Modelado de redes de interconexión de altas prestaciones con OMNeT++

Pedro Yebenes, Jesus Escudero-Sahuquillo, Pedro J. Garcia y Francisco J. Quiles [1]

Resumen—**Las redes de interconexión de altas prestaciones son elementos esenciales en los supercomputadores. De hecho, el rendimiento del sistema depende de la red de interconexión. Para desarrollar nuevas y mejoradas arquitecturas de red y evaluar su comportamiento, se utilizan los simuladores. Sin embargo, desarrollar un simulador específico desde cero es un tarea muy costosa y que consume mucho tiempo. No obstante, existen muchas plataformas para ayudar a los desarrolladores, de entre las cuales OMNeT++ es una de las más populares. En este artículo, se describen los beneficios de OMNeT++ y cómo sus características pueden ser utilizadas para implementar los principales aspectos de un simulador de redes de interconexión: modelado de redes de altas prestaciones, la capa de conmutación, el encaminamiento, los algoritmos de arbitraje, las organizaciones de los buffers y cómo obtener resultados. Además, se muestran unos ejemplos de uso.**

Palabras clave— **Redes de interconexión de altas prestaciones, Evaluación de prestaciones, Herramientas de simulación, OMNeT++**

I. Motivación

LAS redes de interconexión son elementos clave para muchos tipos de sistemas de comunicación y computación: procesadores paralelos masivos, sistemas de redes de área local, routers IP, redes en chip, y clusters de PCs y estaciones de trabajo. En estos entornos, el rendimiento alcanzado por todo el sistema depende en gran medida de el rendimiento logrado por la red de interconexión, especialmente cuando el número de nodos de procesamiento o almacenamiento es elevado. El rendimiento de la red depende de varios factores (topología, encaminamiento, etc.) los cuales deben ser cuidadosamente analizados para conseguir una baja latencia de paquete y un alto ancho de banda de la red.

Actualmente las decisiones de diseño de la red no se basan únicamente en el rendimiento logrado, también se tienen en cuenta otros factores como el coste de la red y el consumo energético[2]. De hecho, la tendencia actual en las técnicas de red de interconexión es proponer diseños costo-efectivos que obtienen unas altas prestaciones, con un consumo muy reducido y unos requerimientos de recursos muy bajos. Ejemplos comerciales de tecnologías de red de interconexión son la especificación Infiniband [1] y Myrinet [2].

En este sentido, uno de los objetivos de la década es lograr niveles de rendimiento medidos en exaflops antes del año 2018, y se espera que el número de nodos crezca mientras se mantiene el consumo energético, por lo que la red de interconexión crecerá en tamaño. Por lo que habrá cambios significativos en la arquitectura de red de interconexión por lo que en primera instancia, las herramientas de simulación [4] serán imprescindibles en la validación y la evaluación de las nuevas arquitecturas de red de interconexión de la próxima década.

Tradicionalmente, en el campo de redes de interconexión de altas prestaciones se han utilizado frecuentemente simuladores de eventos hechos a medida que modelaban la red a nivel de paquete. Estos simuladores, como se ha descrito, permiten modelar y evaluar los nuevos diseños de red sin tener que realizar inversiones en el hardware. El problema de estas herramientas es su alto recorrido, ya que se desarrollan para propósitos muy específicos y su tempo de vida es breve. Además, al desarrollarse prácticamente desde cero, el tiempo invertido en la depuración del código, la validación y la verificación, retrasa de forma considerable de evaluación final del modelo de desarrollado.

El reto, por tanto, consiste en invertir el menor tiempo posible en las fases de depuración del código fuente, validación y verificación del modelo, mediante el uso de herramientas de simulación de propósito general, centrados en ofrecer un marco de trabajo común que facilite el desarrollo de modelos de simulación.

Entre la variedad de frameworks de simulación que existen, hemos decidido utilizar OMNeT++ por los siguientes motivos [5]:

- Es gratuito para uso académico y de investigación sin ánimo de lucro.
- Permite simulaciones a gran escala.
- Proporciona la visualización de los modelos de simulación para ayudar a la depuración.
- Se integra con la plataforma Eclipse y una herramienta de análisis ($scavetool$).
- Se puede ejecutar tanto en Linux, Windows, Mac OS y otros sistemas UNIX.
- El software del simulador es modular y personalizable.
- Existe una gran comunidad de usuarios.
- El código es simple y fácil de entender.
- El desarrollo del modelo es rápido.
- El kernel del simulador esta implementado, verificado y optimizado.
- Se agregan nuevas características con el tiempo, gracias a su comunidad.

En este artículo, proponemos un modelo general de redes de interconexión de altas prestaciones basado en OMNeT++. Nuestro principal objetivo es establecer unas bases para futuras ampliaciones y migrar nuestro simulador hecho a medida a este nuevo modelo.

El resto de este artículo esta organizado de la siguiente manera: la sección II describe el estado del arte en cuanto a simuladores y herramientas de simulación para redes de interconexión de altas prestaciones. La sección III describe en profundidad nuestro modelo. La sección IV muestra varios ejemplos de resultados obtenidos. Por último, la sección V describe las conclusiones y nuestro el trabajo que vamos a realizar en el futuro.

[1]Departamento de Sistemas Informáticos, Universidad de Castilla-La Mancha. E-mail:{pedro.yebenes, jesus.escudero, pedrojavier.garcia, francisco.quiles}@uclm.es

[2]En estos sistemas, el consumo energético de la red es más de un 30% [3]

II. Trabajo Relacionado

A. Bases de la simulación

La simulación es una técnica muy utilizada para analizar el rendimiento de sistemas reales o abstractos. Si el sistema no está disponible, el modelo de simulación permite comparar las diferentes alternativas y evaluar su rendimiento. Además, si el sistema está disponible, el modelo de simulación es preferible a las mediciones porque se pueden comparar distintos ambientes y cargas de trabajo [6]. Hay dos tipos de simulaciones: las *simulaciones dirigidas por trazas* y las *simulaciones de eventos discretos* [6]. Las primeras utilizan trazas como entradas y se utilizan para analizar y optimizar los recursos de los algoritmos de gestión. Sin embargo, las de eventos discretos son las que utilizan un modelo del sistema de estados discretos.

B. Simuladores y frameworks de simulación

Hay muchos simuladores que han sido realizados por grupos de investigación como el BookSim [7], TOPAZ [8] o INSEE [9]. Además, hay algunos que han sido hechos a medida para tesis doctorales [10][11] centradas en las redes de interconexión.

Aparte de estos simuladores, hay varias herramientas de simulación comerciales y no comerciales. Algunas de estas son: NS-2 [12] utiliza simulaciones de eventos discretos. J-Sim [13] es un sistema de simulación basado en Java para construir modelos cuantitativos y analizarlos respecto a datos de referencia experimentales. OPNET Modeler [14] es un simulador de redes comercial que incorpora un amplia gama de protocolos y tecnologías y un entorno de desarrollo que permite modelarlas. SSF [15] es un estándar de dominio publico para eventos discretos para la simulación de sistemas grandes y complejos. QualNet [16] es una plataforma comercial de simulación para desarrollos militares. OMNeT++ [17] es un *framework* de simulación de dirigida por eventos basado en C++ con una arquitectura genérica para el modelado de redes de comunicación, multiprocesadores y otros sistemas paralelos o distribuidos. OMNEST [18] es igual que OMNeT++ pero con licencia comercial. Ya hemos mencionado en la sección I las ventajas, que desde nuestro punto de vista, hacen que OMNeT++ sean de las mejores opciones para desarrollar un simulador. En la siguiente sección vamos a describir la plataforma.

Como se ha descrito, OMNeT presenta ciertas características que lo hacen muy apropiado para modelar redes de interconexión de altas prestaciones. En el siguiente apartado se describe esta plataforma.

C. Descripción de OMNeT++

Un modelo de OMNeT++ está compuesto de módulos que se comunican entre si mediante *paso de mensajes* [5]. Los *módulos simples* están escritos en C++ usando la biblioteca de clases. Estos pueden agruparse para forma *módulos compuestos* y así sucesivamente. Las comunicaciones entre los módulos se hacen enviando mensajes mediante *puertas*, pero estos pueden ser enviados directamente a los módulos. Las *puertas* son las interfaces de entrada y salida de los módulos. Una *puerta* de entrada y otra de salida pueden ser unidas mediante una *conexión* cuyos atributos pueden ser personalizados. Así pues, los mensajes van entre los módulos a través de una cadena de conexiones. Además, los módulos pueden tener parámetros que son usados para pasarles los datos de configuración y a sus submódulos.

La estructura del simulador es definida en el lenguaje declarativo NED (*NEtwork Description*). Con él se declaran los *módulos simples* con sus correspondientes *puertas* y parámetros. Son atómicos y no pueden ser divididos. Luego estos se conectan en *módulos compuestos* para crear la red. Además, es posible crear las topologías regulares más comunes cuyos parámetros son numéricos. Todos estos parámetros de entrada están especificados en los archivos INI. Por lo tanto, una simulación puede ser ejecutada para cada combinación de parámetros.

Además, OMNeT++ provee una biblioteca de simulación para implementar *módulos simples* que implementa depuración de alto nivel y trazabilidad, así como animación automática. También ayuda con *memory leaks*, *pointer aliasing* y otros problemas de asignación de memoria.

Para unir un módulo NED con el código C++, se utiliza la función `Define_Module`. La estructura de este código es siempre muy similar y fácil de entender. Esto es porque todas las clases heredan de `cSimpleModule` porque solo estos módulos tienen funcionalidad. Esta clase tiene tres métodos que hay que sobrescribir: `initialize`, `handleMessage` y `finish`. Obviamente, la primera se utiliza cuando el módulo es creado y aquí se inicializan las estructuras de datos, se obtienen los parámetros del archivo INI, se registran las señales para obtener estadísticas y se programan los mensajes de recolección de datos. La última función, se llama cuando la simulación acaba correctamente y se utiliza para liberar memoria y para registrar algunos valores. La función `handleMessage` se llama cuando un mensaje llega al módulo, y de este hay que obtener su tipo (*kind*) que es un atributo de la clase `cMessage`, de la que heredan todos los mensajes. Dependiendo de su tipo, se llama a una función específica para su procesado.

III. Modelar la red de interconexión con OMNeT++

A. Consideraciones Generales

Los primero que hay que considerar es cómo se va a modelar la red de interconexión. Una red de interconexión se compone principalmente de *host channel adpaters* (HCA) (las interfaces que conectan los nodos de procesamiento a la red), *switches* y enlaces entre ellos.

El switch se encarga de enviar los paquetes dentro de la red y de establecer comunicaciones entre los diferentes caminos de la red. Estos se conectan mediante enlaces y, dependiendo de como lo hagan, crean las topologías de red.

Un switch implementa el encaminamiento, el arbitraje, los mecanismos de gestión de los buffers, el control de flujo y las funciones de conmutación. Fig. 1 muestra la arquitectura del switch y sus componentes: *buffers*, *link controller*, *internal switch*, *routing unit* y *arbitration unit*. Tienen puertos de entrada y de salida que contienen buffers para almacenar paquetes. Ellos están interconectados por un crossbar interno que permite conexiones entre pares de puertos [19]. Los paquetes que llega a los puertos de entrada se almacena en los buffers y hacen una petición para cruzar el crossbar: son encaminados y cuando la petición es concedida, cruzan el crossbar y dejan el switch por el puerto de salida.

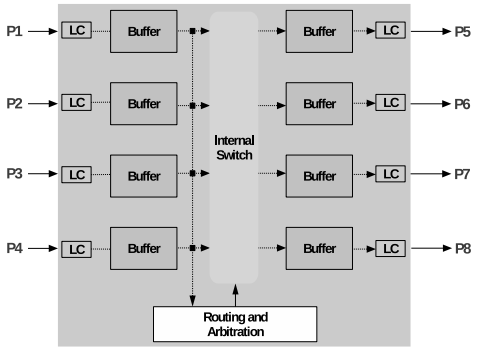


Fig. 1. Diagrama de Switch (LC = Link Controller)

Hemos asumido que un switch segmentado con M puertos, una unidad de encaminamiento y una unidad de arbitraje centralizadas y q colas por buffer. La política de conmutación es *virtual cut-through* (VCT) [20] y el control de flujo está basado en créditos. El ancho de banda de los enlaces internos, externos y el crossbar es de w bits por unidad de tiempo.

Por otro lado tenemos los HCAs, que representan las interfaces de red, que conectan los nodos finales con los switches. Las interfaces de red se consideran elementos esenciales en las redes de interconexión porque son los encargados de la inyección del tráfico generado por los dispositivos los nodos finales. Cuando un paquete es generado, es encaminado y asignado a una cola mientras espera a ser inyectado. Además, consume los paquetes entrantes y los entregar al nodo para el procesado.

B. Módulos NED

Como se ha descrito anteriormente, el primer paso en el desarrollo de un modelo en OMNeT++ es definir los módulos utilizados con el lenguaje NED (NEtwork Description). Estos módulos contienen parámetros que definen su estructura y *puertas* para comunicarse con otros módulos. Cada módulo necesita *puertas* para enviar y recibir mensajes. Normalmente si hay una *puerta* de entrada para recibir paquetes, hay otra de salida para notificar el procesamiento de un paquete previo. Por ejemplo, si un buffer envía un paquete a otro módulo, comunicará a otro módulo conectado de que ha procesado el mensaje.

Por lo tanto el modelo de switch necesita los siguientes módulos simples con sus *puertas*:

- *Link Controller*. Está asociado a un puerto de entrada/salida y gestiona los mecanismos del buffer y el control de flujo. Este módulo tiene las siguientes *puertas*: una de entrada/salida para enviar/recibir paquetes y créditos desde el exterior, una de entrada para enviar paquetes hacia dentro del switch, una de salida para recibir paquetes desde dentro del switch, una de entrada para recibir mensajes de control para que envíe más paquetes y una de salida para pedir nuevos paquetes.
- *Buffer*. Almacena los paquetes que llegan y los que están bloqueados. Tiene dos *puertas* de salida: una para enviar paquetes y otra para enviar mensajes de control; y dos de entrada: una para recibir paquetes y la otra, para mensajes de control.
- *Crossbar Registers*. Almacena los paquetes que han hecho peticiones a los puertos de salida. La conexión entre su *puerta* de salida de los paquetes con el *switch port* es el *crossbar*. Tiene tres arrays de *puertas*: para recibir paquetes desde los puertos, para enviar paquetes a los puertos y para enviarles mensajes de control. Además de esto, tiene tres *puertas* para comunicarse con el router y el árbitro: una de salida para enviar peticiones de paquetes que quieren cruzarlo, una de entrada para recibir las concesiones y una se salida para notificar la liberación de los puertos que han terminado de transmitir.
- *Router*. Implementa el algoritmo de encaminamiento que determina el puerto de salida. Solo tiene un *puerta* de entrada para recibir peticiones y una puerta de salida para enviar las peticiones procesadas.
- *Árbitro*. Decide y establece a qué puertos de entrada, que contienen paquetes listos para ser transmitidos, se permite enviar paquetes a los puertos de salida. Tiene una *puerta* para recibir peticiones y otra para enviarlas. Además tiene un array de *puertas* para recibir información desde el buffer de salida y otra *puerta* de entrada para recibir notificaciones de los enlaces del crossbar que han terminado de transmitir.

Para simplificar el modelo del switch, un módulo *link controller* y dos módulos *buffer* (el de salida y el de entrada), se agrupan para forma un *switch port* como se ve en Fig. 2a. Este módulo compuesto tiene internamente conectado el módulo *link controller* con el módulo *buffer* de entrada y el módulo *buffer* de salida. Sus *puertas* exteriores son: una para la salida de los paquetes desde el buffer de entrada y otra de entrada para la llegada de mensajes de control; una entrada para el buffer de salida y su salida para enviar mensajes de control; y una *puerta* de entrada/salida unida al link controller para conectarse con otro switch.

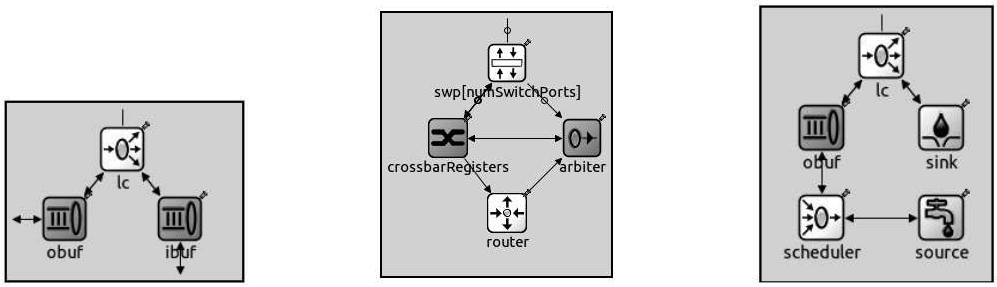
Con todo esto el switch tiene un array de módulos *switch port* (véase Fig. 2b) cuyo tamaño dependerá de la topología de red, un módulo *router*, un módulo árbitro y un módulo *crossbar registers*. Estos módulos unidos forma el módulo compuesto *switch*. Para comunicarse con el exterior, tiene un array de puertas de entrada/salida y, cada una de ellas, está unida a un módulo switch port. Estos internamente están conectados con los módulos *crossbar registers y árbitro*, este con el módulo *router* y el módulo *árbitro*; y el módulo *router* con el de *árbitro*. Además hay un `DatarateChannel` en el enlace entre el puerto de salida de los paquetes del módulo de *crossbar registries* y la *puerta* de entrada del *switch port* que retarda los mensajes que cruzan el *crossbar*.

Además de switch, hay que modelar los HCA. Cada uno tiene una fuente que genera paquetes, que se almacenan en las colas de generación. Si hay créditos para un paquete en el buffer de salida, el planificador lo encamina y le asigna una cola para este buffer, mientras espera a que pueda ser enviado a la red a través del link controller.

Así pues, hay que definir nuevos módulos:

- *Source*. Crea paquetes. Sus *puertas* son una de salida para enviar a paquetes y otra de entrada para recibir mensajes de control..
- *Scheduler*. Asigna colas a los paquetes y les hace su primer encaminamiento.
- *Sink*. Procesa los paquetes. Tiene una puerta de entrada para los paquetes y una puerta de salida para enviar mensajes de control.

Por lo tanto, el módulo compuesto del HCA necesita



(a) Módulo Switch Port (b) Módulo Switch (c) Módulo HCA

Fig. 2. Modulos NED

los siguientes módulos: *source*, *scheduler*, *buffer*, *link controller* y *sink* (Fig. 2c). Este módulo solo tiene una puerta de entrada y salida para comunicarse con otros módulos. Internamente los módulos está conectados así: source, con el scheduler; el scheduler, con el buffer; el buffer, con el link controller; y el link controller, con el sink.

Con estos módulos compuestos se define la red. Esta consiste en un array de HCAs y otro array de switches, que se conectarán dependiendo de la topología. Además, el número de puertos de cada switch de ella. Estos módulos están conectados mediante un `DatarateChannel` que introduce un retardo en la transmisión los mensajes que lo cruzan dependiendo del tamaño de paquete y su distancia.

C. Modelado de mensajes

Además de los módulos, los mensajes que se van a utilizar en nuestro modelo se definen en un archivo MSG. Todos los mensajes de OMNeT++ heredan de la clase `cMessage` que contiene toda la información básica. Además, OMNeT++ provee una subclase de esta, `cPacket`, que contiene información adicional como el tamaño del mensaje.

Los mensajes que utilizan los switches y los HCAs para comunicarse son los paquetes y los créditos. Como se describe después, un paquete se divide en dos mensajes que heredan de `cPacket`: *header message* y *data message*. El primero es la cabecera, y contiene información sobre la fuente, el destino, la cola y los próximos puertos de salida. También hay *credit message* que también heredan de `cPacket`. El tamaño de estos mensajes está definido en al archivo INI.

Además hay mensajes internos que heredan de `cMessage`: *control message* que es utilizado por los módulos para notificar a un módulo conectado que acaban de procesar un paquete, que suele ser cuando lo envían al siguiente módulo. *Request message* se crea por el módulo de *crossbar registers* cuando le llega un paquete y es usado por el árbitro y el router. *Grant message* es enviado por el módulo de árbitro al módulo *crossbar registers* para conceder un puerto de salida a un paquete. *Free message* es el responsable de notificar al árbitro la liberación de los puertos del crossbar. *Timer message* es un mensaje auto-enviado que lo utilizan los módulos *link controller* y los *crossbar registers* para indicar cuándo pueden volver a transmitir. Y por último, *log message* que activa los mecanismos para tomar resultados.

D. Capa de conmutación

Aquí es donde se implementa mecanismo de envío de mensajes a través de la red [20]. Ésta utiliza una política de control de flujo basada en créditos [19]. Tiene un contador por cada cola, los cuales son inicializados al tamaño de las colas del buffer que va a recibir los paquetes. Cada vez que un paquete se envía, el contador correspondiente se decrementa. Cuando el receptor del paquete consume un paquete, devuelve un crédito utilizando un *credit message* el cual incrementa el contador. Si el contador es cero, el módulo emisor tiene que esperar a un crédito para volver a transmitir. Para prevenir que los canales estén ociosos, el número de créditos tiene que ser mayor que el *round-trip time*.

Los link controllers son los módulos que soportan esta funcionalidad. Ellos reciben del archivo INI los parámetros del tamaño de los buffers a los cuales van a enviar, e inicializan el contador o los contadores. Cada link controller tiene una `cPacketQueue` que es una cola FIFO donde se almacenan los *credit message* y un registro para almacenar a un paquete mientras se envían los créditos. Los créditos los crea el link controller cuando recibe un mensaje desde sus puertas conectadas al interior del switch o del HCA, indicando la consumición de un paquete.

Cuando el canal entre los módulos está libre, el link controller empieza a enviar. Primero envía los créditos porque son prioritarios, y hasta que la cola se vacía, no se envía ningún paquete. Cuando un paquete es enviado, el link controller calcula cuánto tiempo va a estar en canal ocupado utilizando su ancho de banda y la longitud del mensaje. Entonces se auto-envía un *timer message*. Cuando llega un paquete al módulo receptor, se envía automáticamente al interior.

Además, los árbitros saben los créditos disponibles para el siguiente buffer para poder seleccionar una petición que pueda ser enviada. Por este motivo, los link controller reenvían los créditos al árbitro.

En algunas topologías con ciertos algoritmos de encaminamiento se puede producir interbloqueo; como los toros con encaminamiento del tipo *dimension order*. Por eso se ha implementado el mecanismo de la *burbuja* [21], que elimina la posibilidad de interbloqueo.

El modelo implementa la conmutación *virtual cut-through* (VCT) por lo que se ha dividido los paquetes en dos: *header message* y *data message*. Por tanto, todos los registros y las colas también deben ser divididos en dos.

E. Organización de los buffers

Se han modelado los esquemas de colas 1Q, VOQsw[22], VOQnet[23] y DAMQ[24] en el módulo buffer. Este módulo toma del archivo INI el número de colas que tiene. Éstas colas es modeladas como un array de `cPacketQueue`. Sin embargo, la lógica que decide en qué cola se almacenar los paquetes es la `VOQLogic`, una

clase abstracta que tiene un método para asignar una cola a un paquete y otro para saber en qué cola se va a almacenar el paquete en el siguiente buffer. De esta clase heredan los distintos tipos de esquemas de colas que sobrescriben estos métodos para darles funcionalidad. Esta lógica se encuentra en el módulo link controller para decidir a dónde van los paquetes que llegan, en el módulo scheduler, para asignarles la cola en el módulo buffer del HCA, y en el módulo árbitro, para saber a que la cola van a ir.

En el esquema de única cola (1Q), toda el espacio disponible es para esa cola. Cuando se utiliza VOQsw, el número de colas de cada buffer es el número de puertos que tiene cada switch; y en VOQnet este número es el número de nodos de la red.

Como DAMQ es una técnica dinámica, hay que tener en cuenta otros valores. El contador de créditos de cada cola se inicializa al tamaño total del buffer y, además se lleva la cuenta del número de créditos totales del buffer. Siempre que un paquete va a ser asignado debe de comprobase si hay créditos en el buffer. Si el número de créditos para esa cola menos la suman de el tamaño mínimo para esas otras colas es mayor que cero. Esto se hace para evitar el *buffer hogging*.

F. Encaminamiento

Los routers modelados implementan *look-ahead routing* [25]. Estos, en vez de realizar el cálculo para el puerto de salida del switch actual, lo hacen para el siguiente, y para el siguiente a este. Estos valores son almacenados en la cabecera del paquete. De esta forma, un paquete puede ser encaminado y arbitrado en paralelo.

Estos valores son necesarios para que el link controller puede asignar una cola del buffer al paquete y para que, cuando una petición suya sea arbitrada, saber si hay créditos en para la cola correspondiente en el siguiente buffer.

En nuestro modelo, se ha creado una clase abstracta (`Routing Unit`) de la que heredan todos los algoritmos de encaminamiento. Tiene un método para encaminar los paquetes, y que sus hijos implementan para que tenga funcionalidad. El módulo router tiene una `RoutingUnit`, así como el módulo scheduler, que es el encargado de hacer el primer encaminamiento.

G. Árbitro y algoritmos de arbitraje

El algoritmo de arbitraje que se ha modelado ha sido el de *tres fases* [19]. Además, aunque haya un árbitro centralizado, se comporta como un conjunto de árbitros distribuidos por puerto de salida.

Para que el arbitraje sea más equitativo utiliza round-robin y esquemas basado en la antigüedad de las peticiones de cruce: Cuando un puerto de salida está libre, elige una petición entre sus colas mediante round-robin. Si el puerto de salida elegido está libre y hay créditos en el el siguiente buffer para el paquete, el árbitro de este puerto de salida comprueba si hay otras peticiones más antiguas. Si no las hay, esta petición es concedida. Pero si las hay, elige la petición más antigua y el puerto de entrada continúa haciendo round-robin hasta que una petición suya sea concedida o no haya más peticiones. Cuando se libera un puerto de salida, el árbitro elige la petición más antigua que lo solicite y que tenga créditos en el siguiente buffer.

TABLA I
ESCENARIOS UTILIZADOS EN LA EVALUACIÓN

	Topología	Tamaño de la red	Tráfico Aleatorio	Tráfico Hot-Spot
#1	2DTorus	16x16	100%	0%
#2	2DTorus	16x16	75%	25%
#3	2DMesh	16x16	100%	0%
#4	2DMesh	16x16	75%	25%

H. Generación de Estadísticas

Los resultados de la simulación son registrados en archivos *output vector* (.vec) y *output scalar* (.sca). Los archivos .vec contienen varios *output vectors*, cada uno con una serie de pares de valores (marca de tiempo, valor), que capturan el comportamiento a lo largo del tiempo. Para registrar estos vectores de datos, se utiliza el mecanismo de *simulation signals*. Este provee `@signals` (propiedades que se añaden a los archivos NED) que son registradas por los módulos por medio de `registerSignal`, a las que se les emite datos con la función `emit`. Los valores escalares son grabados con la función `recordScalar`.

Para recoger los datos hay algunas métricas que emiten datos cada cierto tiempo, como la productividad; y otras, cada vez que ocurre un evento, como la latencia.

IV. EJEMPLO DE USO

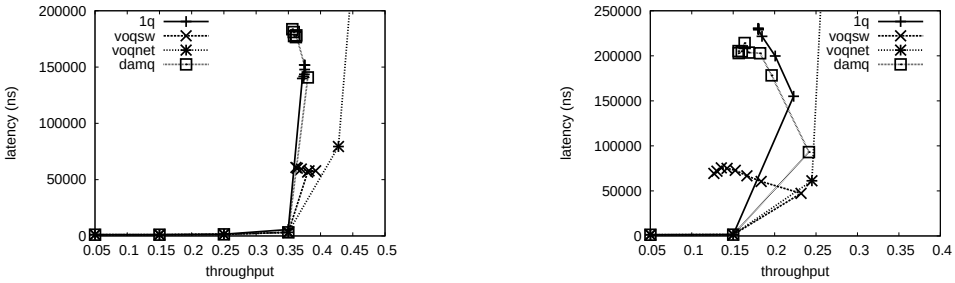
En este apartado y a ejemplo de las prestaciones del simulador, se muestran los resultados que comparan los diferentes esquemas de colas que soporta el simulador en diferentes redes. Los caso de tráfico considerados se muestran en la tabla I. Estos escenarios tienen comunes los siguientes parámetros: el ancho de banda de los enlaces y del crossbar y la generación de paquetes es de 2.5 GB/S; la longitud de los enlaces es de 1.2 m, por tanto el tiempo de vuelo es de 6 nanosegundos; el tamaño del paquete es de 2048 bytes, de los cuales 48 son de la cabecera; el algoritmo de encaminamiento utilizado es *dimension order* (xy); arbitrar y encaminar un paquete lleva 6 nanosegundos; la memoria de los buffers es de 128 Kilobytes excepto en VOQnet, que es de 2 Megabytes, ya que tiene más colas.

A. Discusión de los resultados

Las figuras 3 y 4 muestran los resultados para los esquemas de colas 1Q, VOQsw, VOQnet y DAMQ. En términos generales, la eficiencia de VOQnet obtiene los mejores resultados, ya que previene completamente el HoL-blocking. Sin embargo, la memoria de su buffer es mayor que en las otras técnicas. Por otro lado, 1Q obtiene el peor resultado porque no previene el HoL-blocking, como muestra Fig. 4. VOQsw y DAMQ son muy similares, aunque VOQsw tiene menos latencia.

V. CONCLUSIONES Y TRABAJO FUTURO

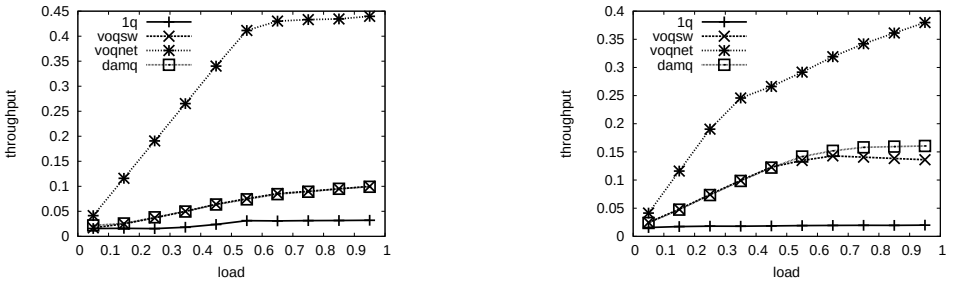
Los simuladores son una herramienta esencial en la evaluación de prestaciones de las arquitecturas de red de interconexión. La tendencia es desarrollar simuladores específicos desde cero, con la creencia de que ésta práctica puede facilitar y acelerar el trabajo. Sin embargo, el desarrollo de un sistema software en general, y las herramientas de simulación en particular, supone una gran inversión en tiempo y en dinero. Por suerte, existen plataformas de desarrollo de simuladores orientadas a fa-



(a) Latencia vs Productividad (Escenario #1)

(b) Latencia vs Productividad (Escenario #3)

Fig. 3. Gráficos de los escenarios #1 y #3



(a) Productividad vs Carga (Escenario #2)

(b) Productividad vs Carga (Escenario #4)

Fig. 4. Resultados de los escenarios #2 y #4

cilitar el trabajo del programador. Una de las más extendidas es OMNeT++, una plataforma de desarrollo de modelos de simulación, apoyada por una gran comunidad de usuarios, que facilita el desarrollo, depuración, validación y verificación de modelos de simulación.

En este artículo, se ha descrito cómo modelar un simulador de redes de interconexión de altas prestaciones utilizando la plataforma OMNeT++. Se ha detallado nuestro modelo de red en OMNeT++ y se han analizado brevemente los resultados obtenidos en ciertos escenarios. Además, de nuestra experiencia con este trabajo podemos concluir que OMNeT++ reduce considerablemente el tiempo invertido en el desarrollo de un modelo de simulación de redes de interconexión.

AGRADECIMIENTOS

Este trabajo ha sido parcialmente financiado por los antiguos MEC y el MICINN (ahora MINECO), así como la Comisión Europea a través de los fondos FEDER, mediante los proyectos CSD2006-00046 y TIN2009-14475-C04. Además, también ha sido parcialmente financiado por la JCCM, mediante el proyecto POII10-0289-3724.

REFERENCIAS

[1] *InfiniBand Architecture Specification Volumen 1*, InfiniBand Trade Association Std., nov 2007. [Online]. Available: http://www.infinibandta.org/

[2] Myrinet, 2000 series networking. [Online]. Available: http://www.cspi.com/

[3] U. D. of Energy, Ed., *The Opportunities and Challenges of Exascale Computing. Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee*, 2010.

[4] J. Banks, J. Carson, B. L. Nelson, and D. Nicol, *Discrete-Event System Simulation (4th Edition)*. Prentice Hall, Dec. 2004.

[5] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *Simutools '08: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*. ICST, Brussels, Belgium, Belgium: ICST, 2008, pp. 1–10.

[6] R. Jain, *The Art of Computer Systems Performance Analysis*, W. P. Computing, Ed. : John Wiley & Sons, Inc, 1991.

[7] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*, Elsevier, Ed. : Morgan Kaufmann Publishers, 2003.

[8] P. Abad, P. Prieto, L. Menezo, A. Colaso, V. Puete, and J.-Ã. Gregorio, "Topaz: An open-source interconnection network simulator for chip multiprocessors and supercomputers," in *NOCS 2012*.

[9] J. Navarridas, J. Miguel-Alonso, J. A. Pascual, and F. J. Ridruejo, "Simulating and evaluating interconnection networks with insee," *Simulation Modelling Practice and Theory*, vol. 19, pp. 494–515, 2011.

[10] J. Escudero Sahuquillo, "Towards efficient and feasible congestion management techniques for high-perfomance interconnection network using distributed routing," Ph.D. dissertation, Departamento de Sistemas Informáticos. Universidad de Castilla-La Mancha, dec 2011.

[11] P. J. García García, "Diseño de una técnica escalable y eficiente para el control de congestión en redes de interconexión," Ph.D. dissertation, Departamento de Sistemas Informáticos. Universidad de Castilla-La Mancha, sep 2006.

[12] S. Bajaj et all, "Improving simulation for network research," University of Southern California, Tech. Rep. 99-702b, March 1999, revised September 1999, to appear in IEEE Computer.

[13] A. Sobeih et all, "J-sim: a simulation and emulation environment for wireless sensor networks," *Wireless Communications, IEEE*, vol. 13, no. 11, pp. 104–119, aug 2006.

[14] Opnet modeler. OPNET Technologies, Inc. [Online]. Available: http://www.opnet.com/

[15] J. H. Cowie, D. M. Nicol, and A. T. Ogielski, "Modeling the global internet," *Computing in Science & Engineering*, vol. 1, no. 1, pp. 42–50, 1999.

[16] Qualnet home page. Scalable. [Online]. Available: http://www.scalable-networks.com/

[17] Omnet++ home page. OpenSim Ltd. [Online]. Available: http://www.omnetpp.org

[18] Omnest home page. OpenSim Ltd. [Online]. Available: http://omnest.com

[19] T. M. Pinkston and J. Duato, "Appendix e," in *Computer Architecture: A Quantitative Approach*, Elsevier, Ed. Morgan Kaufmann Publishers, 2006.

[20] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks an Engineering Approach*, E. Science, Ed. San Francisco, CA, USA: Morgan kaufmann Publiishers, 2003.

[21] C. Carrión, R. Beivide, J. A. Gregorio, and F. Vallejo, "A flow control mechanism to avoid message deadlock in k-ary n-cube networks," in *Proceedings of the Fourth International Conference on High-Performance Computing*, ser. HIPC '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 322–.

[22] W. J. Dally, P. Carvey, and L. Dennison, "Architecture of the avici terabit switch/router." in *Proceedings of the 6th Symposium on Hot Interconnects*, 1998, pp. 41–50.

[23] T. E. Anderson, S. S. Owicki, J. B. Saxe, and C. P. Thacker, "High-speed switch scheduling for local-area networks," *ACM Trans. Comput. Syst.*, vol. 11, no. 4, pp. 319–352, Nov. 1993.

[24] Y. Tamir and G. L. Frazier, "Dynamically-allocated multi-queue buffers for vlsi communication switches," *IEEE Trans. Comput.*, vol. 41, no. 6, pp. 725–737, Jun. 1992.

[25] J.-K. Peir and Y.-H. Lee, "Look-ahead routing switches for multistage interconnection networks," *J. Parallel Distrib. Comput.*, vol. 19, no. 1, pp. 1–10, Sep. 1993.