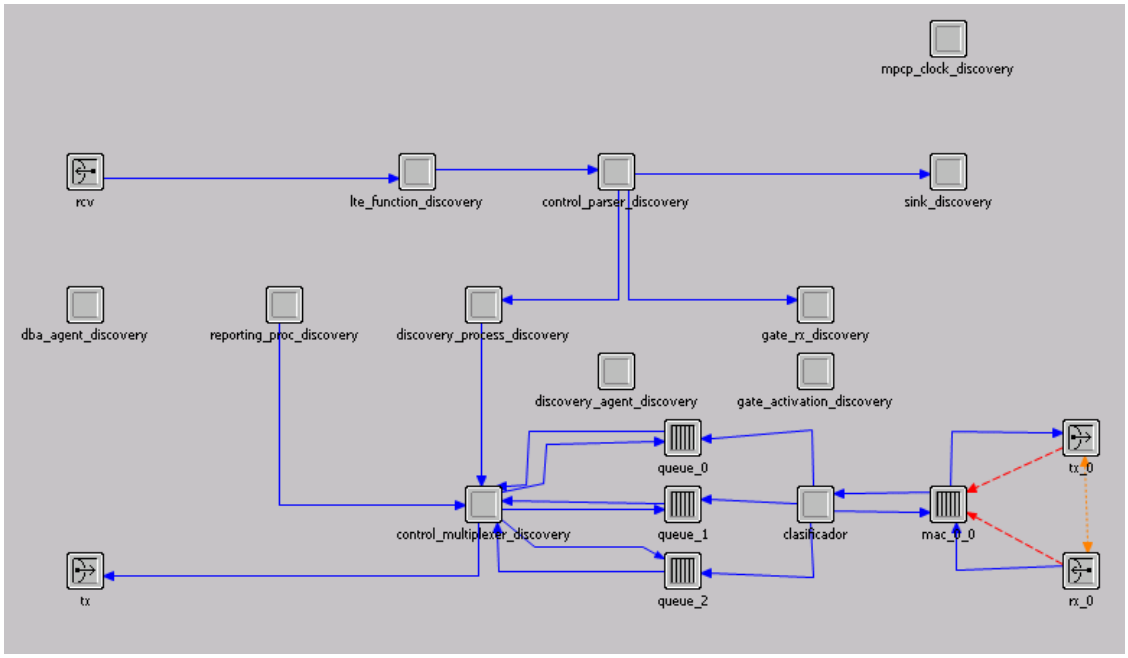

Fig. 4.2. Módulos pertenecientes a la ONU.

Dentro del nodo OLT, el módulo **rcv** recibe los paquetes de información y de control y los envía al módulo **reconciliation_sublayer** encargado de enviar el paquete al módulo **control_parser** de la ONU correspondiente. El procesado del paquete depende de si se trata de un paquete de control o de información:

- Paquete de control: El módulo **report_reception** extrae los valores de los *reports* enviados por las ONUs, activa el módulo **DBA_agent** para calcular la ventana de transmisión, genera el *gate* a través del módulo **gate_generation** y envía el paquete a la ONU correspondiente pasando por el módulo **control_multiplexer** y **tx**.

- Paquete de información: El módulo **clasificador** analiza la información del paquete y lo envía al módulo **q_dest** de salida.

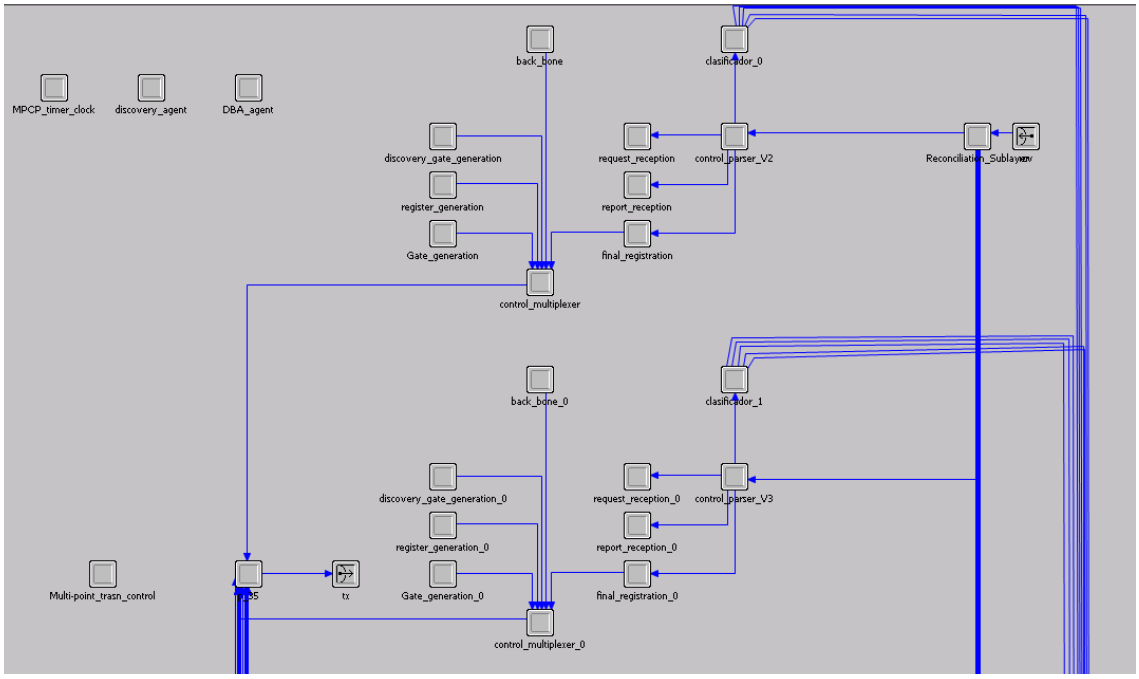

Fig. 4.3. Módulos pertenecientes al OLT (I).

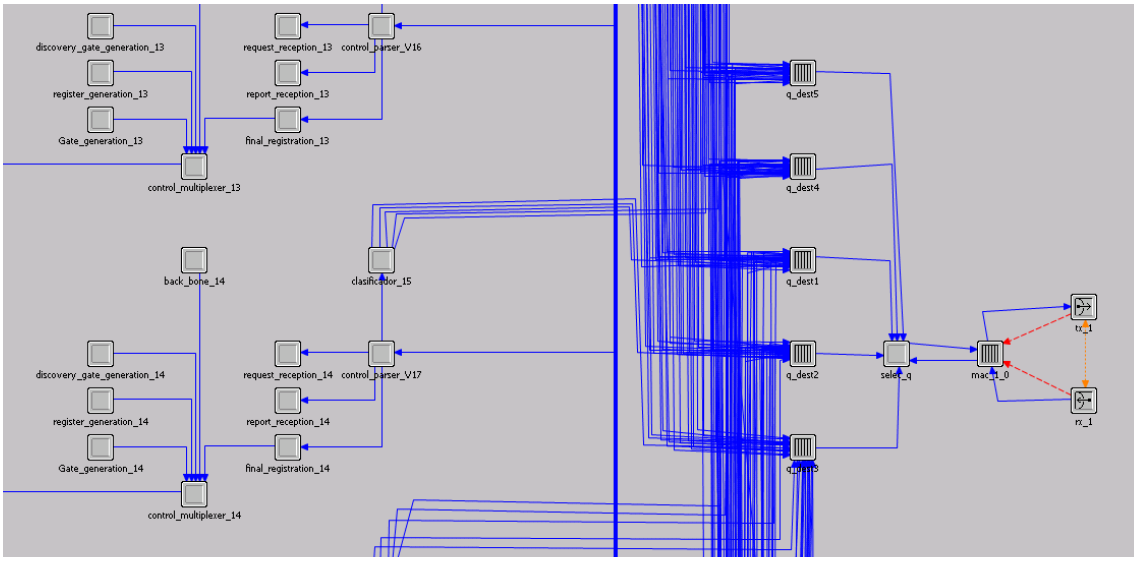

Fig. 4.4. Módulos pertenecientes al OLT (II).

4.4. Modificaciones en módulos

4.4.1. GATE_RX_DISCOVERY

Explicación del módulo:

La función principal de este módulo es esperar la recepción de los *gates*, extraer su contenido y guardar los datos en una estructura llamada *grantList*. Su funcionamiento está relacionado con el **gate_activation_discovery**, módulo encargado de procesar los datos recibidos.

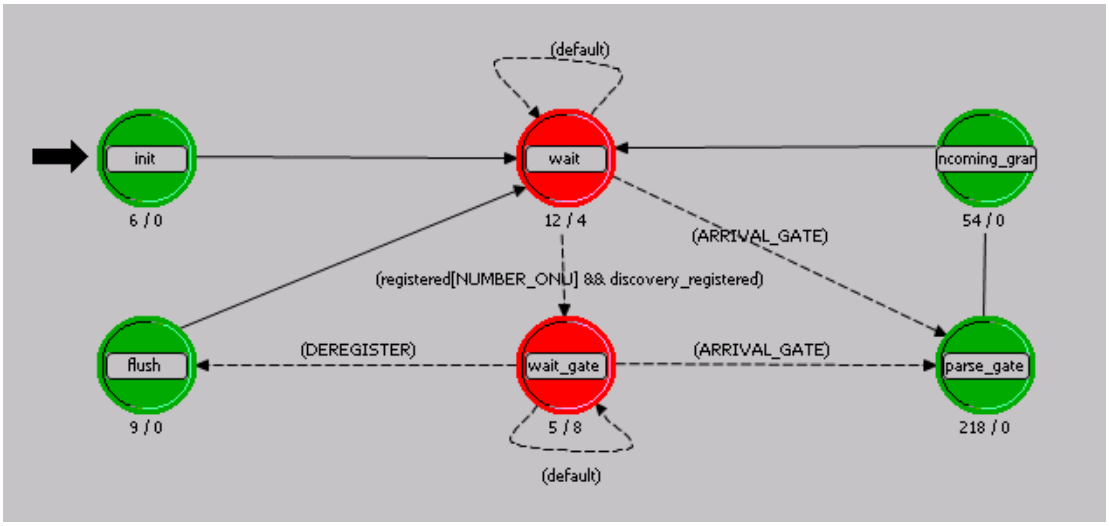

Fig. 4.5. Módulo **gate_rx_discovery**.

Trabajos realizados:

Se corrigió el código del estado **parse_gate** donde se asignaban los mismos valores de APID (identificador del nodo OBS de destino) y de TOS (prioridad) a las variables en dos momentos distintos de ejecución.

Se revisó el método de recogida de los valores APID y TOS especificados en el mensaje *gate*, almacenando sus valores en las variables tos_X[NUMBER_ONU] y apid_X[NUMBER_ONU], donde X corresponde a cada uno de los *grants* de un *gate.* Se detectó que no se asignaba bien el valor de TOS por un error en el módulo **DBA_agent**, que se comentará más adelante.

4.4.2. GATE_ACTIVATION_DISCOVERY

Explicación del módulo:

Este módulo interconecta el módulo **gate_rx_discovery** y el módulo **control_multiplexer_discovery**. Realiza funciones de recogida de valores (almacenados en el *grantList*) y los emplea para indicar datos al módulo **control_multiplexer_discovery** como pueden ser el origen de la información a transmitir o la indicación de los tiempos de envío. Este módulo tiene relación con los módulos **gate_rx_discovery**, **control_multiplexer_discovery**, **DBA_agent** (generación del *report*) y **mpcp_clock** (cálculos de tiempos de transmisión).

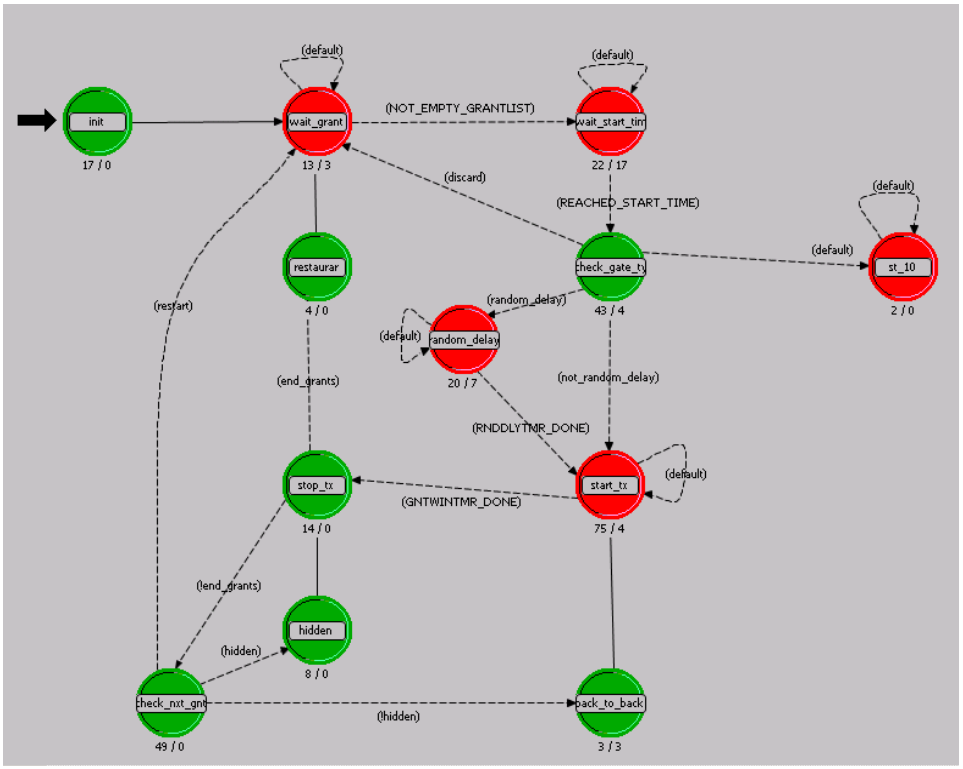

Fig. 4.6. Módulo **gate_activation_discovery**.

Trabajos realizados:

En el módulo **gate_activation_discovery** se tuvo que revisar el cálculo de la variable *effectivelength*. Esta variable indica el tiempo de transmisión que se asigna a una ONU y se calcula como la diferencia entre el tiempo local (localTime) y el tiempo de parada (stopTime). Al comprobar que la variable *effectivelength* tomaba valores positivos y negativos, se analizó el cálculo de la variable stopTime, cuya fórmula es:

```
stopTime[NUMBER_ONU] =  currentGrant[NUMBER_ONU].start +
                        currentGrant[NUMBER_ONU].length -
                        laserOnTime - laserOffTime -
                        syncTime[NUMBER_ONU]
```

Los valores de *start* y *length* dependen del instante de inicio de la ventana de transmisión y del número de bytes que se pueden enviar dentro de la ventana. Se determinó que los valores de estas dos variables no eran correctos y se procedió a realizar las modificaciones para corregirlo.

Inicialmente, para cada *gate* recibido, el estado **start_tx** generaba una única interrupción 590 al módulo **control_multiplexer_discovery** para que la ONU transmitiera los datos especificados en el *gate*. En el DBA OBS-aware, cada mensaje *gate* puede contener hasta 4 *grants* y cada *grant* indica una ventana de transmisión para la ONU asociada a un APID y una prioridad determinada. Así pues, se modificó el estado **start_tx** para que realizara todas las interrupciones 590 necesarias (una por cada *grant*) al módulo **control_multiplexer_discovery.**

Con este cambio, sin embargo, la simulación se interrumpía porque el módulo **control_multiplexer_discovery** no puede quedarse a la espera de varias interrupciones. Así pues, se optó por una solución alternativa en la cuál se redefinieron los vectores *GAD_start* y *GAD_length* que se enviaban al módulo **control_multiplexer_discovery** convirtiéndolos en matrices que almacenarán la información de los distintos *grants* de cada *gate*. Una vez hecha la modificación, se verificó que el módulo **control_multiplexer_discovery** recogiera los datos correctamente para cada *grant*.

4.4.3. CONTROL_MULTIPLEXER_DISCOVERY

Explicación del módulo:

Este módulo gestiona el envío de paquetes, tanto de información como de control. Por lo tanto, interactúa con el módulo de recepción de los *gates* para conocer las colas de las que debe extraer los paquetes a transmitir. Además, debe monitorizar la ocupación de las colas para poder generar los mensajes *report*.

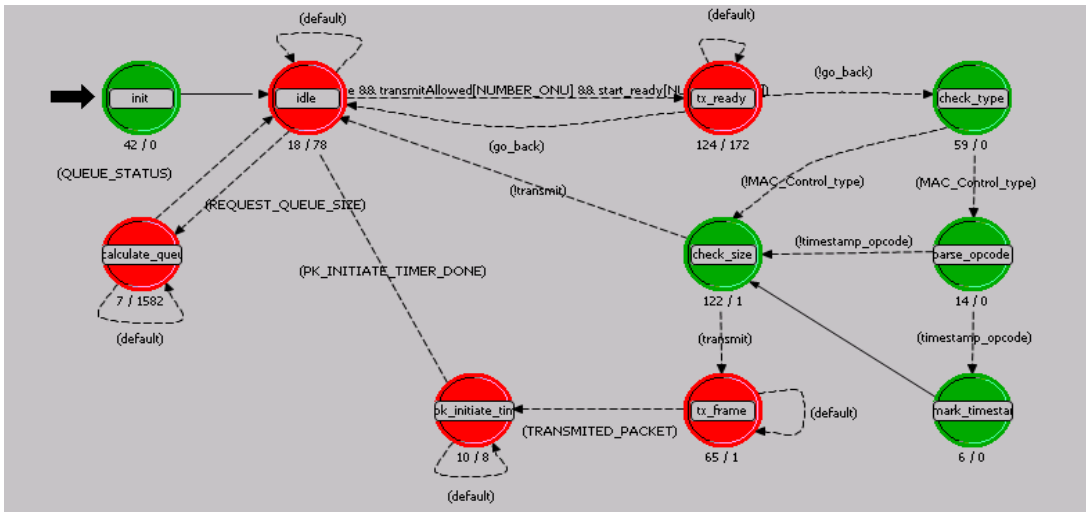

Fig. 4.7. Módulo **control_multiplexer_discovery**.

Trabajos realizados:

En el programa inicial, en el estado **tx_ready** (enter executives), se generaban tres interrupciones 750 (envía paquete), una para cada una de las 3 prioridades consideradas en la implementación de partida.

Luego, para cada paquete que se deseaba transmitir, se modificó el proceso para que se ejecutase el ciclo que se muestra en la figura 4.8:

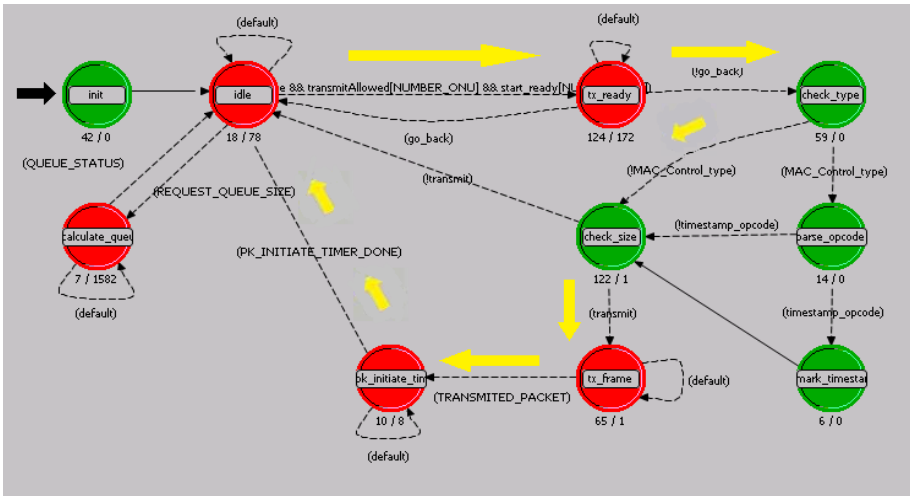

Fig. 4.8. Ciclo ejecución en la extracción de paquetes.

En el DBA OBS-aware, cada *gate* puede contener varios *grants* que permiten sacar paquetes de distintas colas. Para ello, se rediseñó el número de interrupciones, generando una interrupción diferente para cada prioridad (0, 1 y 2, respectivamente):

- Interrupción 750 (ENVIA_PAQUETE_0) → Cola 0.
- Interrupción 751 (ENVIA_PAQUETE_1) → Cola 1.
- Interrupción 752 (ENVIA_PAQUETE_2) → Cola 2.

Otro aspecto que no funcionaba correctamente era el retorno a la cola del último paquete que se intentaba transmitir pero que no se podía enviar porque su longitud era superior a lo que permitía el tamaño final de la ventana indicada en el *grant*.

Para devolver el paquete a su cola correspondiente, se definieron los valores *OUT_TO_QX* (donde *X* es el valor de la cola, 0 1 o 2):

- #define OUT_TO_Q0
- #define OUT_TO_Q1
- #define OUT_TO_Q2

Así, al generar las interrupciones 720 se podían volver a insertar los paquetes en la cola correspondiente con la instrucción:

```
op_pk_send_quiet(pkptr,OUT_TO_Q0);
```

La comunicación *gate-report* implementada en el simulador inicialmente funcionaba de la siguiente manera:

- Generación del primer *gate* en el que se asignan 100 TQs a cada ONU (valor por defecto elegido simplemente para iniciar el diálogo entre OLT y ONUs).
- Recepción del primer *gate* por parte de cada ONU y generación del primer *report* solicitando 67 TQs (valor establecido por defecto cuando no hay paquetes en las colas).
- Envío del segundo *gate* para cada ONU con la asignación de los 67 TQs solicitados.
- A partir de aquí, solicitud y asignación de 90 TQs a cada ONU hasta que llegan paquetes a las colas de las ONUs.
- Interrupción de la simulación por el siguiente error: "*paquet pointer is nil"*, ejecutado en el estado **check_size**.

Para corregir el error fue necesario diferenciar el puntero al primer paquete de cada una de las tres colas y el puntero al mensaje *report*:

- Si *var_activacion* = 5 (cola 0) → Se emplea el puntero *pkptr.*
- Si *var_activacion* = 1 (cola 1) → Puntero *pkptr1.*
- Si *var_activacion* = 2 (cola2) → Puntero *pkptr2.*
- Si *var_activacion* = 3 (*report*) → Puntero *pkptr3.*

Además, se modificó el código que hace referencia a estos punteros, en los procesos: **tx_ready** (exit executives), **check_type** (enter executives), **check_size** (enter executives) y **tx_frame** (enter executives).

A partir de entonces se completaba la simulación pero no llegaban los paquetes al OLT y se detectó un mal funcionamiento en la comunicación entre los módulos **control_multiplexer_discovery** y **queue**.

En el estado **calculate_queue** (enter executives) del módulo **control_multiplexer_discovery** se generaban 3 interrupciones 770 (tamano_cola) que se enviaban una a cada cola, sin hacer distinción entre colas. Para poder distinguir entre colas, se implementaron dos interrupciones más:

- Interrupción 770 (tamano_cola_0) → Cola 0.
- Interrupción 771 (tamano_cola_1) → Cola 1.
- Interrupción 772 (tamano_cola_2) → Cola 2.

Como respuesta a estas interrupciones, se debería informar del estado actual de las colas mediante las interrupciones 940 (*queue_status*). Sin embargo, se recibía sólo 1 de las 3 interrupciones y no se podía conseguir que el estado

calculate_queue se esperase a recibir las 3 interrupciones (porque quedaba en *standby*).

El módulo **queue** genera la interrupción 960 (*envia_paquete*) para volver al módulo **control_multiplexer_discovery**. Sin embargo, el estado **tx_ready** (exit executives), no era capaz de distinguir qué cola había generado la interrupción. Así que, se definieron 3 interrupciones distintas para diferenciar las colas, adaptando todo el código relacionado con la recepción del paquete en el módulo **control_multiplexer_discovery**:

- Interrupción 960 (tamano_cola_0) → Cola 0.
- Interrupción 2020 (tamano_cola_1) → Cola 1.
- Interrupción 2021 (tamano_cola_2) → Cola 2.

Además de distinguir la cola, se requería distinguir el *grant* del *gate*, ya que cada *grant* corresponde a una subcola distinta. Para ello se tuvo que diseñar una estructura de datos y modificar los módulos **gate_activation_discovery** y **control_multiplexer_discovery**.

El sistema de almacenado de datos es el siguiente:

- Se crean vectores para almacenar el momento de transmisión y la duración de los 4 *grants*:
  - Double aux_start[4] → Cada posición almacena el tiempo de inicio (*start)* de cada *grant* del *gate*.
  - Double aux_length[4] → En cada posición almacena la longitud (*length*) de cada *grant* del *gate*.
  - Double aux_stop[4] → En cada posición se almacena el instante de parada (*stop*) calculado a partir de los *start* y *length (aux_stop[x] = aux_start[x] + aux_length[x])* de los vectores anteriores.

- En el estado **idle** se genera el código para guardar estos datos, cuando se genera la interrupción 590 (start) proveniente del módulo **gate_activation_discovery**.

Con estas modificaciones, se empezaron a recibir paquetes en el OLT, pero muchos menos de los esperados. El problema era que las ONUs no interpretaban correctamente los *gates*, porque a la hora de guardar los datos, un *gate* sobrescribía los datos del *gate* anterior cuando aun no se había finalizado la transmisión de los datos autorizados en dicho *gate*. De esta manera se perdía mucho tiempo de transmisión, tal como se muestra en el ejemplo de la siguiente tabla:

Tabla 4.1. Gates gestionados

GATE 1			
GRANT 1:	start 100	length 100	stop 200
GRANT 2:	start 200	length 50	stop 250
GRANT 3:	start 250	length 200	stop 450
GRANT 4:	start 450	length 50	stop 500

GATE 2			
GRANT 1:	start 4000	length 200	stop 4200
GRANT 2:	start 4200	length 100	stop 4300
GRANT 3:	start 4300	length 200	stop 4500
GRANT 4:	start 4500	length 100	stop 4600

GATE interpretado			
GRANT 1:	start 100	length 100	stop 200
GRANT 2:	start 200	length 50	stop 250
GRANT 3:	start 4300	length 200	stop 4500
GRANT 4:	start 4500	length 100	stop 4600

Se diseñó un sistema de almacenado de los datos en variables distintas para poder almacenar la información contenida en dos *gates* consecutivos. Este diseño fue implementado en el estado **idle** (exit executives) cuando se genera la interrupción 590 (start) proveniente del **gate_activation_discovery** y para ello se definieron dos vectores:

- Alternador_rx[NUMBER_ONU]: Permite identificar dónde guardar las variables de los *gates* recibidos y puede tomar dos valores (1 o 2). Para guardar los datos de los *grants*, se duplican las siguientes variables.
  - Para aux_start[4] → aux_start1[4] y aux_start2[4]
  - Para aux_stop[4] → aux_stop1[4] y aux_stop2[4]
  - Para aux_length[4] → aux_length1[4] y aux_length2[4]

- Alternador[NUMBER_ONU]: Este vector indica para cada ONU qué alternador usar (posición 1 o 2) en tiempo de ejecución, para que se acceda a *aux_start1*, *aux_stop1* y *aux_length1* o a *aux_start2*, *aux_stop2* y *aux_length2*, respectivamente.

Así se consigue recibir un *gate* y guardar sus valores en la posición de *alternador_rx 2* mientras en tiempo de ejecución se emplean las variables almacenadas en *alternador_rx 1*.

En la implementación inicial, el *report* se generaba antes de guardar todos los datos del *gate* (porque no se contemplaba la posibilidad que hubiese más de un *grant* en cada *gate*). Para enviar el *report* al finalizar la lectura del *gate* fue

necesario forzar que la interrupción 590 (start) se generara sólo en el último tiempo de *grant* asignado por el *gate* en el **gate_activation_discovery**, antes de que se active la interrupción 730 (queue_size) en el estado **calculate_queue** del módulo **control_multiplexer_discovery** para que concluya el proceso de generación del *report.*

Inicialmente se implementó que fuera el **control_multiplexer_discovery** quien decidiera:

- En el estado **calculate_queue**, se incluyó código para que se generara la interrupción 730 (necesaria para generar el *report*) sólo cuando la simulación estaba en el ultimo *grant* del *gate*.

- Posteriormente, se implementó que la interrupción 730 se generara en el estado **idle** (después de la recepción de las 4 interrupciones 590 del **gate_activation_discovery**).

En ambos casos las ONUs después de realizar las primeras 4 interrupciones 590, interrumpían la simulación.

Se decidió modificar el proceso de recepción de los *gates* y eliminar los alternadores.

Se rediseñaron de nuevo las interrupciones recibidas desde las colas en el estado **tx_ready** (anteriormente definidas como 960, 2021 y 2022), ya que era necesario saber la subcola a la que pertenecían los paquetes.

Así, cada interrupción recibida en el estado **tx_ready** se determinó que tendría la numeración 30XY, dónde X indica el número de cola (0, 1 o 2) y la Y indica el número de subcola (0, 1, 2, 3 o 4). En el módulo **control_multiplexer_discovery** las modificaciones realizadas sirvieron para la correcta recepción de las interrupciones y posterior proceso de salida de información.

Siendo el funcionamiento esperado en los procesos de interrupción de colas y salida de la información, se modificó el código para que se seleccionara el paquete según la información proveniente del *gate*. En el proceso de envío de interrupciones hacia la cola para sacar el paquete del estado **tx_ready** (enter executives) se añadió código que permitiera diferenciar las colas por tipo de servicio mediante las variables tos_1[NUMBER_ONU], tos_2[NUMBER_ONU], tos_3[NUMBER_ONU] y tos_4[NUMBER_ONU] especificadas para cada *grant*. Y para saber el *grant* en el que estaba la ejecución se creó una variable ngrant[NUMBER_ONU] qué lo indicara. En la figura 4.9 se puede observar el paquete y los diferentes *grants* a los que se acceden durante el proceso.

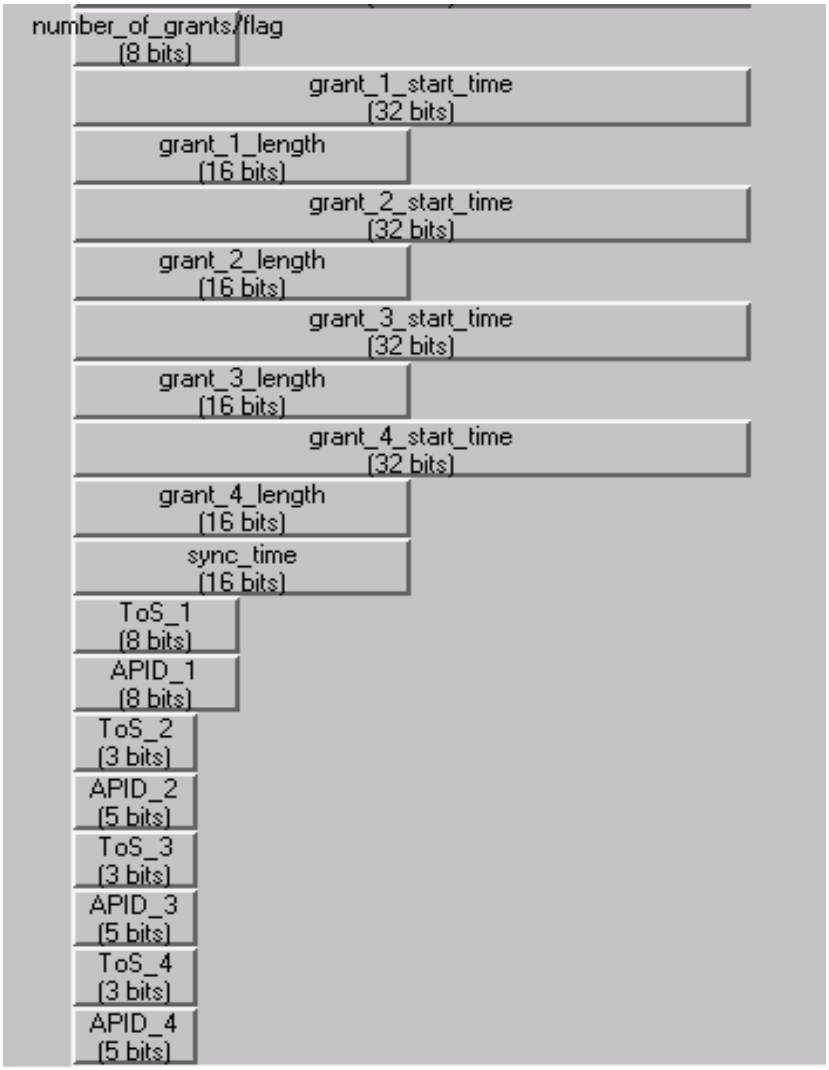

Fig. 4.9. Estructura de *grants*.

La variable *ngrants* se debe actualizar en el estado **tx_ready** (enter executives) para comprobar si el punto actual de simulación ha superado el tiempo establecido para el cambio de *grant*. Cuando se supera el *grant*, se incrementa una unidad (tomando valores módulo 5):

```
If ((aux_stop1[ngrant[NUMBER_ONU]])*16e-9 <= op_sim_time())
        ngrant[NUMBER_ONU]++;
```

La constante 16^{-9} permite pasar de TQs a unidades temporales para que aux_stop1 y op_sim_time() tengan el mismo formato.

La variable *ngrants* se definió como externa, porque las colas deben saber que TOS (indica la subcola) se encuentran en el *grant* de simulación actual.

4.4.4. REPORTING_PROC_DISCOVERY

Explicación del módulo:

Este módulo tiene la función de generar el mensaje *report*, a través de la interrupción enviada desde el estado **gen_report** del módulo **DBA_agent.**

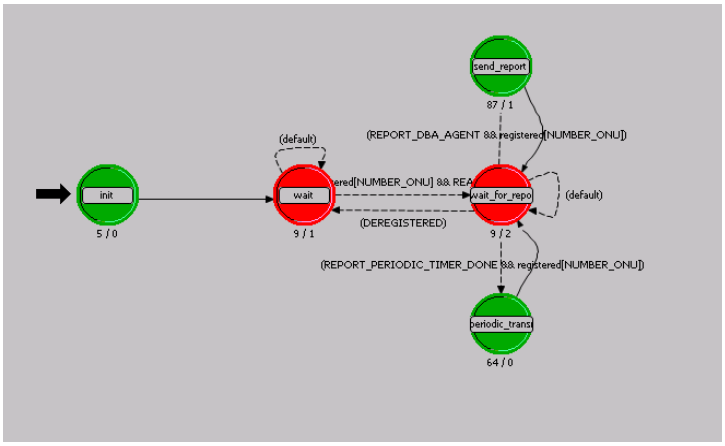

Fig. 4.10. Módulo **reporting_proc**.

Trabajos realizados:

La simulación se paraba porque se generaban *reports* solicitando 0 TQs provocados por la interrupción 410 (report_periodic_timer_done).

En el estado **periodic_transmission** el código especifica que se genere un *report* vacío en caso que el estado **wait_for_report** no reciba la interrupción 110 (report_dba_agent) en un tiempo de 50ms.

Para poder evitar este proceso que provocaba la finalización de la comunicación, se establece un valor fijo mínimo para mantener un espacio de reserva para que la ONU no se desregistre y continúe enviando *reports,* manteniendo viva la comunicación OLT-ONU.

4.4.5. COLAS

Explicación del módulo:

La implementación del simulador permite establecer 3 colas de prioridades en cada ONU (de las 8 totales que define el estándar). Cada cola está programada en un módulo QUEUE_X, donde X indica la prioridad. Para utilizar el DBA OBS-aware, cada cola está dividida en 5 subcolas, cada una de ellas

relacionada con un APID (de los 32 APIDs que en total permite definir el mecanismo OBS-aware).

El módulo **queue** tiene dos funciones principales: calcular el tamaño de la cola para la generación de los *report* y almacenar los paquetes de las fuentes. Los paquetes que generan las fuentes son procesados por el módulo **clasificador** que determina a cuál de los tres módulos **queue** debe enviar el paquete.

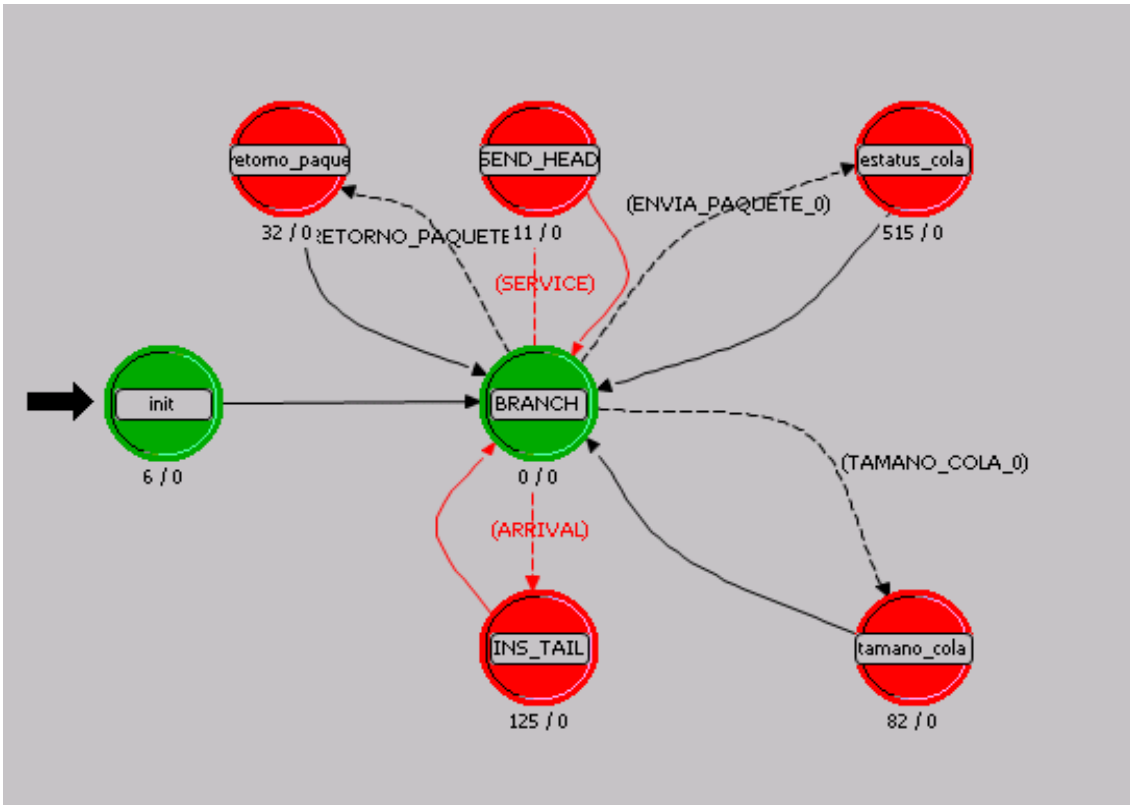

Fig. 4.11. Módulo **queue**.

Trabajos realizados:

En la implementación inicial, las colas de las ONUs se llenaban con los paquetes generados por las fuentes pero no se enviaban paquetes al OLT. Se detectó que el problema radicaba en la generación de un *report* que solicitaba 0 TQs y que provocaba que el OLT no asignara ninguna ventana de transmisión a las ONUs ni enviara el correspondiente *gate*, lo cual, a su vez, impedía que las ONUs generaran *reports* para solicitar el uso del canal de subida.

Así pues, se estableció un tamaño mínimo de cola de 5000 bits con el que se inicializaba la variable queue_size_APIDX[NUMBER_ONU] en el módulo **control_multiplexer_discovery**.

La modificación de las interrupciones que se envían a las colas en el módulo **control_multiplexer_discovery** requirió la modificación del código de

transición al estado **estatus_cola**, adaptando la correcta recepción de las interrupciones 750, 751 y 752 (envía_paquete_x) para cada cola.

Además, en el estado **estatus_cola** (enter executives) se añadieron las interrupciones 960, 2021 y 2022, como respuesta al estado **tx_ready** del módulo **control_multiplexer_discovery**.

Se modificó el proceso que permite extraer los paquetes de las colas en el módulo **control_multiplexer_discovery**. La modificación consistió en generar una única interrupción para la extracción de varios paquetes.

La modificación no proporcionaba resultados adecuados y se decidió crear una nueva estructura de interrupciones en el estado **estatus_cola** (enter executives). La nueva estructura debía permitir identificar la cola y subcola de la cual extraer el paquete en función del número de interrupción recibido por el módulo **control_multiplexer_discovery**. Así, el estado **tx_ready** se programó para recibir interrupciones con la numeración 30XY, dónde X indica el número de cola (0, 1 o 2) y la Y indica el número de subcola (0, 1, 2, 3 o 4):

Tabla 4.2. Tabla interrupciones – colas/subcolas

	COLA 0	COLA 1	COLA 2
Subcola0	3000	3010	3020
Subcola1	3001	3011	3021
Subcola2	3002	3012	3022
Subcola3	3003	3013	3023
Subcola4	3004	3014	3024

4.4.6. DBA_AGENT (OLT)

Explicación del módulo:

El módulo **dba_agent** se encarga de varias funciones:

1. Almacena en una tabla (matriz) las peticiones que las ONUs envían en los mensajes *report*.
2. Calcula el número total de bytes solicitado para cada APID (por ONU y en total).
3. Ordena los APIDs en función del número de bytes solicitados por las ONUs (de más a menos solicitado).
4. Realiza la asignación de bytes a transmitir para cada ONU.

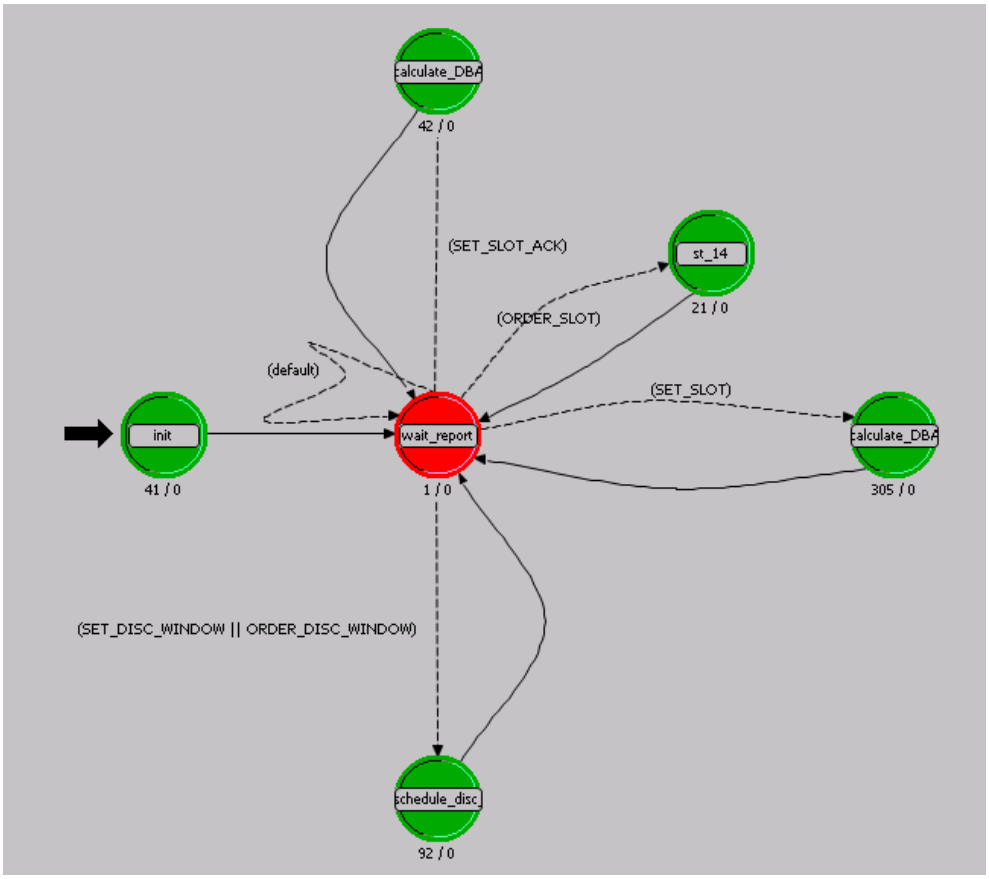

Fig. 4.12. Módulo **dba_agent**.

Trabajos realizados:

Se detectó que todos los *length* (que determinan la longitud de los *grants*) dentro de un mismo *gate* eran iguales y se modificó la asignación. Así, en la expresión:

```
grant_1_length[current_onu]                                    =
tabla_reports[current_onu0][j]+ TREPORT + TON + syncTime +
TOFF;
```

Se sustituyó la variable j por APID[X] (dónde X es la posición del vector de APIDs ordenado previamente por número de bytes solicitado.

Estos valores se guardan posteriormente en las ONUs, de modo que se modificó la variable aux_length[4] para que en el **calculate_queue** (del módulo **control_multiplexer_discovery**) se gestione el *length* correspondiente a cada subcola.

4.4.7. CLASIFICADOR (OLT)

Explicación del módulo:

La única función del módulo **clasificador** es enviar cada paquete a su cola pertinente en función de la dirección destino.

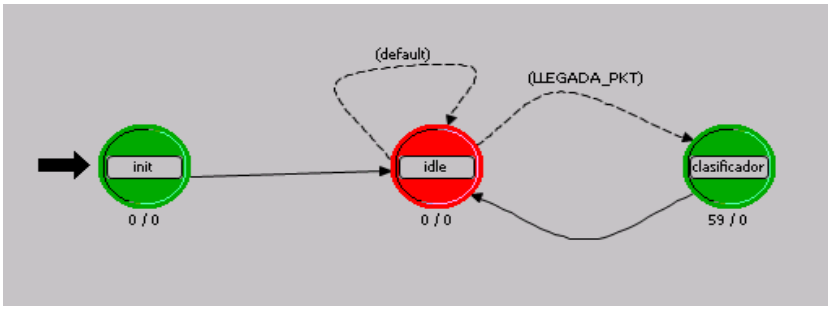

Fig. 4.13. Módulo **clasificador**.

Trabajos realizados:

Para poder implementar 5 colas destino en el OLT se tuvieron que añadir las 5 direcciones destino (ip_addrX) al estado **clasificador**, para enviar el paquete a la subcola de la cola correspondiente.

- Ip_addr0 → TO_Q1
- Ip_addr1 → TO_Q2
- Ip_addr2 → TO_Q3
- Ip_addr3 → TO_Q4
- Ip_addr4 → TO_Q5

CAPITULO 5. RESULTADOS Y GRÁFICAS

Las modificaciones en los módulos indicados en el capítulo anterior han permitido mejorar el funcionamiento del OBS-aware. Se ha conseguido que los paquetes almacenados en las ONUs sean transmitidos según la ordenación del DBA OBS-aware al OLT y, una vez allí, se almacenen en las colas de este nodo en función del APID.

Los resultados que se han podido obtener vía simulación de este escenario muestran que la comunicación entre el OLT y las ONUs, a nivel de mensajes de control, es correcta, tal como muestran las siguientes figuras:

La figura 5.1 muestra un mensaje *report* en el que se solicita ancho de banda para transmitir la información contenida en las colas. El paquete contiene las cabeceras, los campos definidos por el estándar y por último se indica para cada cola y subcola, que cantidad de TQs se almacenan.

```
ID              : 5760
tree ID         : 2817
address         : 0x048581A8
format          : report_aware
creation module : top.Campus Network.ONU_1.reporting_proc_discovery [Objid=1169]
creation time   : 0.013415793881 sec. [0s . 013ms 415us 793ns 881ps]
stamp module    : top.Campus Network.ONU_1.reporting_proc_discovery [Objid=1169]
stamp time      : 0.013415793881 sec. [0s . 013ms 415us 793ns 881ps]
bulk size       : 0 bits
total size      : 584 bits
owner           : top.Campus Network.ONU_1.rx_1 [Objid=1219]
ICI ID          : NONE
ID trace        : off
tree ID trace   : off
encap flags     : NONE
Index Name                    Size I Type             Value
    0 SPD                        8   information      unspecified
1 reserved_0                16   information      unspecified
    2 sld                        8   information      unspecified
3 reserved_1                16   information      unspecified
    4 llid                      16   integer          1
    5 crc_8                      8   information      unspecified
    6 destination_address       48   integer (64 bit) 69,540,876,599,103
    7 source_address            48   integer (64 bit) 1
    8 length/type               16   integer          34,824
    9 opcode                    16   integer          3
   10 timestamp                 32   integer          832,593
   11 number_of_queue_sets       8   integer          5
   12 report_bitmap_0            8   integer          1
13 queue_0_APID_1           16   integer (64 bit) 90
14 queue_1_APID_1           16   integer (64 bit) 859
15 queue_2_APID_1           16   integer (64 bit) 859
   16 report_bitmap_1            8   integer          1
17 queue_0_APID_2           16   integer (64 bit) 90
18 queue_1_APID_2           16   integer (64 bit) 3,166
19 queue_2_APID_2           16   integer (64 bit) 859
   20 report_bitmap_2            8   integer          1
21 queue_0_APID_3           16   integer (64 bit) 90
22 queue_1_APID_3           16   integer (64 bit) 1,628
23 queue_2_APID_3           16   integer (64 bit) 2,397
   24 report_bitmap_3            8   integer          1
25 queue_0_APID_4           16   integer (64 bit) 90
26 queue_1_APID_4           16   integer (64 bit) 2,397
27 queue_2_APID_4           16   integer (64 bit) 1,628
   28 report_bitmap_4            8   integer          1
29 queue_0_APID_5           16   integer (64 bit) 90
30 queue_1_APID_5           16   integer (64 bit) 2,397
31 queue_2_APID_5           16   integer (64 bit) 859
   32 APID_1                     5   integer          0
   33 APID_2                     5   integer          1
   34 APID_3                     5   integer          2
   35 APID_4                     5   integer          3
   36 APID_5                     5   integer          4
   37 pad/reserved               7   information      unspecified
   38 fcs                       32   information      unspecified
```

Fig. 5.1. Paquete *report* capturado.

La figura 5.2 muestra el *gate* enviado por el OLT hacia la ONU como respuesta al *report* de la Figura 5.1. Después de hacer el cálculo para saber qué colas son las que tienen más tráfico entre todas las ONUs, el OLT, selecciona los 3 APIDs y su respectiva calidad de servicio con más tráfico para asignarlos en los 3 primeros *grants* del *gate*. La leyenda de colores, certifica que, para un tráfico seleccionado por el OLT, se asigna la cantidad solicitada por la ONU para ese mismo tipo de tráfico. Entrelazando las figuras 5.1 y 5.2 se comprueba que para el primer *grant* se asigna el APID #1 y el ToS (*Type of Service*) 1, dónde la ONU solicita 859 TQs, y el *length* del *gate* concuerda con esta cantidad. De manera análoga sucede con el 2º y 3r *grant*.

```
Packet content
        ID              : 5817
        tree ID         : 2837
        address         : 0x04881458
        format          : gate_aware
        creation module : top.Campus Network.splitter.PROCESS_1 [Objid=1407]
        creation time   : 0.013581273652 sec. [0s . 013ms 581us 273ns 652ps]
        stamp module    : top.CampusNetwork.OLT.Gate_generation [Objid=26333]
        stamp time      : 0.013510675712 sec. [0s . 013ms 510us 675ns 712ps]
        bulk size       : 0 bits
        total size      : 592 bits
        owner           : top.Campus Network.ONU_1.strm_4 [Objid=1212]
        ICI ID          : NONE
        ID trace        : off
        tree ID trace   : off
        encap flags     : NONE
_____________________________________________________________
Packet Fields
        Index Name                    Size I Type             Value
        ----- --------------------- ------ - ---------------- -------
            0 SPD                        8   information      unspecified
        1 reserved_0                16   information      unspecified
            2 sld                        8   information      unspecified
        3 reserved_1                16   information      unspecified
            4 llid                      16   integer          1
            5 crc_8                      8   information      unspecified
            6 destination_address       48   integer (64 bit) 1
            7 source_address            48   integer (64 bit) 50
            8 length/type               16   integer          34,824
            9 opcode                    16   integer          2
           10 timestamp                 32   integer          844,417
           11 number_of_grants/flag      8   integer          40
           12 grant_1_start_time        32   integer          845,947
        13 grant_1_length           16   integer          859
           14 grant_2_start_time        32   integer          846,806
        15 grant_2_length           16   integer          859
           16 grant_3_start_time        32   integer          847,665
        17 grant_3_length           16   integer          1,628
           18 grant_4_start_time        32   integer          849,293
           19 grant_4_length            16   integer          15,691
           20 sync_time                 16   integer          0
        21 ToS_1                     8   integer          1
        22 APID_1                    8   integer (64 bit) 1
        23 ToS_2                     3   integer          2
        24 APID_2                    5   integer (64 bit) 2
        25 ToS_3                     3   integer          1
        26 APID_3                    5   integer (64 bit) 3
           27 ToS_4                     3   integer          1
        28 APID_4                    5   integer (64 bit) 3
           29 pad/reserved              72   integer          0
           30 fcs                       32   information      unspecified
_____________________________________________________________
Transmission Data Attributes (TDA)
```

Fig. 5.2. Paquete *gate* en respuesta del *report*.

La configuración del escenario parte de:

- Cada fuente está configurada como muestra la figura 5.3.

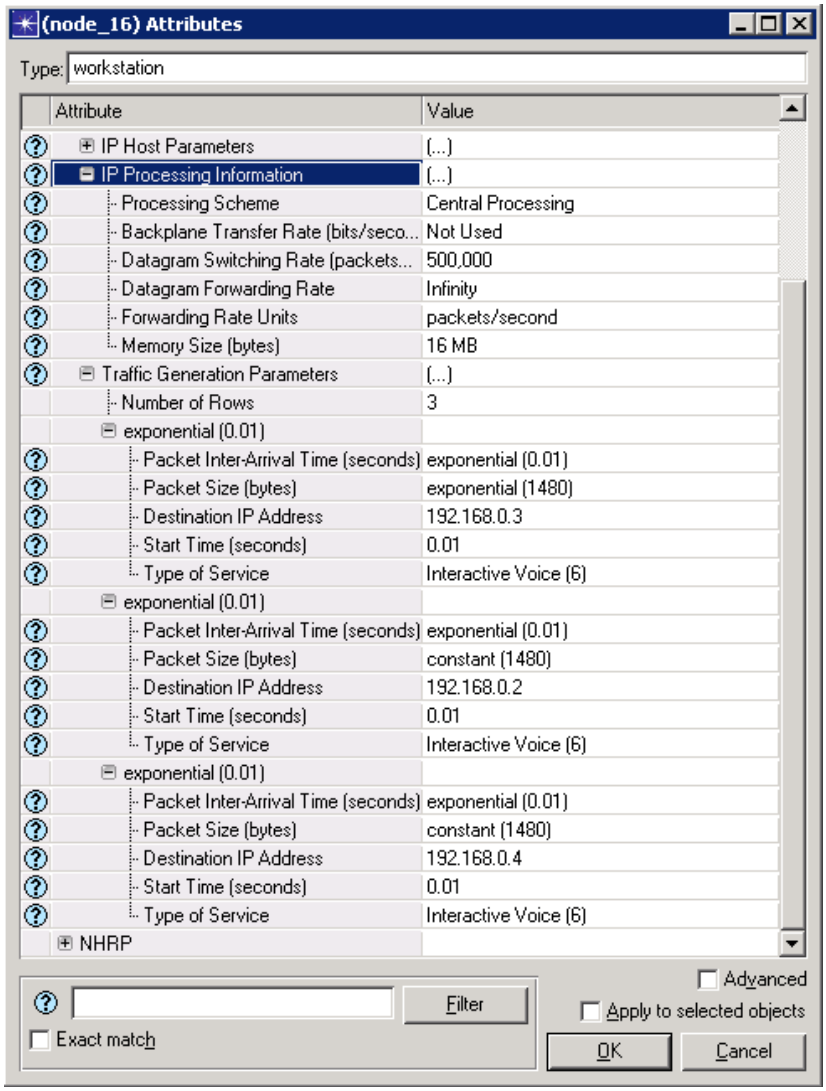

Fig. 5.3. Parámetros de una fuente.

En cada *row* genera 500 paquetes por segundo, existiendo 3 *rows* por fuente y 2 fuentes por ONU. Habiendo un total de 32 fuentes.

- La capacidad del enlace es de 1Gbps.

Dada esta configuración del escenario, el resultado de la simulación se muestra en la figura 5.4, generando el informe de número total de paquetes generados en la simulación.

	Node Name	[Total]	bpdu_format	ethernet_real_epon	ethernet_v2	gate_aware	ip_dgram_v4	report_aware
50	Campus Network.OLT	13940				13940		
51	Campus Network.ONU_1	2689		1342				1347
52	Campus Network.ONU_10	1846		1309				537
53	Campus Network.ONU_11	2708		1368				1340
54	Campus Network.ONU_12	2648		1309				1339
55	Campus Network.ONU_13	2641		1302				1339
56	Campus Network.ONU_14	2671		1332				1339
57	Campus Network.ONU_15	2706		1367				1339
58	Campus Network.ONU_16	1395		1300				95
59	Campus Network.ONU_2	2673		1333				1340
60	Campus Network.ONU_3	2003		1312				691
61	Campus Network.ONU_4	1865		1303				562
62	Campus Network.ONU_5	1595		1264				331
63	Campus Network.ONU_6	1500		1350				150
64	Campus Network.ONU_7	1516		1340				176
65	Campus Network.ONU_8	2670		1330				1340
66	Campus Network.ONU_9	1983		1308				675
67	Campus Network.Server_01	0						
68	Campus Network.Server_02	0						
69	Campus Network.Server_03	0						
70	Campus Network.Server_04	0						
71	Campus Network.Server_05	0						
72	Campus Network.splitter	0						
73	[Total]	112826	54	21169	21349	13940	21169	13940

Fig. 5.4. Generación de paquetes.

Las ONUs generan paquetes *report_aware*, para solicitar ancho de banda para poder realizar los envíos de los paquetes *ethernet_real_epon* (paquetes de información). El OLT genera los *gate_aware* en respuesta a los *reports*. Se comprueba como existe comunicación entre los dos nodos, ya que se responden a las peticiones de ancho de banda. Además, la figura 5.5 demuestra que en el OLT se reciben los paquetes de información que se envían desde las ONUs, quedando almacenados en las colas del OLT.

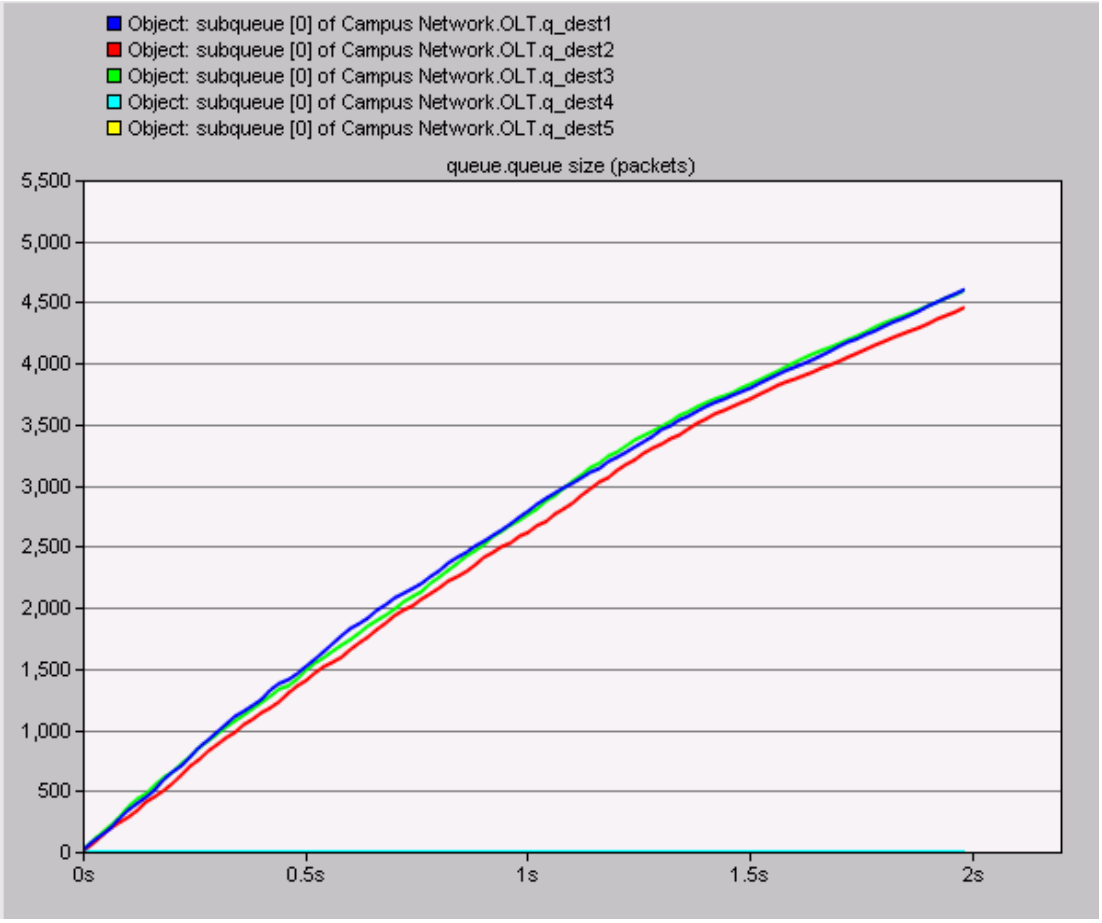

Fig. 5.5. Recepción de paquetes en el OLT.

En la figura 5.5 se muestra, en base al tiempo, la ocupación de las colas del OLT con los paquetes de información que generan las fuentes.
Al no estar implementada la salida de estos paquetes, la gráfica tiende a infinito. Se almacenan paquetes en las 3 colas del OLT (en función de la dirección IP). Con la implementación de la salida de los paquetes hacia sus respectivos destinos de la red OBS, las colas del OLT se vaciarían, tendiendo a tener una ocupación parecida a la de las colas de la ONU, donde se apreciarían picos de llenado y vaciado.

El funcionamiento es el esperado ya que la asignación de los *grants* y del *length* es en función del tráfico que hay para enviar. Asignada en un *grant* la información para el destino con más trafico generado, se asigna ancho de banda para la información del segundo destinatario, y así sucesivamente.

La ocupación de las colas 0 de las ONUs es lo que representan las figuras 5.6 y 5.7 Se almacenan los paquetes recibidos de las fuentes y se transmiten hacia el nodo OLT.

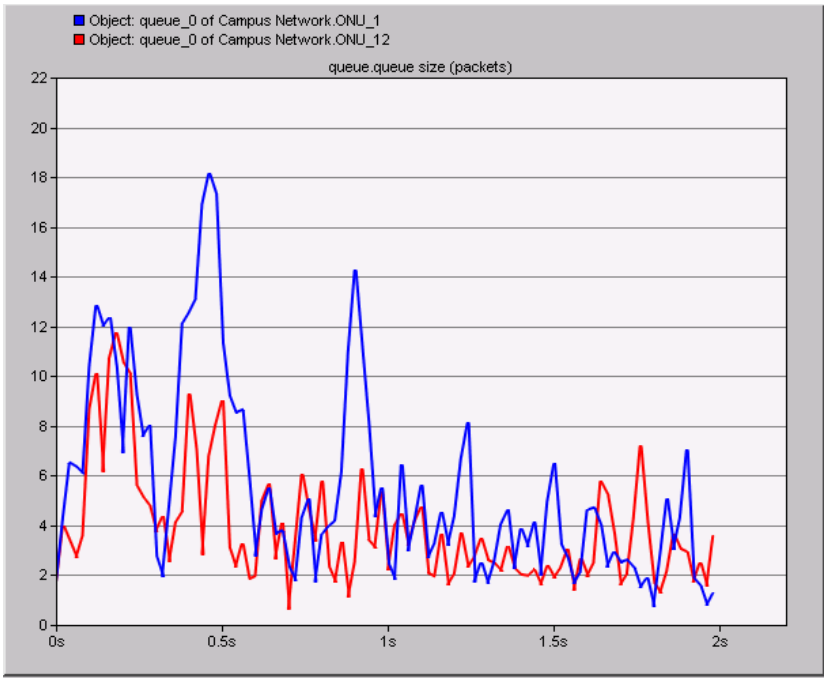

Fig. 5.6. Ocupación de la cola en ONUs 1 y 12.

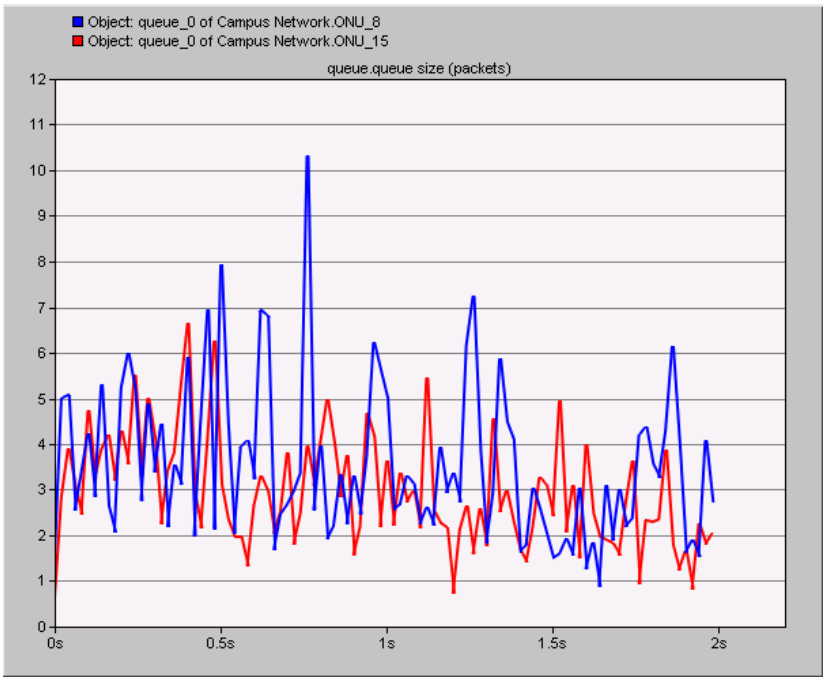

Fig. 5.7. Ocupación de la cola en ONUs 8 y 15.

Las fluctuaciones que hay en las colas son debidas a la funcionalidad de la cola en el sistema, ya que se insertan los paquetes generados por las fuentes y se envían los paquetes que el OLT acepta mediante los mensajes *gate*. Las gráficas de las figuras 5.6 y 5.7 demuestran que el proyecto intercambia la información contenida en los mensajes de control para transmitir la información, y posteriormente ésta es transferida hacia el nodo OLT.

CONCLUSIONES Y ACCIONES FUTURAS

El crecimiento exponencial del tráfico en Internet a lo largo de estos últimos años, ha creado la necesidad de rediseñar las redes de acceso debido a la generación de cuellos de botella, una problemática que previamente había aparecido en las redes troncales.

Una posible solución sería la incorporación de fibra óptica en estas redes. La incorporación de la fibra, tiene un coste elevado en su inicio, pero al implementar redes PON, se puede garantizar un gasto menor a lo largo del tiempo ya que no dispone de elementos activos a mantener, provocando que sea una solución viable.

Para poder realizar la migración a las redes ópticas en las redes de acceso, se necesita una técnica de conmutación adecuada, siendo OBS (*Optical Burst Switching*) la más idónea. Ofrece garantías de transferencia de datos como OCS (*Optical Circuit Switching*), sin desaprovechar tantos recursos cuando no realiza transmisiones de información, permitiendo la posibilidad de realizar conexiones dinámicas y a su vez no está tan limitado como OPS (*Optical Packet Switching*), a la hora de almacenar información de cada paquete, al utilizar un paquete de control capaz de señalizar decenas de canales de datos.

En la red OBS las unidades de los usuarios se agrupan formando ráfagas en la entrada de la red. Se envía un paquete de control con el objetivo de configurar los nodos intermedios y reservar los recursos para poder enviar posteriormente la ráfaga hacia el nodo destino.

A lo largo del trabajo final de carrera se ha estudiado, analizado y evaluado vía simulación (utilizando el simulador de redes OPNET) la implementación del DBA OBS-aware diseñado para minimizar el retardo extremo a extremo en la interconexión de una red EPON y una red OBS.

Descritos los desarrollos realizados en este proyecto, para conseguir el correcto funcionamiento del sistema se propone realizar las acciones siguientes:

- Revisar el estado **dba_agent** en el OLT. Se ha detectado que el funcionamiento no es el idóneo cuando se incrementa el número de colas y subcolas, llegando menos paquetes al OLT de los esperados.

- Implementar más de 3 colas. En el estándar está definido que puede haber un máximo de 8 colas, teniendo en cuenta que la inserción de más colas afecta a los estados del módulo **control_multiplexer_discovery** de la ONU.

- Rediseñar el proceso de generación de los *reports*, ya que restar los paquetes que salen del nodo ONU asignados por el *gate* en el estado **calculate_queue** del módulo **control_multiplexer_discovery**, no es una opción realmente eficiente en términos de procesado de información.

- Implementar del envío de los paquetes que quedan almacenados en las colas del OLT hacia los destinatarios. Estableciendo umbrales máximos de tiempo de permanencia en cola y cantidad de bytes almacenados en cola.

La evolución de este proyecto, desde sus inicios hasta la consecución de las acciones futuras propuestas dará paso a un modelo de funcionamiento deseado y más completo, para el algoritmo DBA OBS-aware que permita tener un retardo todavía menor en la generación de ráfagas entre una red EPON y una red OBS.

BIBLIOGRAFÍA Y REFERENCIAS

[1] Garfias Hernández, P, "*New dynamic bandwidth allocation algorithm analysis: DDSPON for Ethernet passive optical networks*". Trabajo final de carrera.Febrero 2009.

[2]Farahmand, F, Zhang, Q, "CircuitSwitching" 2005

[3] Chen, Y, Qiao, C, Yu, X, "Optical Burst Switching (OBS): A New Area In Optical Networking Research". Mayo 2004.

[4] López Muñoz, F, Reyes Barbancho, J y Agustí Torra, A. "*Estudi i simulació de mecanismes de resolució de contencionsen xarxesòptiques de e commutació de ràfegues (OBS*)". Trabajo final de carrera. Abril 2007.

[5] Luque Ballester, E. "*Estudi i simulació de mecanismes d'assignació d'ample de banda en xarxes EPON"*. Trabajo final de carrera. Desembre 2010.

[6] CLB Tecno Logica SAC. Estudio del impacto medioambiental de un proyecto de instalación de fibra óptica en Arequipa (Perú), en octubre de 2006. http://www.mtc.gob.pe/portal/transportes/asuntos/proyectos/estudios/RE/2007, RD-052.pdf [fecha de último acceso enero de 2012].

[7] Mahecha, C. "*Nuevo chip para banda ancha en redes de fibra óptica y redes eléctricas de QualcommAtheros*" http://zonamovilidad.es/tecnologia/17-tecnologia/2910-qualcomm-atheros-anuncia-el-chip-para-banda-ancha-en-redes-de-fibra-optica-y-redes-electricas [fecha de último acceso enero de 2012].

[8] Fibraopticahoy.com, "*Soluciones eco-sostenibles para reducir el impacto medioambiental en redes de fibra óptica*." http://www.fibraopticahoy.com/soluciones-eco-sostenibles-para-reducir-el-impacto-medioambiental-en-redes-de-fibra-optica/ [fecha de último acceso enero de 2012].

[9] Morey Pascual, B., Naharro Parra, M. i Agustí Torra, A, *"Estudio e implementación de mecanismos de asignación de longitudes de onda para redes OBS sin colisiones."*. Trabajo final de carrera. Enero 2009.

[10] Carvalho, G y Núñez, H, "*Tecnologías de acceso en redes de fibra óptica*" http://www.unefatelecom.com.ve/descargas/1FTTH_PON.pdf [fecha de ultimo acceso noviembre 2011].

[11] Departamento de ciencias de la computación e I.A, "*Acceso al medio. Transmisión de datos y redes de ordenadores*" http://elvex.ugr.es/decsai/internet/pdf/4%20MAC.pdf [fecha de último acceso noviembre 2011].

[12] Kramer, G, "*Ethernet passive optical networks*". McGraw-Hill. Marzo 2005.

[13] Remondo, D. IntensificacióXarxesTelemàtiques. Unidad 17. EETAC

Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

ANNEXOS

TÍTULO DEL TFC: Estudio y simulación del DBA OBS-aware para EPONs

TITULACIÓN: Ingeniería Técnica de Telecomunicaciones, especialidad Telemática

AUTORES: Albert Nogueira Ventura
Rubén Herráiz Buendía

DIRECTORA: Anna Agustí Torra

DATA: 19 de mayo de 2012

IMPACTO MEDIOAMBIENTAL

En este capítulo se pretende orientar sobre el impacto medioambiental que tendría la implementación física del proyecto realizado.

Hay que tener en cuenta que en este proyecto se habla de la interconexión entre dos redes, la red OBS y la red EPON. Actualmente ya existen redes troncales de fibra óptica instaladas, por lo que no se hace necesario realizar un estudio sobre el impacto medioambiental de la red OBS. En cambio sí se deben tener en cuenta las redes EPON.

Así, nos encontramos con que el desarrollo de este proyecto en gran parte es la sustitución de fibra óptica por el par telefónico de las redes de acceso, en algunos casos será la instalación de la fibra paralela al par telefónico y en algunos casos será la instalación como nueva línea. Ello comportará que el estudio a realizar dependa única y exclusivamente de la zona dónde se realice. Los estudios deberán aportar el impacto sobre los diferentes medios que pueden ser afectados.

- **Medio atmosférico:**

En el caso que en la realización del proyecto se hicieran necesarias obras para la construcción de espacios para habilitar los nodos específicos (ONU) o DSLAMs para que los usuarios tuvieran acceso, habría que analizar las emisiones atmosféricas de los vehículos que trabajaran durante la excavación, los vehículos transportadores de materiales industriales desde sus respectivas fábricas y la aportación de material para hacer las cavidades debajo del suelo. Habrá que mantener los equipos y vehículos con las revisiones pertinentes para no sobrepasar los índices de contaminación que las leyes de la zona indiquen.

- **Medio acústico:**

Igual que en el medio atmosférico, los niveles acústicos generados en el intercambio de par de cobre por fibra óptica o en la instalación de la misma paralela al cable de par de cobre, son prácticamente imperceptibles. Por lo que realmente se deben tener en cuenta en aquellos trabajos en los que deban hacerse instalaciones de las cavidades para la instalación de los equipos, ya que la maquinaria y los vehículos son susceptibles de modificar el entorno acústico de la zona donde se realicen las obras. Se deberá intentar mantener los niveles acústicos por debajo de la limitación establecida por la ley donde se realicen dichas obras.

- **Medio acuático:**

Realmente no hay relación con este proyecto ya que normalmente no se hacen instalaciones de redes de acceso de fibra óptica en medios acuáticos.

- **Medio biótico:**

Sólo se deberá tener en cuenta el impacto en el medio biótico en los proyectos que deban hacerse nuevas instalaciones en entornos rurales, en los demás casos no hay previsto una tala de árboles, ni afectación a la flora y fauna de la zona. Así, en zonas rurales deberá hacerse un estudio concreto de cómo puede afectar la nueva instalación a la zona dónde se pretenda realizar, intentando adecuar la ubicación dónde menos perjuicios cause.

- **Medio terrestre:**

Cuando se realice una nueva construcción, habrá que tener en cuenta que se genera contaminación por los residuos que ésta provoca. Además, también hay que tener en cuenta que la fabricación de fibra óptica también genera residuos por lo que habrá que tenerlo en cuenta en todas las instalaciones de nueva fibra que se puedan hacer.

- **Medio social:**

Para este punto habrá que tener en cuenta para las nuevas instalaciones los posibles yacimientos arqueológicos sobretodo en zonas rurales, ya que en nuevas instalaciones en zonas urbanas difícilmente se pueden encontrar yacimientos desconocidos y en muchos casos los espacios donde se harán las nuevas instalaciones no hará falta hacer obra. Además se puede tener en cuenta que se contratará mano de obra tanto cualificada como no cualificada de la misma población para hacer las instalaciones y las obras, por lo que tendrá un impacto positivo en la economía de la población.

- **Coste energético de los equipos:**

Habrá que tener en cuenta también en la instalación de estas nuevas redes EPON, el coste energético que tienen los equipos. En este punto encontramos hoy en día soluciones bastante eficientes para el estándar IEEE 802.3ah como se comprueba en [7], donde podemos encontrar un chip adaptado a las necesidades de una red EPON con un consumo de energía de menos de 600 MW funcionando a pleno rendimiento en una red óptica pasiva de 1Gbps, entre otras aplicaciones.

También encontramos nuevas soluciones eficientes en el campo de la fibra óptica tal como explica en [8]. Donde tenemos un DSLAM FTTB donde el consumo energético es menor a las soluciones presentadas hasta día de hoy y comporta también un beneficio para el usuario al ofrecer un 10% de ahorro de energía comparado con los DSLAM FTTB tradicionales que emplean refrigeración activa gracias a un mecanismo de refrigeración pasiva.

Todo el personal que entre a trabajar en las operaciones de los proyectos que puedan llevarse a cabo en el supuesto caso de ser implementado en la realidad, deberá estar capacitado en temas de prevención, control ambiental y

seguridad industrial, siendo los temas básicos, pero no limitativos los siguientes:

- Educación ambiental.
- Manejo de desechos sólidos y desperdicios de construcción.
- Usos de implementos de seguridad.
- Normativa ambiental y seguridad industrial de la zona de instalación.

Se puede concluir que en la mayoría de casos las obras a ejecutar para la implementación de este proyecto no generarían impactos negativos significativos en las zonas dónde se implementarán. Así podemos concluir que podrían establecerse distintos tipos de actuación dependiendo de la instalación que se requiera:

- Zonas dónde ya esté la instalación de centralitas de par de cobre.

En estas zonas no habrá impacto medioambiental en lo que se refiere a los medios atmosférico, acústico, acuático y biótico. Por lo que los estudios deberán centrarse en el impacto terrestre y social, teniendo en cuenta también el impacto energético. Además éstos no generarán afectación en bienes y predios privados.

- Zonas donde haya que realizar instalación de nuevas centralitas.

En estas zonas el impacto medioambiental será mayor que en el caso anterior. Aquí se deben tener en cuenta factores como el impacto en el medio atmosférico, aunque realmente se puede concluir con que las emisiones atmosféricas son temporales, no peligrosas y de impacto local.

Hay que tener en cuenta también dependiendo de si es zona urbana o zona rural el impacto biótico que tendrá. En una zona urbana no tiene gran impacto, pero en una zona rural habría que ver que afectación tendría en la flora y la fauna.

Es de vital importancia a la hora de realizar el informe económico si se expropiarán terrenos privados para tener en cuenta tanto la valoración del terreno como las afectaciones de predios y bienes cercanos a la zona del proyecto y alrededores.

DESCRIPCIÓN DE ESTADOS

GATE_RX_DISCOVERY

Init:
Obtiene el valor de NUMBER_ONU, first_time=1 y discovery_registered=1

Wait:
Se ejecuta la interrupción 202 a si mismo, en caso que discovery_registered=1, cosa que siempre se produce porque en el estado **init** se pone a este valor y solo se cambia a 0 en caso de pasar por el estado **flush** en caso de hacer un deregister.

Wait_gate:
Se ejecuta cuando se da la condición registered[NUMBER_ONU]&&discovery_registered. Se espera en este estado hasta que reciba una interrupción 330 (deregister) o (arrival_gate) como es lo normal, para pasar al estado **parse_gate**.

Flush:
Se ejecuta en caso de producirse la interrupción 330 (deregister). Añade el valor de 0 en las 4 posiciones de grantList[NUMBER_ONU].start y grantList[NUMBER_ONU].length y además pone el valor discovery_registered=0

Parse_gate:
Se ejecuta cuando se produce la interrupción arrival_gate. Se extraen los valores de los campos que contiene el *gate*. (Descripción: Campo → variable que se usa).
Destination_address → DA[NUMBER_ONU]
Number_of_grants → flag_gate
Grant_1_start_time → start_time[0]
Grant_2_start_time → start_time[0]
Grant_3_start_time → start_time[0]
Grant_4_start_time → start_time[0]
grant_1_length → length_grant[0]
grant_2_length → length_grant[2]
grant_4_length → length_grant[3]
ToS_1 → tos_1[NUMBER_ONU]
ToS_2 → tos_2[NUMBER_ONU]
ToS_3 → tos_3[NUMBER_ONU]
ToS_4 → tos_4[NUMBER_ONU]
APID_1 → apid_1[NUMBER_ONU]
APID_2 → apid_2[NUMBER_ONU]
APID_3 → apid_3[NUMBER_ONU]
APID_4 → apid_4[NUMBER_ONU]
Generación de interrupción 220 (mpcp_timer_done) al módulo **discovery_process_discovery**.

Incoming_grant:
Llega a este estado directamente desde el estado **parse_gate**, verifica los grants y añade los valores de las variables a grantList[NUMBER_ONU].
grantList[NUMBER_ONU].start[valid_grants[NUMBER_ONU]]=start_time[counter-1];
grantList[NUMBER_ONU].length[valid_grants[NUMBER_ONU]]=length_grant[counter-1];
grantList[NUMBER_ONU].tOsi[valid_grants[NUMBER_ONU]]=tos[counter-1];
grantList[NUMBER_ONU].APidi[valid_grants[NUMBER_ONU]]=apid[counter-1];
Envía interrupción 240 (not_empty_grantlist) al módulo **gate_activation_discovery.**

GATE_ACTIVATION_DISCOVERY

Init:
random_delay = 2;
send = 1;
restart = 0;
discard = 0;
deferred_grant = 0;
num_grant[NUMBER_ONU] = 0;
primera = 1;

Wait_grant:
Se ejecuta este estado directamente con transmitAllowed[NUMBER_ONU] = 0 y en caso de ser deferred_grant=1 (no proviene del estado **init**) se ejecuta la interrupción 240 (not_empty_grantlist) a sí mismo y vuelve a asignar el deferred_grant=0. A la salida del estado se hace:
current[NUMBER_ONU] = 0;
first_time = 1;

Wait_start_time:
Se ejecuta este estado cuando se produce la interrupción 240 (not_empty_grantlist) recibida del estado **gate_rx** después de guardar todos los datos del *gate* en el grantList.
Ejecuta en caso de first_time=1 (siempre se da el caso ya que proviene del estado **wait_grant**)
first_time = 0;
discard = 0;
ngrant[NUMBER_ONU]=0;
RemoveHead(ngrant[NUMBER_ONU]);
La función RemoveHead inserta los valores de grantList de start y lenght a GAD
Ejemplo:
GAD_start[aa[NUMBER_ONU]][NUMBER_ONU]=grantList[NUMBER_ONU].start[current[NUMBER_ONU]];
Realiza la interrupcion 510 al módulo **mpcp_clock.**

Check_gate_type:
Se ejecuta este estado cuando se recibe la interrupción 830 (reached_start_time) enviada desde el módulo **mpcp_clock** que indica que ya ha alcanzado el tiempo de iniciar la transmisión.
Al ser siempre una ONU registrada, se ejecuta la interrupción 560 (gen_report) al módulo **DBA_agent** y se comunique con el módulo **control_multiplexer** para comprobar el estado de las colas.
Se toma la decisión de si aplicar o no random delay, y solo se aplica en caso que la ONU no este registrada, por lo que en este proyecto siempre se ejecuta not_random_delay.
En caso de ser una ONU registrada se actualizan las variables:
Not_random_delay=1;
Random_delay=0;
Discard=0;
Y a la salida del estado actualiza otras variables:
Passed_rnddly=0;
Passed_starttx=0;
First_time=1;

St_10:
Este estado no ejecuta ninguna línea de código y únicamente sirve como espera por defecto del estado **check_gate_type** anterior hasta cumplir una de sus condiciones.

Random_delay:
Se ejecuta este estado en caso de que se cumpla la condición random_delay en el estado **check_gate_type**. Se hace el calculo del delay máximo permitido, para que su valor no se extienda mas del timeslot asignado en la ONU. Ejecuta la función start_rndDlyTmr(); que ejecuta a su vez la interrupción 550 (rnddlymr_done).
A la salida del estado actualiza las variables:
Passed_rnddly=1;
Passed_starttx=0;

Start_tx:
Se ejecuta este estado de tres formas posibles.
Cuando recibe la interrupción 550 (rnddlymr_done) del estado **random_delay** o directamente después de pasar por el estado **check_gate_type** y cumplirse la condición not_random_delay o directamente después de ejecutarse el estado **back_to_back**.
Al comprobar Passed_starttx=0; se calcula el stopTime a raíz del startTime y el length recibidos en el *gate* y se actualiza la variable transmitAllowed[NUMBER_ONU]=1 y stream_queue=1;
Se incrementa la variable ngrant[NUMBER_ONU] cada vez que pasamos por este estado hasta llegar al valor de ngrant[NUMBER_ONU]=3 donde se detecta que es el cuarto grant y es el momento de realizar la interrupción 590 (start) al módulo **control_multiplexer**.
Cada vez que se ejecuta este estado se actualiza el valor del effectiveLenght, que se calcula mediante la resta del localTime al valor de stopTime y además

se ejecuta la función start_gntWinTmr(); la cual realiza la interrupción 1000 (gntwintmr_done).

Stop_tx:
Se ejecuta este estado cuando se recibe la interrupción 1000 (gntwintmr_done) del estado **start_tx**.
insideDiscoveryWindow[NUMBER_ONU]=0;
start_ready[NUMBER_ONU];
Se verifican si existen *grants* pendientes, para tomar la decisión de desplazarse a otro estado.

Check_nxt_gnt:
Se ejecuta este estado si hay grants pendientes, calculado en el estado **stop_tx**. Se incrementa el valor del num_grant[NUMBER_ONU] y se revisan los valores siguientes del GrantList. Hace una comparativa entre el stopTime y el nextstopTime para que en caso que el siguiente valor de parada sea superior al actual actualizamos el valor de la variable hidden=0; en caso contrario hidden=1.

Hidden:
Se ejecuta este estado desde el estado **check_nxt_gnt** cuando hidden=1; y envía el proceso al estado **stop_tx**.

Back_to_back:
Se ejectuta este estado desde el estado **check_nxt_gnt** cuando hidden=0; ejecuta la función RemoveHead y actualiza sus valores. A la salida de este estado actualiza las variables:
Passed_startx=0;
Start_ready[NUMBER_ONU]=1;

Restaurar:
Se ejecuta este estado al calcular que no existen *grants* pendientes en el estado **stop_tx** y se actualizan las variables:
Num_grant[NUMBER_ONU]=0;
Stream_queue=0;

REPORTING_PROC

Init:
Obtiene el valor de NUMBER_ONU y de MAC_ONU

Wait:
first_time = 1;

Wait_for_report:
Se ejecuta este estado cuando se cumple registered[NUMBER_ONU] && ready (310), interrupción recibida desde el estado **discovery_process**
Proviene del estado **wait** con first_time=1, ejecuta el start_periodic_timer(); que genera la interrupción 410 (report_periodic_timer_done) a sí mismo (report_periodic_timer_expire*16e-9)
Si recibe la interrupción 110 (report_dba_agent) del **dba_agent**, se detiene el report_periodc_timer, y se resetea el temporizador con op_intrpt_clear_self().

Send_report:
Se ejecuta este estado con la condición report_dba_agent (110) && registered[NUMBER_ONU], interrupción generada en el estado **dba_agent.**
Se llenan los campos del MPCPDU REPORT con la información recibida del estado **dba_agent.**

llid → llid_assigned[NUMBER_ONU]
destination_address →MAC_OLT
source_address → MAC_ONU
length/type → 34824
opcode → 3
number_of_queue_sets → 5
report_bitmap_0 → 1
queue_0_APID_1 → queue0_size_APID1[NUMBER_ONU]
queue_1_APID_1 → queue1_size_APID1[NUMBER_ONU]
queue_2_APID_1 → queue2_size_APID1[NUMBER_ONU]
report_bitmap_1 → 1
queue_0_APID_2 → queue0_size_APID2[NUMBER_ONU]
queue_1_APID_2 → queue1_size_APID2[NUMBER_ONU]
queue_2_APID_2 → queue2_size_APID2[NUMBER_ONU]
report_bitmap_2 → 1
queue_0_APID_3 → queue0_size_APID3[NUMBER_ONU]
queue_1_APID_3 → queue1_size_APID3[NUMBER_ONU]
queue_2_APID_3 → queue2_size_APID3[NUMBER_ONU]
report_bitmap_3 → 1
queue_0_APID_4 → queue0_size_APID4[NUMBER_ONU]
queue_1_APID_4 → queue1_size_APID4[NUMBER_ONU]
queue_2_APID_4 → queue2_size_APID4[NUMBER_ONU]
report_bitmap_4 → 1
queue_0_APID_5 → queue0_size_APID5[NUMBER_ONU]
queue_1_APID_5 → queue1_size_APID5[NUMBER_ONU]
queue_2_APID_5 → queue2_size_APID5[NUMBER_ONU]
APID_1 → 0

APID_2 → 1
APID_3 → 2
APID_4 → 3
APID_5 → 4

Se realiza el envío y se genera la interrupción 1024 (report_generated) al **control_multiplexer.**

Periodic_transmission:
Se ejecuta este estado con la condición report_periodic_timer_done (410) && registered[NUMBER_ONU], interrupción generada en el módulo **reporting_process** en caso de no haber recibido la 110 en el estado **wait_for_report**.
Se rellena un paquete *report* de valores predefinidos, pero con los mismos campos que en el estado **send_report**.

MPCP_CLOCK

Init:
Obtiene el valor de NUMBER_ONU y Recount_localtime=0

Wait_localtime:
Se ejecuta este estado desde el estado **init** o cuando se produce deregistered=1 desde el estado **count.**
Deregistered=0

Count:
Se ejecuta este estado con la interrupción 11 (new_localtime)/new_lt generada en el módulo **gate_rx_discovery**.
La función new_lt resetea el contador cuando el localtime se actualiza y inicia el contador en mitad del periodo si la condición de paridad es verdadera.
En este estado espera una de las posibles interrupciones.
Si Recount_localtime=1 se pone Recount_localtime=0 y genera la interrupción 810 a sí mismo cada 16ns

- Si recibe la 810 (recount) incrementa el localtime
- Si recibe la interrupción 12 (restart) generada en el estado **control_parser** (en la función local_time_onu).
- Si se cumple la condición de paridad se actualiza recount_localtime=0 si no se cumple Recount_localtime=1
- Si recibe la interrupción 510 generada en el módulo **gate_activacion_discovery** (función RemoveHead).
  Interrupt_start = valor de starttime – localtime
  Genera la interrupción 820 (sin nombre)
- Si recibe la 820
  Alcanza el momento de inicio de la ventana de tx
  Genera la 830

- Si recibe la 340 (sin nombre) que se genera en el módulo **discovery_process** (estado **remote_deregistered**) actualiza: Deregistered = 1(condición para volver al estado **wait_localtime**).

Continue_count:
Se ejecuta este estado al recibir la interrupción 810 (recount) y se actualiza recount_localtime=1.

DBA_AGENT_DISCOVERY

Wait:
first_time = 1;

Request_queue:
Se ejecuta este estado mediante la interrupción 560 (gen_report), generada en el módulo **gate_activation_discovery.**
Genera la interrupción 120 (request_queue_size) al módulo **control_multiplexer** para consultar el valor de las colas, proveniente del estado **wait** con first_time=1.

Gen_report:
Se ejecuta mediante la interrupción 730 (queue_size), generada en el módulo **control_multiplexer**.
Genera la interrupción 110 (report_dba_agent) al módulo **reporting_proc** para generar el *report* con el estado de las colas.

CONTROL MULTIPLEXER

Init:
Iniciación de las variables.

Idle:
Se ejecuta al iniciar la simulación. Se queda a la espera de distintas interrupciones. Cuando se genera cualquier interrupción destinada al módulo **control_multiplexer** que no sea la interrupción 120 (request_queue_size) se genera la interrupción 792.
Se actualiza go_back=2 y se mantiene a la espera de una de las siguientes interrupciones:

- AVAILABLE: Actualiza stream_queue = 1.

- START: Interrupción nº 590 que proviene del módulo **gate_activation_discovery**, actualiza start_ready[NUMBER_ONU]=1 y stream_queue=1. Se guardan los valores del start, stop y length del *gate* recibido y se actualiza once=1.

- Variable deferred: Actualiza stream_queue=1.

- Report_generated: Interrupción nº 1024 que proviene del módulo **DBA_Agent** (estado **send_report**). Actualiza report_ready = 1, y en el caso de que once = 1, si currentGrant[NUMBER_ONU].discovery == 0 && report_ready == 0, se modifica stream_queue = 0, y si currentGrant[NUMBER_ONU] .discovery == 0 && report_ready == 1 se modifica stream_queue = 1, report_ready = 0 y once = 0.

Calculate_queue:
Se ejecuta mediante la interrupción 120 (request_queue_size). Segeneran las interrupciones 770 (tamaño_cola_0), 771 (tamaño_cola_1), y 772 (tamaño_cola_2) a las colas para que pasen al estado de calculo de tamaño de sus colas.
Se mantiene a la espera de que se genere la interrupción 940 (del estado **queue_status**) proveniente de las colas. La primera vez que se ejecuta el código se asigna a todas las subscolas un valor de 67, las demás veces se comprueba el TOS y el APID para ver de qué cola se ha sacado la información, para posteriormente restar el length a la subcola pertinente y mantener las subcolas ya que no se han vaciado, mediante el proceso:

- Si tos_1[NUMBER_ONU]==0 (tos_1 indica *grant1* puede tener los valores 0,1,2 y 3 para la cola que toque vaciar.
- Si apid_1[NUMBER_ONU] == 1 (apid_1 indica qué subcola).
- Si queue0_size_APID1[NUMBER_ONU]/16)<aux_length1[0][NUMBER_ONU] se asigna un valor simbólico, queue0_size_APID1[NUMBER_ONU] = 750, sino se le asigna queue0_size_APID1[NUMBER_ONU] = abs((queue0_size_APID1[NUMBER_ONU]/16)-aux_length1[0][NUMBER_ONU]+77), y se deja el resto de subcolas de esa misma cola sin restar el length.

Se hace el proceso de manera que se accederá a 4 colas, ya que hay 4 variables de TOS porque se vacían 4 subcolas al igual que existen 4 *grants* en el *gate*. Finalmente se genera la interrupción 730 (en el estado **queue_size**) al módulo **DBA_agent** para la generación del *report*. Con la interrupción 940 (queue_status) se dirige el proceso al estado **idle**.

Tx_ready :
Para acceder a este estado hay que cumplir la condición stream_queue && transmitAllowed[NUMBER_ONU] && start_ready[NUMBER_ONU], esto sucede cuando cuando se ejecuta el estado **start_tx** del modulo **gate_activation**. Lo primero que se realiza es comprovar si aux_stop1[var[NUMBER_ONU]][NUMBER_ONU]*16e-9) <= op_sim_time(), ya que de ser así se ha acabado el tiempo asignado al *grant*, por lo que se actualiza la variable que indica en qué *grant* se encuentra el proceso, var[NUMBER_ONU] = var[NUMBER_ONU]+1, de manera cíclica porque al llegar a 4 se actualiza al valor 0.

Una vez determinado en qué *grant* se encuentra la simulación se comprueba si es un paquete de información, de ser así, dependiendo del TOS, se genera la interrupción 750 (envía_paquete_0), 751(envía_paquete_1) o 752 (envía_paquete_2), en caso contrario genera la interrupción 777.
Se mantiene a la espera de recibir la interrupción de la cola a la que se ha interrumpido previamente, con un rango comprendido de 3000 a 3004, de 3010 a 3014 o de 3020 a 3024 dependiendo de qué subcola se vacíe. En cada caso se coge el paquete pkptr = op_pk_get (IN_STRM_Q0), y se introduce el campo LLID y asigna valor a la variable var_activation que indica si el paquete viene de una cola (5, 1 o 2) o es un *report* (3), finalmente se asigna go_back = 0 para que pueda salir del estado (hacia el estado **check_type**). En el caso de recibir la interrupción 961 (no hay paquetes en cola) asigna go_back = 1 y stream_queue = 0 para que pueda volver al estado **idle** e iniciar de nuevo el proceso.

Check_type :
Se accede al estado mediante !go_back y se comprueba el length/type del paquete para determinar si es *report* o paquete de información y poder asignar el valor de MAC_Control_type a 1 o 0 respectivamente, se dirige al estado **check_size** en caso de !MAC_Control_type o al estado **parse_opcode** en el caso de cumplirse la condición MAC_Control_type.

Parse _opcode :
Se analiza el opcode para ver el tipo de mensaje de control MPCPDU con el valor op_pk_nfd_get_int32 (pkptr3, "opcode", &control_opcode). Si control_opcode requiere timestamp se envía al estado **mark_timestamp** mediante la variable timestamp_opcode, en caso contrario al estado **check_size**.

Mark_timestamp :
Se introduce en el paquete el valor de timestamp op_pk_nfd_set_int32(pkptr3, "timestamp", localTime[NUMBER_ONU]). Y se dirige al estado **check_size** sin ninguna condición.

Check_size :
Se comprueba el var_activation y se recoge el valor del paquete en bits, nextTxTime = op_pk_total_size_get(pkptr), y se modifica el valor a TQs para posteriormente comprobar que el paquete se puede transmitir dentro del tiempo remanente del *timeslot*, mediante la función nextTxTime_2 = round_up(nextTxTime_2) para posteriormente compararlo if(nextTxTime_2 <= (sur - localTime[NUMBER_ONU])). Indica si el paquete cabe en el *timeslot*, transmit = 1 y cambia al estado **tx_frame**, si no cabe el paquete se modifica la variable transmit = 0, para retornar al estado **idle** y empezar el proceso, además de devolver el paquete a la cola con op_pk_send_quiet(pkptr,OUT_TO_Q0) y ejecutando la interrupción 720 (retorno_paquete) para avisar a la cola.
tx_ready_=1.

Tx_frame :
Se comprueba de que cola/report pertenece y ejecuta op_pk_send(pkptr,OUT_STRM_OLT), a través del módulo **tx.** Finalmente se genera la interrupción 790 (transmited_packet), para cambiar al estado **pk_initate**.
first_time = 1.

Pk_initiate_timer:
Se actualiza first_time=0 y se introduce el delay op_intrpt_schedule_self(op_sim_time()+(defaultDelay*16e-9),710),
generándose la interrupción 710 (pk_initiate_timer_done) al mismo estado.
Se espera la interrupción 710 (que introduce el delay del IFG) y si (currentGrant[NUMBER_ONU].discovery==1), modifica el estado **start_ready** [NUMBER_ONU] = 0 para volver a llegar al estado **idle**, que posteriormente vuelve a pasar al estado **tx_ready** para seguir enviando paquetes, y en el caso que se haya transmitido todo, vuelve al estado **idle** mediante la variable go_back.

QUEUE_0

Init:
Obtiene el valor de NUMBER_ONU y de MAC_ONU
Arrival=0;
Q_index=0;

Branch:
Es un estado simple de transición, sin líneas de código.

Retorno_paquete:
Se ejecuta cuando se recibe la interrupción 720 (retorno_paquete). Comprueba que la cola no esté vacía y entonces se comprueba por el valor del APID a que subcola tiene que retornar el paquete.

Send_head:
Se ejecuta cuando se recibe una solicitud para acceder a la cola.

Estatus_cola:
Se ejecuta cuando se recibe la interrupción 751 (envía_paquete_0). Se obtiene el paquete de la subcola determinada, se fija el valor de llid y se envía el paquete al módulo **control_multiplexer**.
Se comprueba que la cola no esté vacía, además del valor de var[NUMBER_ONU] (0,1,2,3) y el valor del apid_X (0,1,2,3,4) y se envía la interrupción al módulo **control_multiplexer** con identificativo 30XY donde X corresponde al numero de cola y la Y corresponde al número de subcola a la que se ha accedido.
En caso de no cumplirse las condiciones de paquetes en las subcolas se envía una interrupción 961 al módulo **control_multiplexer** para informar que no existen paquetes en cola.

Ins_tail:
Se ejecuta cuando se recibe la interrupción arrival (llegando paquetes de las fuentes).
Se obtiene la dirección destino del paquete y se encapsula el ip_dgram_v4 en una trama Ethernet (Ethernet_real_epon) y se inserta en la subcola adecuada.

Tamaño_cola:
Se ejecuta cuando se recibe la interrupción 770 (tamano_cola_0). Se comprueba que la cola no esté vacía y de que subcola proviene y se determina el numero de paquetes (number_packets_apidX) y el valor del tamaño de la cola (queue0_size_APIDX), donde X será el valor de la subcola.
Se ejecuta la interrupción 940 (queue_status) al módulo **control_multiplexer.**

LISTADO DE INTERRUPCIONES

220: MPCP_TIMER_DONE
Gate_rx_discovery (Parse_gate) → Discovery_process_discovery (remote_dereg)

240: NOT_EMPTY_GRANTLIST
Gate_rx_discovery (Incoming_grant) → Gate_activation_discovery (start_tx)

510: (Sin nombre)
Gate_activation_discovery (wait_start_time) → MPCP_clock (count)

820: MPCP_CLOCK
Mpcp_clock (count) → Mpcp_clock (count)

830: REACHED_START_TIME
Mpcp_clock (count) → Gate_activation_discovery (check_gate_type)

560 GEN_REPORT
Gate_activation_discovery (check_gate_type) → Dba_agent (request_queue)

730: QUEUE_SIZE
Control_multiplexer (calculate_queue) → Dba_agent (gen_report)

110: REPORT_DBA_AGENT
Dba_agent (gen_report) → Reporting_process_discovery (send_report)

1024: REPORT__GENERATED
Reporting_process_discovery (sud_rep) → Control_multiplexer (Idle)

590: START
Gate_activation_discovery (start_tx) → Control_multiplexer (Idle)

1000: GNTWINTMR_DONE
Gate_activation_discovery (start_tx) → Gate_activation_discovery (stop_tx)

120: REQUEST_QUEUE_SIZE
Dba_agent (request_queue) → Control_multiplexer (Idle)

770: TAMANO_COLA
Control_multiplexer (calculate_queue) → COLA (tamaño_cola)

940: QUEUE_STATUS
COLA (tamaño_cola) → Control_multiplexer (Idle)

750/752/752: ENVIA_PAQUETE
Control_multiplexer (tx_ready) → COLA (estatus_cola)

3000/3010/3020…
COLA (estatus_cola) → (Control_multiplexer (tx_ready)

720: RETORNO_PAQUETE
Control_multiplexer (check_size) → COLA (retorno_paquete)

790: TRANSMITED_PACKET
Control_multiplexer (tx_frame) → Control_multiplexer (pk_initiaite_time)

710: PK_INITIATE_TIMER_DONE
Control_multiplexer (pk_initiate_time) → Control_multiplexer (Idle)

DESCRIPCIÓN DE VARIABLES

Gate_rx_discovery

Se utilizan variables para el tratamiento de la información que se recibe en los *gates* que llegan del OLT. Para poder guardar los valores del *gate* se define la estructura grant donde se almacena el start, length, TOS y APID (teniendo en cuenta que hay 4 *grants* en un *gate*).

```
struct estructure_grants{
        unsigned int start[5];
        unsigned int length[5];
        unsigned int tOsi[5];
        unsigned int APidi[5];
};
```

Para guardar todos los datos de los *gates*, existe una estructura para guardar todos los *grants*.

```
extern struct estructure_grants grantList[TOTAL_ONUS];
```

Se realizan variables para recoger los datos del *gate* para posteriormente guardarlos en los *grants*. Estos son:

```
int tos [TOTAL_ONUS]
int tos_1[TOTAL_ONUS]… tos_4[TOTAL_ONUS];
OpT_Int64 apid [TOTAL_ONUS]… OpT_Int64 apid_4[TOTAL_ONUS];
```

OpT_Int64 DA[TOTAL_ONUS]; → Guarda la dirección destino.
int grant_number[TOTAL_ONUS]; → Heredado del ipact, hay 1 *grant* por ONU, para que en OBS pueda funcionar con uno, dos, tres y o 4 *grants* por ONU.
int valid_grants[TOTAL_ONUS];→ Heredado del ipact ya que hay 1 *grant* por ONU, para que en OBS pueda funcionar con uno, dos, tres y o 4 *grants* por ONU.
int mpcp_timer_done[TOTAL_ONUS]; → Condicional para Proceso Discovery.

Variables para la recogida y el guardado de valores:

```
op_pk_nfd_get_int32 (pkptr, "grant_1_start_time", &start_time[0]);
op_pk_nfd_get_int32 (pkptr, "grant_1_length", &length_grant[0]);
op_pk_nfd_get_int32 (pkptr, "grant_2_start_time", &start_time[1]);
```

op_pk_nfd_get_int32 (pkptr, "grant_2_length", &length_grant[1]); → Se recoge del paquete recibido.

```
grantList[NUMBER_ONU].start[valid_grants[NUMBER_ONU]]=start_time[count
er-1];
grantList[NUMBER_ONU].length[valid_grants[NUMBER_ONU]]=length_grant[c
ounter-1];
grantList[NUMBER_ONU].tOsi[valid_grants[NUMBER_ONU]]=tos[counter-1];
```

grantList[NUMBER_ONU].APidi[valid_grants[NUMBER_ONU]]=apid[counter-1];
→ Se asigna a grantList que es externa y se puede usar desde cualquier módulo.

Gate_activation_discovery

Se realizan las funciones de recogida de datos de los *grants*, para cuando llegue al momento de interrumpir del módulo **control_multiplexer**, al módulo **DBA_agent** y empezar a transmitir, tanto paquetes de información como *reports*, para ello se definen las siguientes variables:

extern double GAD_start[4][TOTAL_ONUS];
extern double GAD_length[4][TOTAL_ONUS]; → Variables para transferir los datos del *gate* al módulo **control_multiplexer**.

extern int transmitAllowed[TOTAL_ONUS]; → Variable que controla el tiempo de transmisión.

```
extern struct Grant {
        OpT_Int64 DA;
        unsigned int start;
        unsigned int length;
        unsigned int tOsi;
        unsigned int APidi;
        int force_report;
        int discovery;
 };
```

extern struct Grant currentGrant[TOTAL_ONUS]; → Estructura para poder guardar los datos del grant actual.

Struct Grant nextGrant;→Estructura para sacar los datos del próximo grant, mientras aun se usan los del grant actual. Heredado de ipact.

```
struct estructure_grants{
        unsigned int start[5];
        unsigned int length[5];
        unsigned int tOsi[5];
        unsigned int APidi[5];
};
```

struct estructure_grants grantList[TOTAL_ONUS]; → Estructura para guardar los datos de los *gates*.
extern int current[TOTAL_ONUS];→ Variable para saber el *grant* actual en el que se encuentra la simulación, ya que para procesar la información recibida se debe conocer el actual porque las variables tienen valores distintos en función del *grants*. Posteriormente se accede a apid_1, o apid_2, o apid_3… dependiendo del current[NUMBER_ONU].
extern unsigned int stopTime[TOTAL_ONUS];

extern int insideDiscoveryWindow[TOTAL_ONUS];→Variable utilizada por el Discovery Process.

Variables externas (globales) declaradas den el módulo **gate_rx_discovery**:
int force_report[4];
int valid_grants[TOTAL_ONUS];
int registered[TOTAL_ONUS];

int num_grant[TOTAL_ONUS];
OpT_Int64 DA[TOTAL_ONUS];

Algunas funciones definidas, utilizan las variables para sacar la información tratada para que posteriormente gestione los tiempos y saber de dónde extraer la información.

currentGrant[NUMBER_ONU].DA = DA[NUMBER_ONU];
currentGrant[NUMBER_ONU].start=
grantList[NUMBER_ONU].start[current[NUMBER_ONU]];
currentGrant[NUMBER_ONU].start);
currentGrant[NUMBER_ONU].length=
grantList[NUMBER_ONU].length[current[NUMBER_ONU]];
currentGrant[NUMBER_ONU].length);
currentGrant[NUMBER_ONU].tOsi =
grantList[NUMBER_ONU].tOsi[current[NUMBER_ONU]];
currentGrant[NUMBER_ONU].tOsi);
currentGrant[NUMBER_ONU].APidi =
grantList[NUMBER_ONU].APidi[current[NUMBER_ONU]];
currentGrant[NUMBER_ONU].APidi);

En los estados se recogen valores, se calculan tiempos y se guardan en variables para que otros módulos puedan acceder a ellas.

stopTime[NUMBER_ONU] = currentGrant[NUMBER_ONU].start + currentGrant[NUMBER_ONU].length - laserOnTime - laserOffTime - syncTime[NUMBER_ONU];

GrantActual[NUMBER_ONU]++; → Variable para saber en qué *grant* estamos. Se diferencia de current[NUMBER_ONU] en qué esa variable se usa para saber a qué *grant* acceder del *gate* en la recepción de los datos, y ngrant[NUMBER_ONU] se usa para el proceso que se comunica con el módulo **control_multiplexer**, ya que en mismo tiempo de simulación puede requerirse la recepción de datos de un *gate* con la emisión de paquetes.

effectiveLength = stopTime[NUMBER_ONU] - localTime[NUMBER_ONU]; → Esta variable se usa para el contador start_gntWinTmr(), que generará una interrupción cuando se deba empezar a transmitir paquetes.

En el estado **check_nxt_gen**, se responsabiliza de revisar el next_grant antes de resetear a falso la variable transmitAllowed, al verificar que el siguiente *grant*

no se superponga con el *grant* en curso, de solaparse la transmisión puede continuar hasta el fin determinado por el next_grant.

num_grant[NUMBER_ONU]++;

++current[NUMBER_ONU]; → Para revisar el siguiente Grant del grantList:

nextGrant.DA = DA[NUMBER_ONU];
nextGrant.start = grantList[NUMBER_ONU].start[current[NUMBER_ONU]];
nextGrant.length = grantList[NUMBER_ONU].length[current[NUMBER_ONU]];
nextGrant.tOsi = grantList[NUMBER_ONU].tOsi[current[NUMBER_ONU]];
nextGrant.APidi = grantList[NUMBER_ONU].APidi[current[NUMBER_ONU]];
nextGrant.force_report = force_report[current[NUMBER_ONU]];
nextGrant.discovery = discovery[NUMBER_ONU];
nextStopTime[NUMBER_ONU] = nextGrant.start + nextGrant.length - laserOnTime - laserOffTime - syncTime[NUMBER_ONU];

Control_multiplexer

Se realizan los envíos de paquetes, tanto de información como de *reports*, por lo que tiene interacción tanto con el proceso de recepción de *gates*, como con el proceso de saber cuanta información permanecen en colas, y su posterior extracción. Para gestionar estos procesos se definen y emplean las distintas variables:

OpT_Int64 apid [TOTAL_ONUS]… OpT_Int64 apid_3[TOTAL_ONUS]; → Variables usadas para cuando se accede a la información almacenada en el *grant*, para decidir de qué subcola se extrae el paquete, según el *grant,* la información de que subcola se encontrará en una variable u otra.

Declaracion de Variables Globales:
extern OpT_Int64 queue0_size_APID1[TOTAL_ONUS], queue0_size_APID2[TOTAL_ONUS]… queue2_size_APID5[TOTAL_ONUS]; → Variables usadas para saber qué cantidad de información en bits hay en las colas, para obtener el tamaño en TQs basta con dividir el valor entre 16.

extern int aux_apid[TOTAL_ONUS][15];
extern int aux_tos[TOTAL_ONUS][15];
int transmitAllowed[TOTAL_ONUS];
unsigned int localTime[TOTAL_ONUS];
unsigned int stopTime[TOTAL_ONUS];

double GAD_start[4][TOTAL_ONUS];
double GAD_length[4][TOTAL_ONUS];
double GAD_stop[4][TOTAL_ONUS]; → Se necesitan matrices para diferenciar las 4 posiciones de cada ONU, con un vector cada ONU tiene para guardar un solo start, length y stop y necesitan uno para cada *gate*. Se usan para pasar los datos del módulo **gate_activation_discovery** al módulo **control_multiplexer**.

```
unsigned int tos_1[TOTAL_ONUS];
unsigned int tos_2[TOTAL_ONUS];
unsigned int tos_3[TOTAL_ONUS];
unsigned int tos_4[TOTAL_ONUS]; → Variables para identificar la cola.

extern int ngrant[TOTAL_ONUS]; → Variable usada en el proceso de sacar paquetes de la cola, es necesario saber en qué grant se está para seleccionar correctamente de qué cola y subcola se debe sacar el paquete.

struct Grant {
        OpT_Int64 DA;
        unsigned int start;
        unsigned int length;
        unsigned int tOsi;
        unsigned int APidi;
        int force_report;
        int discovery;
};
struct Grant currentGrant[TOTAL_ONUS];

Las variables definidas se usan para tratar de acceder a la información que corresponde para posteriormente guardarla y poder usarla cuando finalice el temporizador de transmisión de información.

aux_start1[ngrant[NUMBER_ONU]][NUMBER_ONU]=GAD_start[ngrant[NUMBER_ONU]][NUMBER_ONU];

aux_stop1[ngrant[NUMBER_ONU]][NUMBER_ONU]=GAD_start[ngrant[NUMBER_ONU]][NUMBER_ONU]+GAD_length[ngrant[NUMBER_ONU]][NUMBER_ONU];

aux_length1[aa[NUMBER_ONU]][NUMBER_ONU]=GAD_length[ngrant[NUMBER_ONU]][NUMBER_ONU];

nextTxTime = op_pk_total_size_get(pkptr3);
nextTxTime_2 = (int)nextTxTime;
nextTxTime_2 = round_up(nextTxTime_2);
stop= (int)aux_stop1[ngrantNUMBER_ONU]];
if (nextTxTime_2 <= (stop - localTime[NUMBER_ONU]))
```

En el módulo **control_multiplexer**, también se realiza el proceso para calcular cuantos paquetes hay en cola para generar el *report*. En este proceso se tienen en cuanta las variables que se usan para saber cuantos paquetes hay en cola, pero además se usa aux_length1[4][16] para saber que cantidad de información sale de las colas para saber exactamente cuantos paquetes hay en cola.

Reporting_proc_discovery

Declaración de variables globales:
int registered[TOTAL_ONUS];
int num_queue_sets[TOTAL_ONUS];
OpT_Int64 queue0_size_APID1[TOTAL_ONUS], queue1_size_APID1[TOTAL_ONUS]… queue2_size_APID5[TOTAL_ONUS];
→ Variables usadas para guardar los valores del tamaño de colas/subcolas.

En este módulo se pueden encontrar las variables que se usan para generar el *report*, llenando los campos del MPCPDU con información del módulo **DBA_Agent**.

op_pk_nfd_set_int32(pkptr,"llid", llid_assigned[NUMBER_ONU]);
op_pk_nfd_set_int64(pkptr,"destination_address",MAC_OLT);
op_pk_nfd_set_int64(pkptr,"source_address",MAC_ONU);
op_pk_nfd_set_int32(pkptr,"length/type",34824);
op_pk_nfd_set_int32(pkptr,"opcode",3);
op_pk_nfd_set_int32(pkptr,"number_of_queue_sets",5);
op_pk_nfd_set_int32(pkptr,"report_bitmap_0",1);
op_pk_nfd_set_int64(pkptr,"queue_0_APID_1",queue0_size_APID1[NUMBER_ONU]);
op_pk_nfd_set_int64(pkptr,"queue_1_APID_1",queue1_size_APID1[NUMBER_ONU]);
op_pk_nfd_set_int64(pkptr,"queue_2_APID_1",queue2_size_APID1[NUMBER_ONU]);→ se repite el código para todas las subcolas y los 4 report_bitmap.
op_pk_nfd_set_int32(pkptr,"APID_1",0)…op_pk_nfd_set_int32(pkptr,"APID_5", 4);→ se repite el codigo para los 5 APIDs.

Mpcp_clock_discovery

unsigned int localTime[TOTAL_ONUS];

```
struct Grant {
        double DA;
        unsigned int start;
        unsigned int length;
        unsigned int tOsi;
        unsigned int APidi;
        int force_report;
        int discovery;
};
```

struct Grant currentGrant[TOTAL_ONUS]; → Estructura usada para guardar los datos en el *grant* con el mismo formato.

int current[TOTAL_ONUS]; → Para saber en qué *grant* se encuentra el proceso.

recount_localtime = 0 → Variable usada para ir cambiando de estado, se va modificando su valor de 0 a 1.

interrupt_start = abs(currentGrant[NUMBER_ONU].start - localTime[NUMBER_ONU]); → En este estado, lo que realmente se ejecutan son timers, se emplean las variables para definir los tiempos de estos timers.

Queue

Realiza el cálculo del tamaño en cola y el proceso de inserción/extracción de paquetes controlado por el módulo **control_multiplexer** (extracción) y el módulo **clasificador** (inserción). Para estos procesos se definen las siguientes variables:

OpT_Int64 queue0_size_APID1[TOTAL_ONUS] … queue0_size_APID5[TOTAL_ONUS]; → Variables para saber que tamaño hay en cada subcola, los módulos cola 0, cola 1 y cola 2 son independientes por lo que cada cola tiene las variables de su cola.

int llid_assigned[TOTAL_ONUS];

int ngrant[TOTAL_ONUS]; → Variables para guardar los valores de los cuatro *grants* de cada *gate* para el proceso de comunicación con el módulo **control_multiplexer**.

```
if (op_q_empty () != OPC_TRUE) {
if (ngrant[NUMBER_ONU] == 0){
if (apid_1[NUMBER_ONU] == 0){
if (op_subq_empty (0) != OPC_TRUE){
pkptr_temp_queue = op_subq_pk_remove (0, OPC_QPOS_HEAD);
op_pk_nfd_set_int32 (pkptr_temp_queue, "llid", llid_assigned[NUMBER_ONU]);
op_pk_send_quiet (pkptr_temp_queue, 0);
op_intrpt_schedule_remote (op_sim_time (), 3000, op_id_from_name (op_topo_parent (op_id_self ()), OPC_OBJTYPE_PROC, "control_multiplexer_discovery"));
}}}}
```

→ Se comprueban las variables determinadas para saber de qué subcola debe salir el paquete y finalmente se envía hacia el módulo **control_multiplexer** junto con la interrupción pertinente.

```
int number_packets_apid1… number_packets_apid5;
number_packets_apid1 = op_subq_stat (0,OPC_QSTAT_IN_PKSIZE);
queue0_size_APID1[NUMBER_ONU] = op_subq_stat (0,OPC_QSTAT_IN_BITSIZE) + 1344 + ((number_packets_apid1 + 1)*96);
```

→ Este código pertenece al proceso del cálculo de tamaño de subcolas.