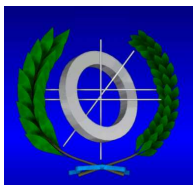


UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA POLITÉCNICA SUPERIOR

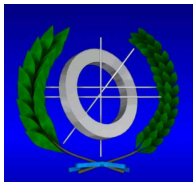
INGENIERÍA
EN INFORMÁTICA

PROYECTO FIN DE CARRERA

Estudio y puesta en marcha de una subred InfiniBand

Juan Antonio Villar Ortiz

Julio, 2004



UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA POLITÉCNICA SUPERIOR

Departamento de Informática

PROYECTO FIN DE CARRERA

Estudio y puesta en marcha de una subred InfiniBand

Autor: Juan Antonio Villar Ortiz
Directores: José Luis Sánchez García y Francisco José Alfaro Cortés

Julio, 2004

Reunido en la fecha el Tribunal evaluador, que más abajo se cita, del Proyecto Fin de Carrera titulado:

_____

_____

presentado por D/D[a]

_____

y siendo su/s tutor/es

_____

se otorga la calificación de _____

Y para que así conste, se firma la presente acta en

Albacete a ..... de ..... de 20.....

PRESIDENTE: _____

SECRETARIO: _____

VOCAL: _____

SECRETARIO PRESIDENTE VOCAL

RESUMEN

InfiniBand es una tecnología de interconexión entre estaciones y que también puede ser utilizada como sistema de Entrada/Salida. Fue desarrollada por la InfiniBand Trade Association (IBTA) para soportar fiabilidad, disponibilidad, rendimiento y escalabilidad necesarias para los sistemas presentes y futuros, en niveles significativamente mejores que pueden alcanzarse con los sistemas de E/S basados en buses.

A partir de las especificaciones de InfiniBand, se están realizando propuestas para explotar sus propiedades. La mayoría de esas propuestas se estudian utilizando simuladores, donde los investigadores pueden estudiar su comportamiento en un entorno seguro y controlable, pero carecen de las características propias de un entorno real. Es interesante conocer el comportamiento de todas esas propuestas sobre productos reales. En esta línea, el grupo de *Redes y Arquitecturas de Altas Prestaciones* (RAAP) del *Instituto de Investigación en Informática de Albacete* (I^3A) ha adquirido el material necesario para montar una red InfiniBand real, y así poder evaluar y contrastar los trabajos hasta aquí realizados mediante simuladores. Antes de que se puedan trasladar sus propuestas de los simuladores a este entorno, es necesario que pasen por un período de adaptación a dicho entorno y de preparación para su trabajo con estas máquinas. En el RAAP aún no se tienen los conocimientos necesarios para acondicionar el entorno de trabajo, ya que no existen los precedentes necesarios sobre este producto en concreto. Reducir el tiempo que los investigadores necesitan para adaptarse a este entorno es de vital importancia para acelerar su trabajo.

Con el cumplimiento de los objetivos de este proyecto se generará en el grupo un conocimiento base, para que sus miembros puedan inciarse en este entorno de un modo más seguro. Se ha tenido que poner en marcha el material, solucionando todos los problemas aparecidos, y adquiriendo la experiencia necesaria para utilizar los dispositivos. Es por ello, que los conocimientos en administración de sistemas han sido clave para la consecución de estos objetivos. Paralelamente, se ha realizado un estudio previo sobre elementos software de gran importancia en una red InfiniBand, como puede ser el Subnet Manager. Los miembros del grupo pueden encontrar en este proyecto una valiosa introducción sobre el software OpenSM.

Quiero dedicar este proyecto a mis padres **Antonio** y **Belén** quienes me han formado como persona y han hecho posible que hoy tenga esta oportunidad. No puedo olvidar a mis hermanas **Joaquina** y **Belén**. Gracias.

Quiero dedicar también un agradecimiento especial a mis directores de proyecto, José Luis y Francisco José, que han colaborado en la realización de este proyecto. También quiero agradecer la colaboración de Vicente y Raúl por su aportación de ideas fundamentales para el desarrollo de este proyecto.

No quiero olvidarme de mis compañeros que han estado ahí para ofrecerme su amistad y ayuda. Sois únicos.

Índice general

Índice de figuras

Capítulo 1

INTRODUCCIÓN

Desde sus comienzos, la informática ha experimentado contínuos cambios. La búsqueda incesante de sistemas con más prestaciones e inferiores costes a los actuales ha sido y sigue siendo la referencia que guía a los diseñadores de computadores. Todos, usuarios y fabricantes, quieren mejores máquinas y más baratas que las existentes.

Un sistema informático presenta una estructura heterogénea de subsistemas unidos con un único fin, mejores prestaciones. Para aumentar el rendimiento del sistema global se intenta mejorar los subsistemas que lo componen, pero debido a las dependencias entre ellos esta regla no se cumple a la perfección y hasta sus mejores componentes ven disminuir sus prestaciones por culpa de otros de menor velocidad. La velocidad la marca el más lento. Ello induce a que siempre se esté en una contínua carrera donde lo moderno deja de serlo en años o meses.

El pilar fundamental en la informática es el computador. La idea clásica de un ordenador se ha mantenido hasta nuestros días, pero evolucionando hasta donde no se habían imaginados los pioneros. Un caso particular es el de los *supercomputadores*. Durante mucho tiempo se ha creído en la idea de los supercomputadores, máquinas que dan un rendimiento aceptable pero que tienen el inconveniente implícito, en muchas ocasiones insalvable, de su elevado coste. Con todo lo aprendido en los supercomputadores, el interés de los investigadores se dirigió a conseguir sistemas más económicos y que dieran prestaciones similares a los supercomputadores para un gran número de aplicaciones.

Esto es posible gracias al aumento de las prestaciones de los computadores personales, su reducción de precio y fundamentalmente al avance en las tecnologías de redes de interconexión, capaces de proporcionar un ancho de banda alto y latencia baja. Así surgen los *clusters*, sistemas formados por computadores personales conectados por medio

de redes de altas prestaciones, que mediante el software apropiado permiten la comunicación y control de los procesos de manera eficiente, de forma que para ciertas aplicaciones proporcionan un rendimiento equiparable a los supercomputadores. Esto ha permitido que dichas aplicaciones, que antes se veían restringidas a ciertos sistemas muy especializados, puedan ser acogidas por comunidades más generales, institutos, empresas, universidades, etc. La idea que subyace es que muchos ordenadores conectados entre sí pueden igualar e incluso superar las prestaciones de los supercomputadores a un coste aceptable.

En el mercado existen redes de características muy variadas, *Ethernet*, *Fibre Channel* [5], *Myrinet* [17], etc. Todas ellas alcanzan actualmente velocidades de transmisión elevadas. Sin embargo, no alcanzan las velocidades que los procesadores actuales requieren. Cada vez que un procesador necesita un dato externo a su sistema local, su rendimiento se ve mermado por el mero hecho de la espera producida hasta que el dato está disponible para su uso. Además, en los últimos años las aplicaciones requieren más y más información, aplicaciones multimedia, Internet de banda ancha, transmisión en tiempo real, etc.

Entre los muchos componentes de un sistema, el sistema de entrada y salida (E/S) es uno de los más importantes. Incluso ya *Von Newmann* lo incluía como uno de los cinco componentes de un ordenador. Todos los datos que pasan por el sistema deben hacerlo utilizando el sistema de E/S, por lo que su rendimiento influirá directamente en el rendimiento global.

El clásico bus *PCI* [19] es uno de los sistemas de E/S más utilizados en el mundo. A partir de él, han surgido varias evoluciones, como el *PCI-X* [19], pero que resultan insuficientes para las necesidades actuales. Desafortunadamente la E/S de un ordenador se ha convertido en un "cuello de botella" agravado por la gran evolución en las redes de altas prestaciones y en los mismos ordenadores personales. La velocidad que se puede alcanzar en una red moderna está limitada al sistema de E/S. Si los sistemas son más rápidos entonces necesitan más información y en cantidades enormes. Se ve claramente que se necesita una nueva generación de sistemas de E/S que sean capaces de soportar los requisitos actuales y futuros. Es la llamada *3GIO* o *tercera generación de E/S*.

Ante este nuevo reto, el bus PCI respondió con el nuevo *PCI-Express* [19] de *3GIO*. Otra opción es *InfiniBand* que surgió para sustituir a los sistemas de E/S y a las redes de altas prestaciones, pero ante la aparición del *PCI-Express* de *3GIO* el campo de los sistemas de E/S parece tenerlo perdido. No obstante, en el mundo de las redes de altas prestaciones se apunta como un gran competidor por el gran apoyo que ha recibido de grandes empresas del sector como *Intel* [3], *Mellanox* [11], *TopSpin* [1], *Voltaire* [21], etc.

Se está haciendo un gran esfuerzo para que InfiniBand se considere una alternativa factible. Las especificaciones de InfiniBand están disponibles para el público lo que conlleva que existan muchos proyectos de investigación, comerciales y educativos, orientados a InfiniBand. Las especificaciones dejan un gran margen de maniobra a los investigadores, sobre todo en la parte del sistema operativo. El elemento más importante en la red InfiniBand es el *Subnet Manager*, responsable de configurar la red y controlar su buen funcionamiento. La especificación marca las responsabilidades de un *Subnet Manager* pero deja abiertos los métodos concretos de implementación, por ejemplo, define la tabla de encaminamiento pero no el algoritmo que el subnet manager sigue para rellenar dicha tabla.

Para InfiniBand se han desarrollado varios Subnet Managers. Por ejemplo, Mellanox tiene software que sirve como Subnet Manager, *minism*, pero con una funcionalidad muy básica. Cada fabricante, a su vez, posee versiones de otros Subnet Managers propios. Afortunadamente, muchos de estos fabricantes están poniendo mucho empeño para desarrollar una versión de Subnet Manager llamado *OpenSM* y que muchos destacan como la más importante en el mundo [18]. Si los investigadores pudieran conocerla y usarla se les abriría un abanico de grandes posibilidades de que sus propuestas sean implementas en ella y de este modo conseguir el reconocimiento perfecto para su trabajo.

El grupo de investigación de Redes y Arquitecturas de Altas Prestaciones del Instituto de Investigación en Informática de Albacete (I^3A) ha adquirido los dispositivos necesarios para montar una red InfiniBand. Se trata por el momento de una configuración básica compuesta por un switch y tres hosts, pero que permite analizar las características de InfiniBand y evaluar el comportamiento y viabilidad de las propuestas que se puedan hacer en el futuro mediante otros proyectos fin de carrera, trabajos académicamente dirigidos, tesis doctorales, etc.

El trabajo que se ha realizado correspondiente a este proyecto fin de carrera está centrado precisamente en ese material. La idea básica es estudiar las características de los dispositivos disponibles, formar con ellos una red InfiniBand y ponerla en marcha.

El manejo de productos reales, desconocidos y de muy reciente aparición en el mercado siempre entraña una dificultad que no se sabe apreciar hasta que se hace uso de ellos. Además, el software que ha de usarse sobre esta plataforma está en pleno desarrollo y sometido a contínuos cambios, lo que hace aún más complicado su manejo.

En una red InfiniBand el Subnet Manager es un componente de extrema importancia puesto que es el encargado de configurar la red y controlar el funcionamiento de los dispositivos. Dicho comportamiento está determinado por los algoritmos que el Subnet

Manager posee en su interior. Este proyecto fin de carrera definió el objetivo de conocer los algoritmos concretos que se implementan en OpenSM para las tareas principales de un Subnet Manager y de esta forma prestar la ayuda inicial, y no menos importante, a otros para que puedan desarrollar propuestas nuevas.

A pesar de las dificultades antes mencionadas, finalmente se han conseguido cubrir los objetivos planteados en un principio. Uno de ellos, quizás el más importante, es que este trabajo allana el camino a otros posteriores que pudieran realizarse con este material, o una posible ampliación del mismo. Con la experiencia adquirida y la documentación generada, el estudio de propuestas existentes o nuevas sobre este tipo de redes será seguramente más sencillo y rápido.

El contenido de este proyecto fin de carrera es el siguiente: El Capítulo 2 presenta los objetivos del proyecto y la metodología que se ha seguido para llevarlos a cabo. En el Capítulo 3 se hace un resumen del estado del arte en cuanto a la especificación de InfiniBand. El estudio de los dispositivos hardware disponibles y paquetes software que se han instalado y estudiado se exponen en el Capítulo 4, y es en el Capítulo 5 donde se explica como poner en marcha el sistema. En el Capítulo 6 se estudia el Subnet Manager *OpenSM* y para terminar, en el Capítulo 7 se presentan las conclusiones obtenidas y se plantea el trabajo futuro.

Capítulo 2

OBJETIVOS Y METODOLOGÍA

En este capítulo se indican los objetivos del proyecto, señalando, además, cuáles son las etapas a seguir para conseguir alcanzarlos. Se comenzará viendo de forma detallada todos los objetivos que se pretendían cubrir al inicio de este proyecto. A continuación, se verá la metodología usada y las distintas etapas seguidas a lo largo de este trabajo.

2.1. OBJETIVOS

Los objetivos principales de este proyecto son la puesta en marcha y la administración de la red de altas prestaciones InfiniBand adquirida por el grupo RAAP y el análisis del Subnet Manager llamado OpenSM.

Para lograr este objetivo general inicialmente se plantearon una serie de metas u objetivos parciales que se resumen de la siguiente forma:

- **Estudio de los componentes de la red**

  Puesto que el trabajo desarrollado se ha centrado en una red InfiniBand, es fundamental conocer con cierto detalle sus características, estructura y funcionamiento, para en definitiva, sacarle el máximo rendimiento.

- **Puesta en marcha de la red**

  Una vez cubierto el objetivo anterior, se trata de poner en funcionamiento todos los componentes, y de esta forma tener la red InfiniBand en funcionamiento.

- **Migración del sistema operativo de los Nitro II**

  En el caso particular de los nodos Nitro II (Sección 4.1.5) es muy interesante conocer las acciones necesarias para poder cambiar el sistema operativo que ejecutan, a pesar de que el sistema operativo que proporciona el fabricante funciona. Se trata con esto de conseguir flexibilidad para los trabajos de investigación posteriores.

- **Estudio del código de un Subnet Manager**

  Se trata de estudiar el código de un Subnet Manager, centrándose en su estructura y características, para después, comprender en profundidad cómo implementa las tareas más importantes de las que un Subnet Manager es responsable, como pueden ser la configuración e inicialización de la red.

2.2. METODOLOGÍA

Para alcanzar los objetivos parciales, y con ello el objetivo principal del proyecto, se plantearon al inicio del mismo una serie de tareas adecuadamente programadas según la metodología habitual en este tipo de trabajos. Se indican en este punto cuáles han sido y en qué han consistido estas tareas:

- **Estudio del estado del arte de las redes InfiniBand**

  Como en todos los proyectos, en primer lugar se debe conocer la arquitectura sobre la que se trabaja, en nuestro caso, Infiniband, aunque no sea necesario profundizar en esta primera etapa. Se utilizará toda la bibliografía disponible, esto es, manuales, libros, ponencias de congresos, páginas web, etc. Paralelamente, se tratará de conocer los proyectos que se están desarrollando sobre esta tecnología, principalmente en el tema de gestión.

- **Estudio de la documentación específica del producto disponible**

  Se revisará toda la documentación que el fabricante proporciona a través de su página web, centrándose en encontrar información relevante para su uso posterior tanto en la administración de la red como en la realización de esta memoria.

- **Estudio de los dispositivos hardware disponibles**

  Una parte del trabajo de búsqueda debe enfocarse a cada uno de los dispositivos, para analizar su estructura y características hardware, de cara a la puesta en marcha y administración de la red.

- **Estudio del software disponible para los dispositivos hardware**

  A pesar de toda la información descriptiva del hardware, se hace imprescindible estudiar qué software será adecuado tener instalado. Es muy importante conocerlo y saber utilizarlo.

- **Estudio del sistema de arranque de los nodos Nitro II**

  Los nodos Nitro II tienen ciertas características que los diferencian de un ordenador estándar. Es necesario estudiar el sistema de arranque que utilizan.

- **Estudio del proceso de construcción de un sistema operativo GNU/Linux para los Nitro II**

  Con intención de instalar un sistema GNU/Linux sobre la plataforma estudiada, es necesario hacer un estudio en profundidad de los pasos a seguir, primero en una instalación genérica de este sistema, y más tarde, con las modificaciones necesarias para su adaptación. Con todo ello, se podrá reconstruir los pasos que el fabricante dio para construir un sistema operativo adecuado para los Nitro II.

- **Análisis del modelo de gestión definido en la especificación de InfiniBand**

  En la especificación se definen unos mecanismos concretos destinados a la gestión de red y que todos los componentes deben implementar. Es necesario buscarlos en la especificación y analizarlos, para poder adquirir los conocimientos teóricos suficientes para poder analizar el código fuente de los gestores de red.

- **Análisis del código de OpenSM**

  Una vez que se conoce cómo la especificación define los mecanismos de la gestión, se puede abordar cómo se han implementado en un gestor real. El análisis comenzará desde un punto de vista general, tipos de datos, funciones, localización de módulos, comentarios, manuales, etc, para terminar en las funciones concretas donde se implementan las tareas principales, y de ahí poder extraer los algoritmos utilizados.

Capítulo 3

INFINIBAND

En este capítulo se presenta una breve descripción de InfiniBand analizando sus características principales. Este capítulo no pretende ser un recorrido exhaustivo por las especificaciones de InfiniBand. Si el lector está interesado, o necesita más detalles sobre alguno de los aspectos aquí tratados, puede consultar las especificaciones de InfiniBand [8] o alguno de los libros publicados sobre InfiniBand [6, 20].

3.1. DESCRIPCIÓN

A finales de los años noventa la idea de InfiniBand surge de la fusión de dos iniciativas existentes: NGIO *(Next Generation I/O)* y FIO *(Future I/O)*. Ambas compartían una buena parte de sus metas, y pronto se vió que no había mercado para las dos. Se decidió hacerlas converger en una única propuesta. De esta forma, en octubre de 1999 se fundó la IBTA *(InfiniBand Trade Association)*.

En octubre del año 2000 se publicó la primera versión, 1.0, de las especificaciones de InfiniBand [8]. En junio del año 2001 se publicó la primera modificación a las especificaciones, 1.0a [9], que consistió en corrección de errores e introducción de algunas anotaciones. En noviembre del año 2002 se publicó una versión con todas las características adicionales 1.1. En junio del año 2003 se publicó las correcciones a erratas en la especificación 1.1.

InfiniBand es una tecnología de interconexión entre sistemas de procesamiento y dispositivos de E/S que permite formar una red de área de sistema (Fig. 3.1). La arquitectura definida por InfiniBand es independiente del sistema operativo y de la plataforma.

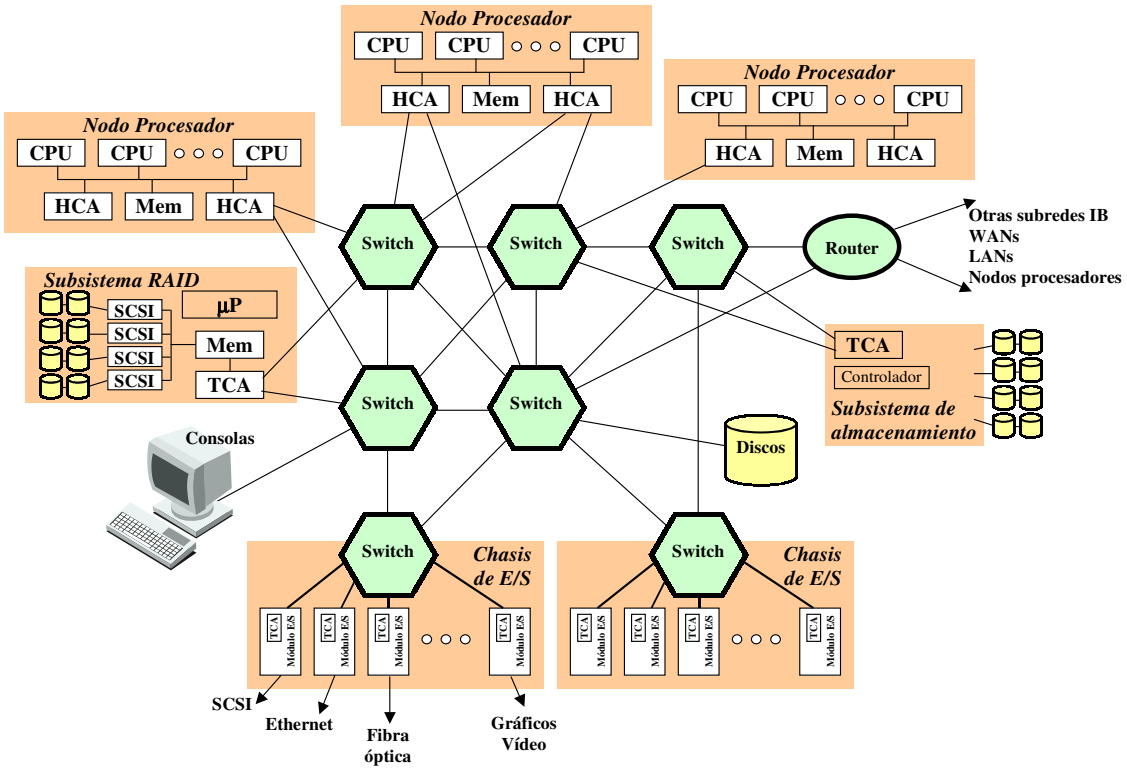


Figura 3.1: Red de área de sistema con InfiniBand.

Las dos principales metas que en un principio se plantea InfiniBand son: salvar las limitaciones que presentan los actuales buses PCI (por ejemplo, cuellos de botella, fiabilidad, escalabilidad, etc.), y estandarizar las emergentes tecnologías propietarias en el terreno de los clusters (por ejemplo, Servernet, Myricom, Giganet, etc.).

Sin embargo, InfiniBand pretende ir mucho más allá que una simple sustitución del típico bus PCI. InfiniBand incorpora características que hasta ahora sólo podían encontrarse en supercomputadores grandes y costosos. Estas características son importantes para el montaje de clusters de altas prestaciones y permiten aprovechar las posibilidades de la tecnología actual.

3.2. ARQUITECTURA

InfiniBand define una red de área de sistema (System Area Network, SAN) (Fig. 3.1) para conectar ordenadores, sistemas de E/S y dispositivos de E/S. InfiniBand proporciona la infraestructura adecuada para comunicación y gestión, tanto para transacciones de E/S como para comunicación entre ordenadores. Un sistema InfiniBand puede variar desde un pequeño servidor formado por un procesador y unos cuantos dispositivos de E/S conectados, hasta un supercomputador masivamente paralelo con miles de procesadores y

dispositivos de E/S que está conectado vía Internet a otras plataformas de procesamiento y/o sistemas de E/S.

InfiniBand define una interconexión conmutada que permite a muchos dispositivos intercambiar datos de forma simultánea, con gran ancho de banda y baja latencia. Al ser un sistema conmutado, se pueden conseguir características como protección, fiabilidad, escalabilidad, seguridad, etc, hasta ahora impensables en sistemas de E/S, e incluso en la mayoría de las redes habituales para conexión de computadores.

Un nodo final en una SAN InfiniBand puede comunicarse por medio de múltiples puertos del conmutador al que está conectado, pudiéndose habilitar de esta manera caminos alternativos. Así, se podría aprovechar la disponibilidad de caminos alternativos tanto para incrementar el ancho de banda real, como para permitir tolerancia a fallos.

InfiniBand permite a las unidades de E/S comunicarse entre ellas y con cualquier sistema de procesamiento existente en el sistema. De esta manera, una unidad de E/S tiene la misma capacidad de comunicación que cualquier otro nodo de procesamiento.

3.2.1. TOPOLOGÍA

InfiniBand tiene una topología conmutada con conexiones punto a punto, lo que permite tanto topologías regulares como irregulares. El sistema de administración será capaz de identificar cualquier topología formada, y construir las tablas de encaminamiento adecuadas para permitir el intercambio de información entre dos elementos cualquiera conectados a través de la red.

Desde un punto de vista de alto nivel, InfiniBand sólo es un medio para interconectar nodos entre sí (Fig. 3.2), donde un nodo puede ser un sistema de procesamiento, una unidad de E/S o un encaminador hacia otra red.

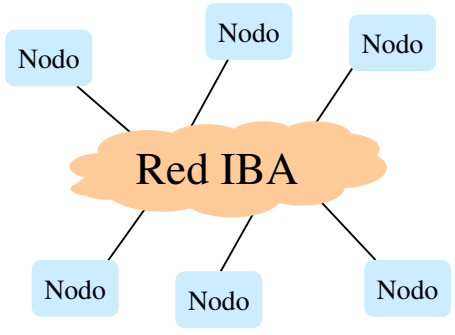


Figura 3.2: Red InfiniBand.

Una red InfiniBand está dividida en subredes interconectadas entre sí mediante routers o encaminadores (Fig. 3.3). Los nodos finales estarán conectados a una única subred o a múltiples subredes por medio de distintas interfaces.

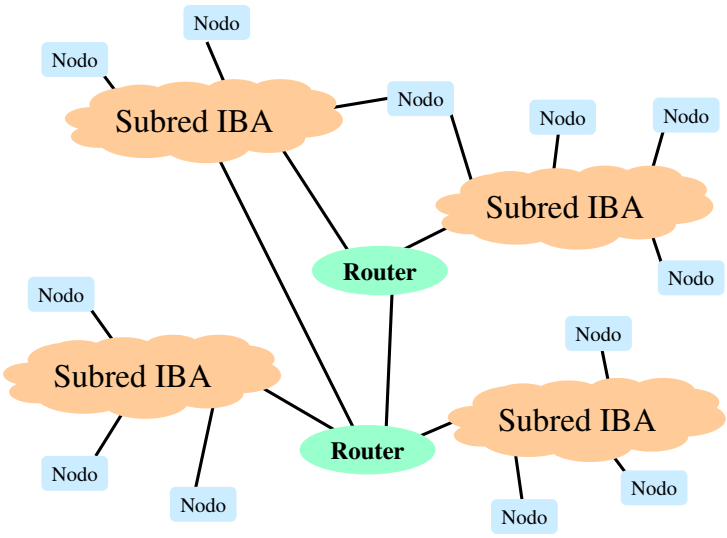


Figura 3.3: Componentes en una red InfiniBand.

3.2.2. COMPONENTES DE INFINIBAND

Una subred InfiniBand puede estar compuesta por nodos finales, conmutadores, encaminadores y el Subnet Manager, todos ellos interconectados por enlaces (Fig. 3.4). Estos enlaces pueden ser de cable, de fibra óptica o grabados en la placa base.

Entre cualquiera de los componentes de la red pueden existir múltiples enlaces, consiguiendo de esta forma incrementar el ancho de banda, así como mejorar la tolerancia a fallos. Además, cualquiera de los componentes de la red InfiniBand puede estar conectado a un único conmutador, a varios, o directamente con cualquier otro dispositivo sin pasar por un conmutador. Sin embargo, la conexión directa entre dos nodos finales forma una subred independiente, sin conexión con el resto de dispositivos. En este caso, ambos se pondrán de acuerdo para que uno de ellos actúe como Subnet Manager.

Los dispositivos en un sistema InfiniBand están clasificados como:

Enlaces y repetidores

Los *enlaces* (Links) interconectan channels adapters, switches, routers y repetidores formando una estructura. Los enlaces son cables de cobre, de fibra óptica o impresos

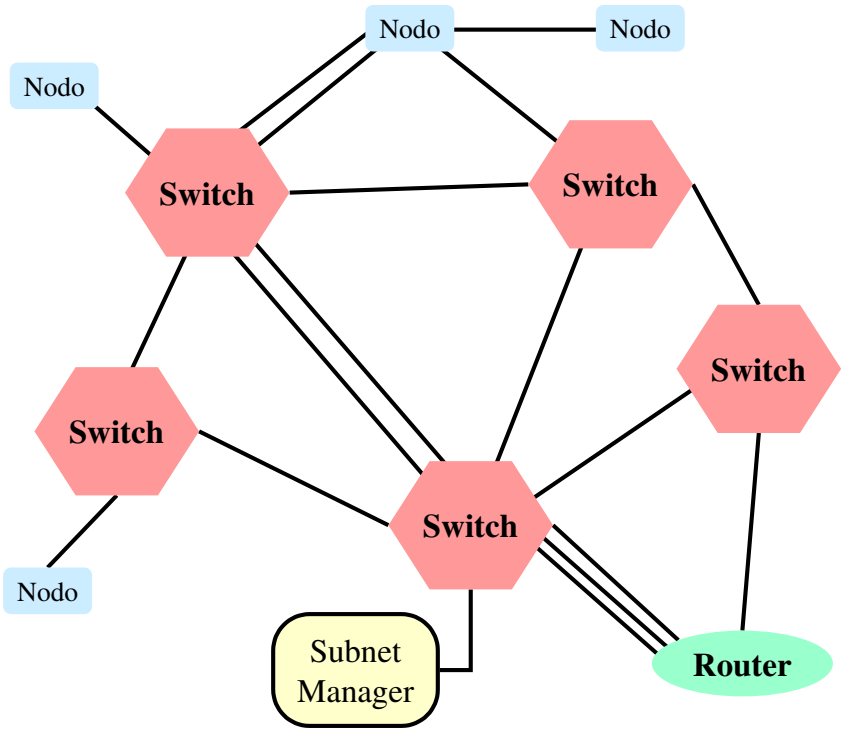


Figura 3.4: Componentes en una subred InfiniBand.

en circuitos.

Los *repetidores* (Repeaters) son dispositivos transparentes que extienden el alcance de los enlaces. Sólo participan en los protocolos de la capa física y los nodos no detectan su presencia. En el volumen 2 de la especificación [9] se describe los cables y repetidores para varios tipos de medios físicos.

Channel adapters

Un *channel adapter* (CA) es el componente en un nodo final que lo conecta a la red. Un channel adapter es un dispositivo DMA programable con características especiales de protección, que permite que las operaciones de DMA sean iniciadas de forma local o remota. Cada channel adapter tiene uno o varios puertos, tal y como puede verse en la Figura 3.5. Normalmente cada uno de ellos suele estar conectado a un puerto de un conmutador, aunque también es posible conectarlos entre sí directamente.

Cada puerto del channel adapter tiene su propia dirección local configurada por el Subnet Manager. La entidad local que se comunica con el Subnet Manager para la configuración del channel adapter es el *Subnet Management Agent* (SMA). Por otra parte, cada puerto tiene su propio conjunto de buffers de envío y recepción, y es capaz de estar

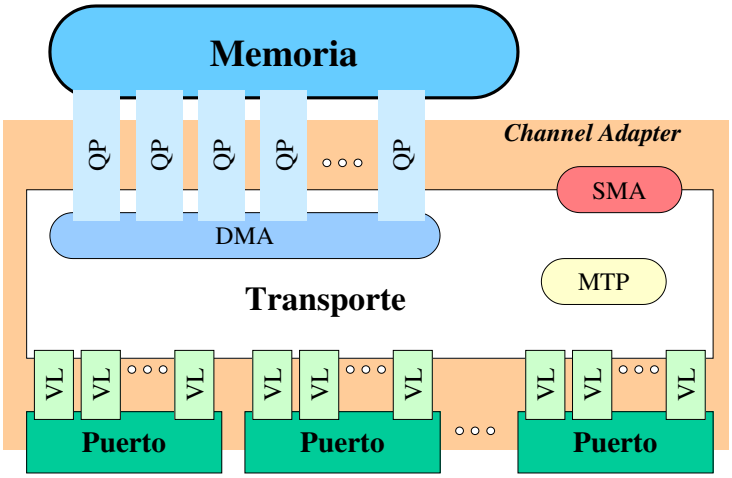


Figura 3.5: Estructura de un channel adapter.

enviando y recibiendo información al mismo tiempo. El almacenamiento de los buffers está distribuido en varios canales virtuales (VL), teniendo cada VL su propio control de flujo.

InfiniBand permite el particionado virtual de la red por medio de la definición de dominios de acceso lógico. Cada puerto de cada channel adapter pertenece al menos a una partición y puede comunicarse sólo con otros puertos que también pertenezcan a alguna de sus particiones. De esta forma, la propia red proporciona un nivel de protección.

Los channel adapters pueden clasificarse en:

Host Channel Adapter (HCA) Un HCA es para sistemas de procesamiento con gran capacidad para ejecutar distinto tipo de software.

Target Channel Adapter (TCA) Un TCA es para dispositivos de E/S, que por su simplicidad, no suelen tener capacidad para ejecutar software.

Conceptualmente tanto el HCA como el TCA son idénticos. Desde un punto de vista arquitectónico, el HCA difiere del TCA porque el HCA tiene una interfaz más estructurada que ofrece al sistema operativo.

Conmutadores

Los *conmutadores* (Switches) (Fig. 3.6) disponen de una serie de puertos donde se conectan los enlaces. A todos los efectos, los conmutadores son completamente transparentes para los nodos finales. Es decir, ellos no saben de su existencia, ni mucho menos

les son accesibles.

En contraste con los channel adapters, los conmutadores ni generan ni consumen paquetes, salvo los propios de gestión de la subred. Los conmutadores simplemente transmiten paquetes desde un puerto de entrada hacia un puerto de salida, en base a la dirección destino que el paquete lleva en su cabecera.

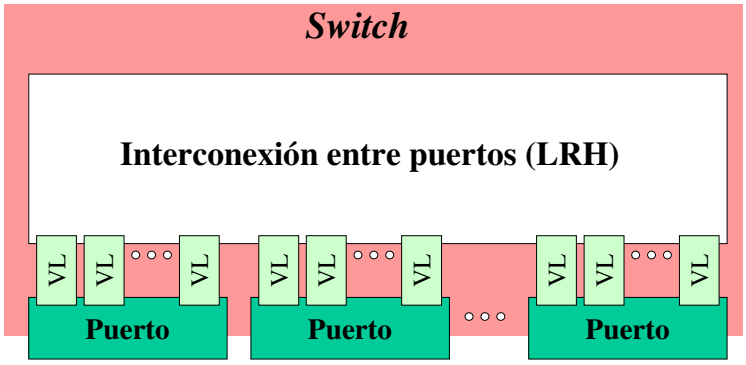


Figura 3.6: Estructura de un conmutador de InfiniBand.

Los paquetes que cruzan los conmutadores permanecen inalterados en su contenido, aunque sí puede que sufran cambios en su cabecera.

Los conmutadores tienen unas tablas de encaminamiento para decidir por qué puerto sacar cada paquete. Estas tablas se cargan en el momento del arranque, o tras un cambio en la topología, por el Subnet Manager. El conmutador, en base a esas tablas y a la dirección destino que viene especificada en la cabecera del paquete, debe decidir por qué puerto reenviar cada paquete que le llega. Según las especificaciones, no todos los conmutadores deben tener capacidad de reenvío multidestino, dejando este aspecto a criterio de los fabricantes.

Encaminadores

Un *encaminador* (Router) se encarga de comunicar entre sí distintas subredes. Los encaminadores tampoco tienen capacidad para generar o consumir paquetes (sin contar los de control). Nuevamente, se limitan a trasvasarlos desde uno de sus puertos de entrada a uno de los de salida. En este caso la decisión de qué puerto de salida se toma en base a una dirección global (GID). Esta dirección global es única en todas las redes InfiniBand, mientras que la dirección local (LID) es propia de cada subred.

El esquema básico de un encaminador se muestra en la Figura 3.7. Al igual que los conmutadores, los encaminadores tienen varios puertos con capacidad de recibir y

transmitir de forma simultánea. El espacio para almacenamiento también está distribuido entre varios canales virtuales.

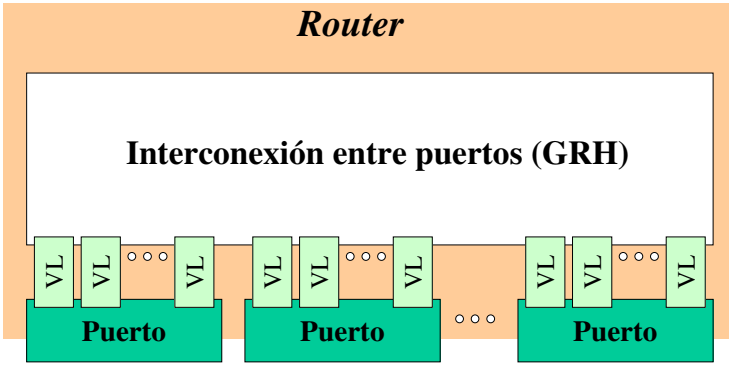


Figura 3.7: Estructura de un encaminador de InfiniBand.

Los encaminadores no son completamente transparentes para los nodos finales, puesto que el nodo fuente debe especificar la dirección local (LID) del encaminador de salida al exterior de su subred, además de indicar la dirección global (GID) del nodo destino.

Cada encaminador envía los paquetes hacia la siguiente subred a otro encaminador, hasta que el paquete llega a la subred destino. El último encaminador envía el paquete al nodo final utilizando la dirección local (LID) que se especificó en el origen.

3.2.3. CARACTERÍSTICAS DE INFINIBAND

Pares de colas

Los *pares de colas* (queue pairs, o QP) son la interfaz virtual (Fig. 3.8) que el hardware proporciona a un productor de información en InfiniBand, y el puerto de comunicación virtual que proporciona para el consumidor de dicha información. De esta forma, la comunicación tiene lugar entre un QP fuente y un QP destino. La arquitectura permite hasta 2^{24} QPs por channel adapter, de forma que las operaciones en cada QP son independientes de las del resto de QPs.

Cada QP permite tener aislada y protegida esa comunicación de la del resto de QPs u otros consumidores. Sin embargo, un QP puede ser considerado un recurso privado asignado a un único consumidor. No obstante, un consumidor probablemente consumirá de múltiples QPs (Fig. 3.8).

Todos los tipos de servicio, salvo el *raw datagram*, necesitan tener establecido

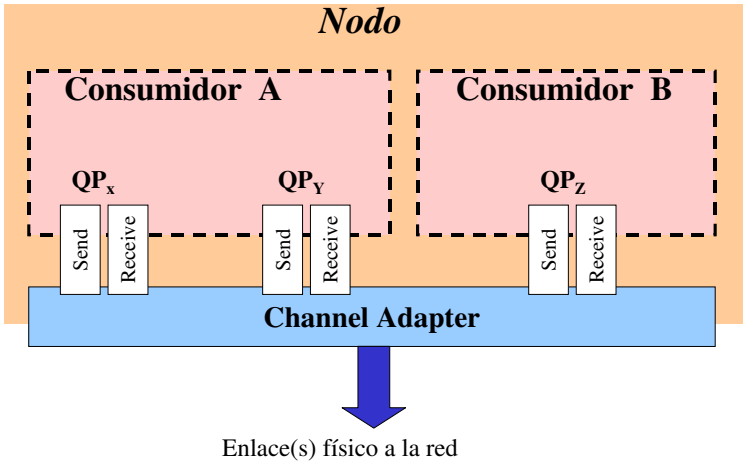


Figura 3.8: Interfaz para la comunicación en InfiniBand.

un QP asociado. Para los tipos de servicio con conexión, se establece una comunicación previa entre ambos extremos para asociar un QP origen con un único QP destino y configurar el contexto de las QPs con cierta información como LID destino, nivel de servicio y límites operativos. Los tipos de servicio sin conexión utilizan un QP genérico compartido por varias aplicaciones.

Tipos de servicio

Cada QP se configura para proporcionar un determinado tipo de servicio. Dicho tipo de servicio está basado en cómo interactúan los QPs fuente y destino, que deben estar configurados con el mismo tipo de servicio. Hay tres características que identifican a cada tipo de servicio:

- Servicio orientado a conexión frente a no orientado a conexión. El servicio no orientado a conexión suele llamarse también *datagrama*. En un tipo de servicio orientado a conexión cada QP fuente está asociado con un único QP destino, y viceversa. En un tipo de servicio no orientado a conexión está permitido que un QP envíe/reciba paquetes a/desde cualquier otro QP en cualquier nodo.

- Servicio con confirmación frente a sin confirmación. En un servicio confirmado, cuando un QP recibe un paquete debe confirmar al QP origen que lo ha recibido correctamente. Estos mensajes de confirmación pueden ir integrados en otro paquete con información, o en un mensaje ACK o NAK (negative acknowledged) propio. El servicio confirmado se dice que es fiable, pues el protocolo de transporte garantiza un envío sin errores y con entrega en orden para el posterior reensamblado de los paquetes en un mensaje de nivel superior. Por contra, el servicio sin confirmación se

dice que es no fiable pues el protocolo de transporte no asegura que la información llegue a su destino.

- Servicio de transporte de InfiniBand frente a otro tipo de transporte. Como se verá más adelante, el servicio de transporte de InfiniBand permite transmitir paquetes *en bruto* encapsulando paquetes de otros protocolos de transporte, como por ejemplo IPv6, o de otros tipos de redes.

Así pues, los tipos de servicios definidos en InfiniBand, y sus características, son los mostrados en la Tabla 3.1.

Tipo de Servicio	Orientado a Conexión	Confirmado	Transporte
Reliable Connection	Sí	Sí	InfiniBand
Unreliable Connection	Sí	No	InfiniBand
Reliable Datagram	No	Sí	InfiniBand
Unreliable Datagram	No	No	InfiniBand
Raw Datagram	No	No	En bruto

Cuadro 3.1: Tipos de servicio de transporte en InfiniBand.

Claves

Las *claves* (keys) proporcionan un cierto nivel de aislamiento y protección del tráfico. Son unos valores asignados por las entidades de administración y que luego se insertarán en los paquetes según la función y destino al que vayan dirigidos. Las aplicaciones sólo podrán acceder a los paquetes que contengan claves para los que ellas estén habilitadas. Los diferentes tipos de claves son:

- **Management Key** (M_Key). Esta clave se usa para tareas de gestión. Las administra el Subnet Manager, que puede asignar una distinta a cada puerto. Una vez hecha la asignación, todo el tráfico de control con ese puerto deberá llevar esa clave insertada en los paquetes.
- **Baseboard Management Key** (B_Key). Permite que actúe el gestor de subred en placa (subnet baseboard manager). Esta clave la contienen cierto tipo de paquetes de gestión de la subred.
- **Partition Key** (P_Key). Permite la división lógica de la subred en distintas zonas. Cada adaptador contiene una tabla de claves de partición que define las particiones

para las que ese adaptador está habilitado. Hay un gestor de particiones (Partition Manager, PM) único, que se encarga de gestionar las claves de las particiones.

- **Queue Key** (Q_Key). Permite controlar el derecho de acceso a las colas para los servicios sin conexión (datagram). De esta forma, dos nodos que no hayan establecido previamente una conexión pueden intercambiar información de forma que esta clave identifique unívocamente a los interlocutores.

- **Memory Keys** (L_Key y R_Key). Permite el uso de direcciones de memoria virtuales y dota al consumidor de un mecanismo para controlar el acceso a dicha memoria. El consumidor le especifica al adaptador una zona de memoria y recibe de éste una L_Key y otra R_Key. El consumidor usa la L_Key en las gestiones locales de memoria, y pasa la R_Key a los consumidores remotos para que la usen en las operaciones remotas de DMA (RDMA).

Las claves no proporcionan seguridad por sí solas pues dichas claves están disponibles en la cabecera de los paquetes que circulan por la red. No obstante, permiten hacer un aislamiento virtual de forma que cada adaptador sepa qué mensajes le atañen directamente a él, o van dirigidos a otro tipo de entidades.

Direcciones de memoria virtual

La arquitectura de InfiniBand está optimizada para manejar direccionamiento virtual (virtual memory addresses). Un consumidor puede especificar direcciones virtuales y es el adaptador el que convierte la dirección virtual a una dirección física. Un consumidor registra regiones de memoria virtual con su adaptador, y éste le devuelve dos claves llamadas L_Key y R_Key. A partir de ese momento el consumidor utiliza L_Key en las peticiones que requieran un acceso a la memoria de esa región.

InfiniBand también permite el acceso remoto (RDMA) a una región de memoria previamente registrada en el adaptador. Para el RDMA el consumidor pasa a otro consumidor remoto la R_Key y una dirección virtual de un buffer en esa región de memoria. Ese consumidor remoto utiliza en sus accesos a esa zona remota de memoria la R_Key que le han proporcionado.

Dominios protegidos

Los dominios protegidos (protection domain) permiten conceder distintos modos de acceso a las zonas de memoria registrada. Un consumidor establece el conjunto de QPs que pueden acceder a sus distintas zonas de memoria y los permisos que les da.

Antes que un consumidor reserve un QP o registre una zona de memoria, debe crear uno o más dominios protegidos. Tanto los QPs como las zonas de memoria registradas están asociadas a un dominio protegido. Tanto las claves L_Keys como las R_Keys de un dominio de memoria concreto sólo son válidas para los QPs creados para el mismo dominio protegido.

Particiones

Las particiones (partitions) permiten aislar de forma virtual zonas de una subred InfiniBand. Este particionamiento no está ligado a límites establecidos por subredes, conmutadores o encaminadores. Son zonas virtuales que permiten aislar ciertas zonas de la subred.

Cada puerto o interfaz de la subred es miembro de al menos una partición y puede pertenecer a varias de ellas. El gestor de particiones asigna claves de partición (P_Keys) a cada adaptador de la subred. Estas claves se insertan en los paquetes que se envían luego a la red. Cuando una entidad de la subred recibe un paquete con una P_Key para la que no está habilitado debe descartar dicho paquete.

Los conmutadores y los encaminadores pueden ser configurados por el gestor de particiones para que también descarten paquetes aislando físicamente zonas de la subred.

Canales virtuales

Los canales virtuales (VL) constituyen un mecanismo para crear múltiples enlaces virtuales con un único enlace físico. Un canal virtual representa un conjunto de buffers de transmisión y recepción en un puerto (Fig. 3.9).

Todos los puertos soportan un VL_{15} exclusivamente para la gestión de la subred. También deben soportar como mínimo un canal virtual para datos, que es el VL_0. El resto de canales virtuales de datos (VL_1 a VL_{14}) son opcionales. Dependiendo de la implementación, un puerto puede tener 16, 8, 4 ó 2 canales virtuales implementados.

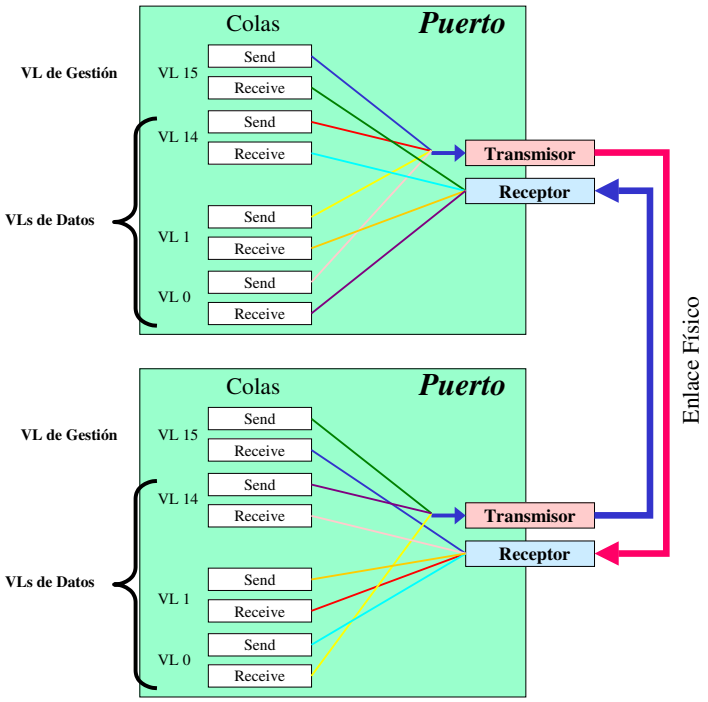


Figura 3.9: Multiplexación del enlace físico en canales virtuales.

El VL que en cada momento se usa viene configurado por el Subnet Manager, y esta decisión está basada en el campo nivel de servicio (Service Level, SL) que llevan los paquetes. En concreto, existe una tabla *SLtoVLMappingTable* que especifica la correspondencia entre los SLs y VLs.

El tráfico de cada VL no influye en el tráfico de otros VLs. Cada puerto mantiene control de flujo separado para cada VL de datos. Cada paquete incorpora en su cabecera un SL. Por otra parte, cada puerto mantiene la tabla de correspondencia entre SLs y VLs. Así, cuando el paquete va fluyendo por la subred es ese SL que tiene en su cabecera el que va indicando el VL a usar en cada caso.

Cada puerto debe adaptarse al mínimo número entre los que él incorpora y los que tiene el puerto con el que está conectado. Esto se realiza durante la fase de establecimiento de la subred.

Control de la tasa de inyección

InfiniBand define enlaces serie punto a punto full-duplex funcionando a una frecuencia de 2,5 GHz. La velocidad de transmisión que se obtiene es 2,5 Gb/seg, que se denomina 1X. Sin embargo, InfiniBand permite alcanzar mayores velocidades usando varios de esos enlaces en paralelo. De esta forma, otras velocidades que también están

definidas en InfiniBand son 10 Gb/seg (4X) y 30 Gb/seg (12X).

De cara a ser capaz de utilizar componentes que funcionen a diferentes velocidades, InfiniBand define un mecanismo de control de inyección que previene que un puerto pueda llegar a saturar a otro más lento (Fig. 3.10).

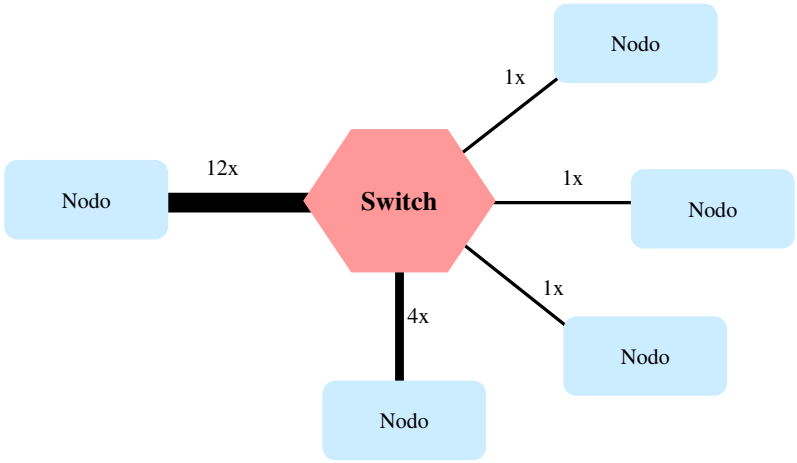


Figura 3.10: Ejemplo de una subred con enlaces de diferentes velocidades.

El control de flujo aunque no soluciona este problema, evitará que se pierdan paquetes y provocará congestión hacia atrás en la ruta. De esta forma, se llegarán a congestionar incluso aquellos enlaces que si no fuera por este cuello de botella no se habrían visto afectados. InfiniBand soluciona este problema por medio de un mecanismo de control de inyección que deben incorporar los puertos que funcionen a velocidades mayores de 1X.

Cada fuente tiene un tiempo de time-out asociado a cada destino. Ese valor ha sido calculado como el ratio entre las velocidades de la fuente y de los enlaces intermedios hasta llegar al destino. De esta forma, cada vez que el origen envía un paquete, debe esperar a que transcurra ese tiempo antes de enviar otro paquete a la misma fuente.

Direccionamiento

Cada nodo final puede tener uno o más adaptadores y a su vez éstos pueden tener uno o más puertos. Además, cada adaptador tiene un número variable de pares de colas (QPs). Cada QP tiene asignado un número (queue pair number, QPN) por parte del adaptador, que lo identifica de forma única.

Cada puerto tiene un identificador local (LID) asignado por el Subnet Manager. Dentro de la subred estos LIDs son únicos y se usan para encaminar los paquetes.

Cada conmutador de la subred tiene una tabla de encaminamiento que está basada en esos identificadores locales. De esta forma, cada paquete cuando viaja por la subred lleva incorporadas la dirección del puerto fuente (SLID) y la del puerto destino (DLID).

Cada puerto también tiene asignada una dirección global (GID) que sigue el formato de las direcciones IPv6. Estas GIDs son únicas en cualquier subred InfiniBand. Los paquetes que vayan a viajar entre distintas subredes deberán incorporar las direcciones globales fuente (SGID) y destino (DGID), que serán usadas por los encaminadores intermedios.

Así pues, la dirección que identifica a dos entidades que se comunican será la combinación de las direcciones de los puertos (GID + LID) y de los pares de colas que intervienen (QPN).

Multicast

Multicast es un paradigma de comunicación uno-muchos/muchos-muchos diseñado para simplificar y mejorar la comunicación entre un conjunto de nodos finales. InfiniBand permite la comunicación multidestino. No es obligatorio que los fabricantes lo incorporen, pero si lo hacen, sí está especificado cómo debe ser su comportamiento.

Cada grupo multidestino está identificado por una dirección LID y GID única. De esta forma, puede haber grupos multidestino en la misma subred o también entre distintas subredes. Cada nodo se une a un grupo multidestino indicando los LID de los puertos que van a participar. Esta información se distribuye a los conmutadores, que se configuran para saber a qué puertos deben viajar los paquetes de ese grupo multidestino. Evidentemente, se tiene en cuenta que no se formen ciclos en esas rutas multidestino.

Cuando un conmutador o un encaminador recibe un paquete multidestino lo reenvía por todos aquellos puertos a través de los que se accede a puertos que participen en el grupo multidestino, salvo por el puerto por donde llegó dicho paquete. De esta forma, el paquete llegará una sola vez a cada miembro del grupo multidestino.

Los channel adapter deben limitar el número de QPs que pueden registrarse para las mismas direcciones de multicast. Los channel adapter distribuyen los paquetes por estas QPs.

Verbs

InfiniBand describe el comportamiento de la interfaz entre el adaptador del host y el sistema operativo por medio de un conjunto de comportamientos llamados verbos. La intención de los verbos no es definir un API, sino el comportamiento de la interfaz. De esta forma, los diseñadores de los sistemas operativos deberán definir APIs adecuados que aprovechen las ventajas de esta arquitectura. No importa como estén implementados (software, hardware, firmware, etc), sólo importa que estén implementados.

Los verbos describen los parámetros necesarios para configurar y gestionar el adaptador, crear y liberar colas, configurar operaciones con las colas, hacer peticiones de envío a las colas, consultar el estado de las peticiones, etc.

3.3. ESTRUCTURA ARQUITECTÓNICA

El funcionamiento de InfiniBand se describe mediante la interacción de una serie de niveles. El protocolo que gobierna cada nivel es independiente del resto de niveles. Cada nivel usa los servicios que le proporciona el nivel inferior y a su vez proporciona una serie de servicios al nivel inmediatamente superior.

Los distintos niveles que distingue InfiniBand, y sus modos de actuación se muestran en la Figura 3.11. Por encima del nivel de transporte están los manejadores de los sistemas operativos.

3.3.1. NIVEL FÍSICO

El nivel físico especifica cómo se trasladan los bits al cable. InfiniBand utiliza el sistema de codificación 8B/10B. El nivel físico varía dependiendo del medio físico que se vaya a utilizar. InfiniBand especifica tres tipos de medios físicos: par trenzado, fibra óptica o circuito en placa. El nivel físico también especifica los símbolos que se usarán para indicar principio y final de paquete, datos, espacio entre paquetes, el protocolo de señalización a utilizar, etc.

La capa física está descrita en el volumen 2 de la especificación de InfiniBand [9].

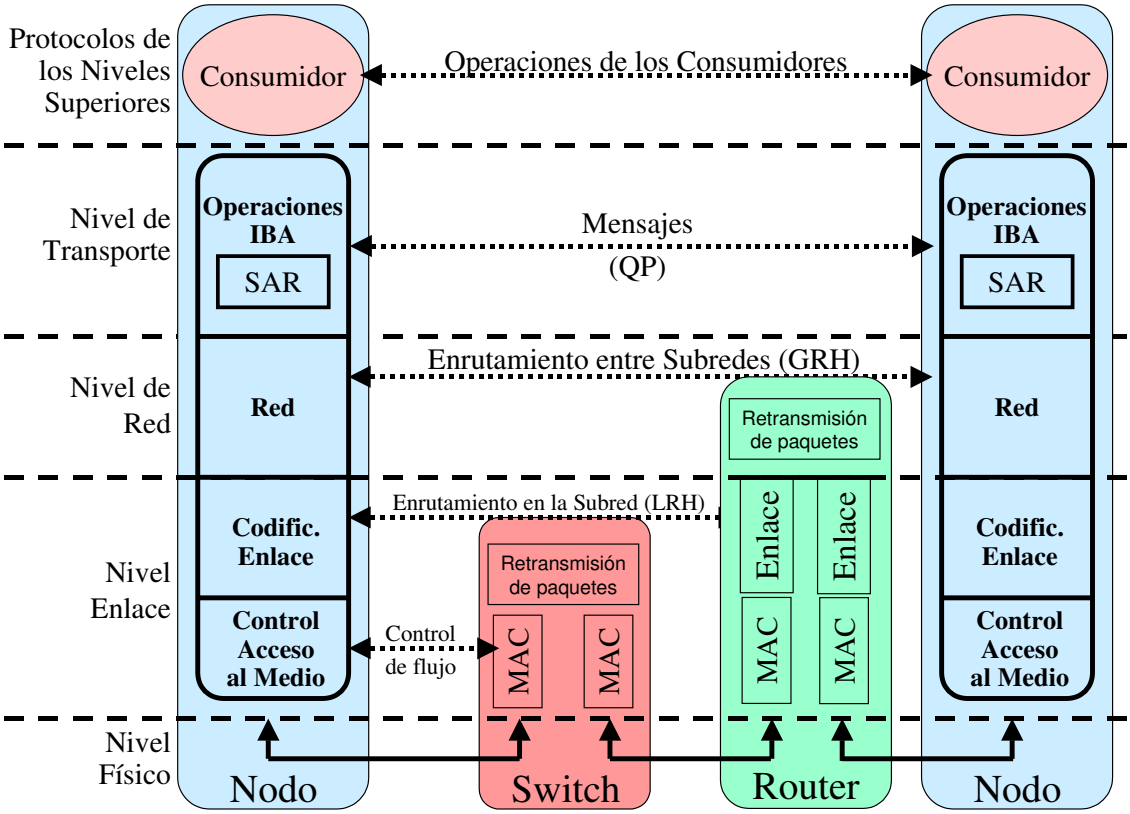


Figura 3.11: Niveles en la arquitectura de InfiniBand.

3.3.2. NIVEL DE ENLACE

El nivel de enlace describe los formatos de paquete a usar y los protocolos para las operaciones con ellos. También son tareas de este nivel el control de flujo y el encaminamiento de los paquetes dentro de la misma subred.

Se pueden distinguir dos tipos de paquetes:

Paquetes de gestión. Se utilizan para mantenimiento de las operaciones de enlace. Estos paquetes se crean y se consumen en el nivel de enlace, y no están sujetos a control de flujo. Se suelen usar para operaciones de negociación de parámetros entre los puertos de cada lado del enlace, como velocidad, tasa de inyección, etc. También son estos paquetes los que se utilizan para control de flujo y mantenimiento de la integridad del enlace. Estos paquetes sólo se envían entre ambos extremos de un enlace, y nunca se retransmiten hacia otro destino.

Paquetes de datos. Estos son los paquetes que intercambian las operaciones de InfiniBand y pueden contener varias cabeceras, algunas de las cuales son opcionales.

La Figura 3.12 muestra el formato del paquete del nivel de enlace. Se observa

como se encapsulan los paquetes de los niveles superiores en los paquetes que envían los niveles inferiores de la arquitectura.

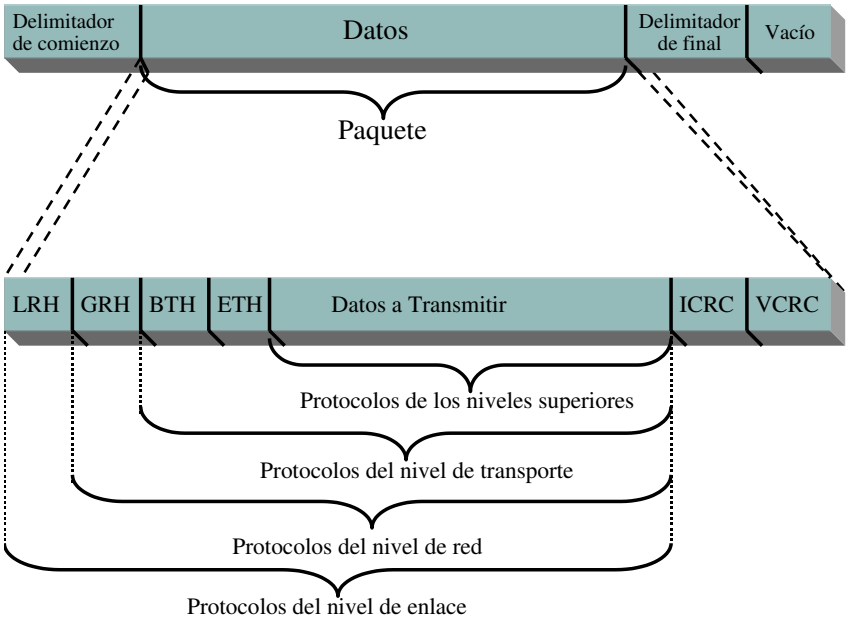


Figura 3.12: Formato del paquete de datos en InfiniBand.

La *Local Route Header* (LRH) siempre está presente e identifica los puertos origen y destino donde los conmutadores intermedios encaminarán el paquete. Esta cabecera también contiene el nivel de servicio y el canal virtual que debe utilizar el paquete. Este canal virtual podría cambiar a lo largo de la ruta, pero el resto de campos de esta cabecera permanecerán inalterados durante todo el trayecto.

El paquete también contiene dos CRCs:

Invariant CRC (ICRC) El ICRC cubre aquellos campos del paquete que no van a variar a lo largo de la ruta.

Variant CRC (VCRC) El VCRC cubre todo el paquete, con lo que deberá ser recalculado en cada conmutador o encaminador intermedio.

La combinación de ambos CRCs permite a los nodos intermedios modificar apropiadamente las cabeceras, controlando entre ellos los posibles errores locales, y sin embargo seguir manteniendo el control de la integridad del paquete extremo a extremo.

El control de flujo que realiza la capa de enlace está basado en créditos. En cada enlace, el extremo receptor envía al origen los créditos disponibles. Estos créditos se contabilizan para cada canal virtual, e indican el número de paquetes de datos que el receptor puede aceptar en un canal virtual. El extremo origen no enviará paquetes a menos

que el receptor le haya indicado previamente que tiene espacio disponible para alojarlos. Por otra parte, el canal virtual número 15 (el canal virtual de gestión) no está sujeto a control de flujo.

3.3.3. NIVEL DE RED

El nivel de red describe el protocolo para encaminar un paquete entre distintas subredes. Este nivel define una *Global Route Header* (GRH) que debe estar presente en los paquetes que tengan que ser encaminados entre dos o más subredes. La GRH identifica los puertos origen y destino usando direcciones globales (GID) en el formato de una dirección IPv6.

Los encaminadores se basan en la información que contiene la GRH para encaminar los paquetes hacia su destino. Conforme los paquetes van avanzando entre subredes los encaminadores van modificando el contenido del GRH y reemplazando el LRH. Sin embargo, los identificadores globales de los nodos fuente y destino no variarán a lo largo de toda la ruta, y están protegidos por el campo ICRC.

El nodo origen pone el identificador global (GID) del destino en la GRH pero el identificador local (LID) del encaminador en la LRH. Cada conmutador intermedio hace avanzar el paquete hacia la siguiente subred hasta que llega a la subred destino. El último encaminador reemplaza la LRH usando el identificador local del destino.

3.3.4. NIVEL DE TRANSPORTE

Los protocolos de enlace y de red llevan un paquete al destino deseado. La parte de nivel de transporte del paquete se encarga de hacer que se entregue el paquete en la cola (QP) apropiada, indicándole además cómo procesar los datos contenidos en el paquete.

El nivel de transporte es el responsable de segmentar una operación en varios paquetes si los datos a transmitir exceden el tamaño máximo de paquete (MTU). El QP en el extremo final reensambla los paquetes para formar la secuencia de datos que se quiso enviar.

La *Base Transport Header* (BTH) está en todos los paquetes, salvo en los datagramas RAW. Esta cabecera especifica el QP destino e indica el código de operación, el número de secuencia del paquete y la partición a la que pertenece.

El número de secuencia del paquete se inicializa durante el proceso de establecimiento de conexión y se incrementa cada vez que el QP crea un nuevo paquete. El QP receptor utiliza este número de secuencia para detectar si se ha perdido algún paquete. En el caso de los servicios fiables, el receptor enviará un paquete de asentimiento (ACK o NACK) notificando al remitente si el paquete se ha recibido correctamente.

Cuando ocurre un error en la secuencia es que se ha perdido algún paquete intermedio. En ese caso, el receptor descartará los paquetes siguientes que pudieran llegar hasta que llegue el paquete perdido. En el caso del servicio sin confirmación el receptor detecta un paquete perdido, aborta la operación y descarta todos los paquetes siguientes hasta que comience otra operación nueva.

Dependiendo de la clase de servicio utilizada y de la operación a realizar, un paquete podría llevar varias cabeceras extra (*Extended Transport Headers*, ETH). Será el QP destino el que deberá analizar las ETHs presentes y realizar las acciones oportunas según la operación a realizar. Para servicios orientados a la conexión, las ETH identifican el contexto EE que la QP usa para detectar paquetes perdidos. Una EE define un canal extremo a extremo que permite a múltiples QPs comunicarse entre sí.

El nivel de transporte es el núcleo central de la arquitectura de InfiniBand. Los servicios del nivel de transporte se acceden desde los QPs, donde cada QP se configura para una clase de servicio específico.

3.3.5. PROTOCOLOS DE LOS NIVELES SUPERIORES

InfiniBand soporta cualquier número de protocolos de nivel superior utilizados por varios usuarios (Fig. 3.13). También define los mensajes y protocolos para ciertas funciones de gestión. Estos protocolos están separados en Servicios de Subred y Gestión de la Subred.

InfiniBand es una plataforma para que las aplicaciones intercambien información.

3.4. INFRAESTRUCTURA DE GESTIÓN

La especificación de InfiniBand define los mensajes y protocolos para muchas de las tareas de gestión de la red. Estos protocolos de gestión se pueden agrupar en gestiones de la subred y servicios de la subred, teniendo cada uno de esos grupos propiedades

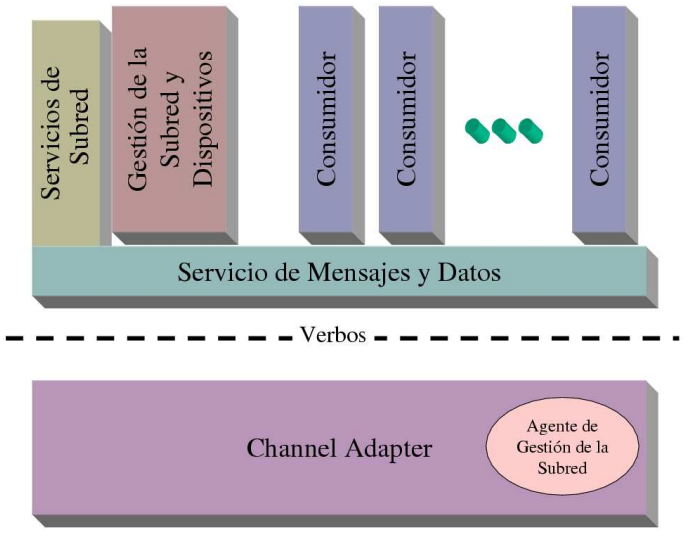


Figura 3.13: Niveles superiores en InfiniBand.

distintas.

En cada subred debe existir un *Subnet Manager* encargado de configurar la red y administrar la información necesaria para las operaciones de la subred. InfiniBand define la infraestructura básica para permitir diversas clases de gestión (*Management Classes*). Cada clase de gestión permite gestionar diversos aspectos de la red de forma independiente.

Para permitir la consistencia entre distintas clases, la infraestructura global de gestión define un conjunto común de acciones de gestión llamados métodos que pueden usar todas las clases. Cada clase puede definir los atributos a los que se les podrán aplicar esos métodos. De hecho, una clase define qué atributos son apropiados para cada método.

3.4.1. CLASES DE GESTIÓN

Algunas de estas clases son obligatorias y otras opcionales, y depende de los implementadores el que aparezcan o no en los productos finales. Ciertas clases necesitan un agente especial que proporcione ese servicio, mientras que otras las proporciona directamente el Subnet Manager.

3.4.2. ELEMENTOS DE GESTIÓN

La infraestructura de gestión de la red se basa en tres tipos de elementos de gestión: agentes de servicio, gestores de clase (*class managers*) y clientes y aplicaciones de gestión (Fig. 3.14).

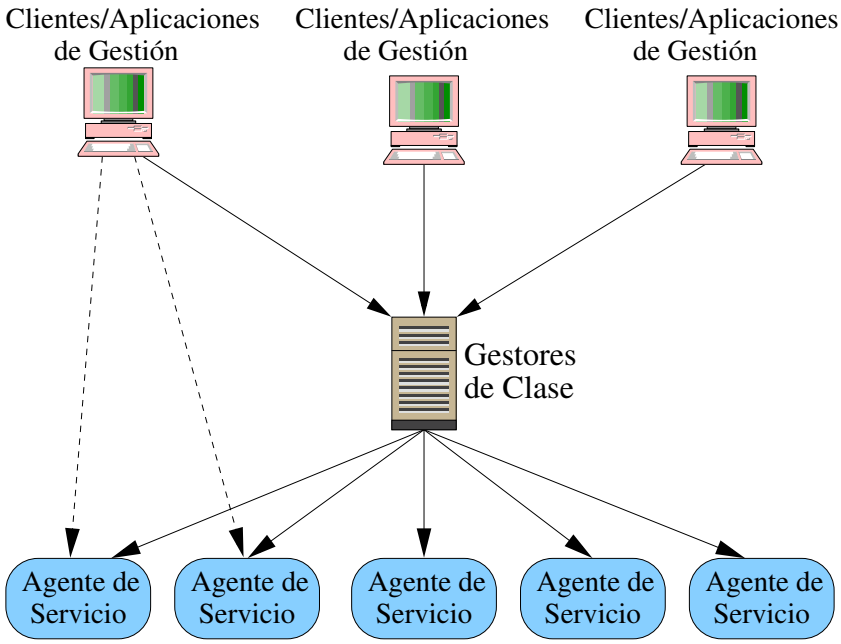


Figura 3.14: Elementos de gestión en InfiniBand.

Un agente de servicio es una entidad que reside en los nodos gestionados. Tiene una interfaz conocida para recibir y responder a las peticiones relacionadas con la gestión. La arquitectura de InfiniBand permite cualquier número de agentes de gestión en cada nodo. Lo usual es que en cada puerto haya un agente por cada clase de gestión. InfiniBand especifica cómo descubrir y configurar esos agentes de gestión.

Por otra parte, hay un gestor de clase en aquellos casos en que se necesite tener una entidad central de gestión que interactúe con los agentes de gestión. Estos gestores de clase son útiles por ejemplo para realizar una recolección eficiente de información de gestión distribuida entre los nodos, o para asegurar la atomicidad en el acceso a ciertos agentes de servicio.

Los clientes y aplicaciones de gestión suelen estar ubicados normalmente en hosts. Estos clientes y aplicaciones de gestión acceden a los agentes de servicio y a los gestores de clase para obtener información relativa a la gestión y solicitar funciones de gestión.

3.4.3. MENSAJES Y MÉTODOS DE GESTIÓN

La infraestructura para la gestión define la estructura y formato de los paquetes de gestión (*Management Datagram*, MAD). Estos paquetes de gestión son los que intercambian los elementos de gestión. Estos mensajes se envían usando la clase de servicio no fiable y sin conexión (UD). Todos los paquetes MAD deben tener exactamente 256 bytes, y consisten en una cabecera MAD y los datos correspondientes. El contenido del campo de datos variará en función de la clase concreta para la que se esté usando.

Las especificaciones también definen los métodos que usarán los elementos de gestión para intercambiar información. Estos procedimientos incluyen:

- Gets y Sets: Leen y modifican los atributos de los objetos gestionados.
- Traps: Permiten obtener información de ciertos eventos de forma asíncrona por parte de un agente de servicio.
- Subscription: Es la forma en que las aplicaciones piden recibir traps.
- Reports: Es la forma en que el gestor de una clase puede distribuir traps y otras informaciones a las aplicaciones que lo hubieran solicitado.

3.4.4. SUBNET MANAGER

El *Subnet Manager* (SM) se encarga de configurar y mantener la subred. El SM utiliza un formato especial de paquete llamado *Subnet Management Packet* (SMP) que tiene prioridad sobre el resto de paquetes. Un paquete SMP es un tipo especial de paquete MAD.

Algunas de las funciones del SM son descubrir la topología física de la subred, configurar los nodos y los conmutadores (LIDs, GIDs, Claves, etc.), calcular y distribuir a los conmutadores las tablas de encaminamiento en la subred, recibir peticiones de los agentes locales de gestión, etc.

Cada nodo contiene un Agente de Gestión de la Subred (*Subnet Management Agent*, SMA) que responde a los paquetes de gestión enviados por el SM. Cada puerto tiene un cola llamada QP0 dedicada a enviar y recibir paquetes de control. La interfaz que utiliza QP0 es el *Subnet Management Interface* (SMI) (Fig. 3.15).

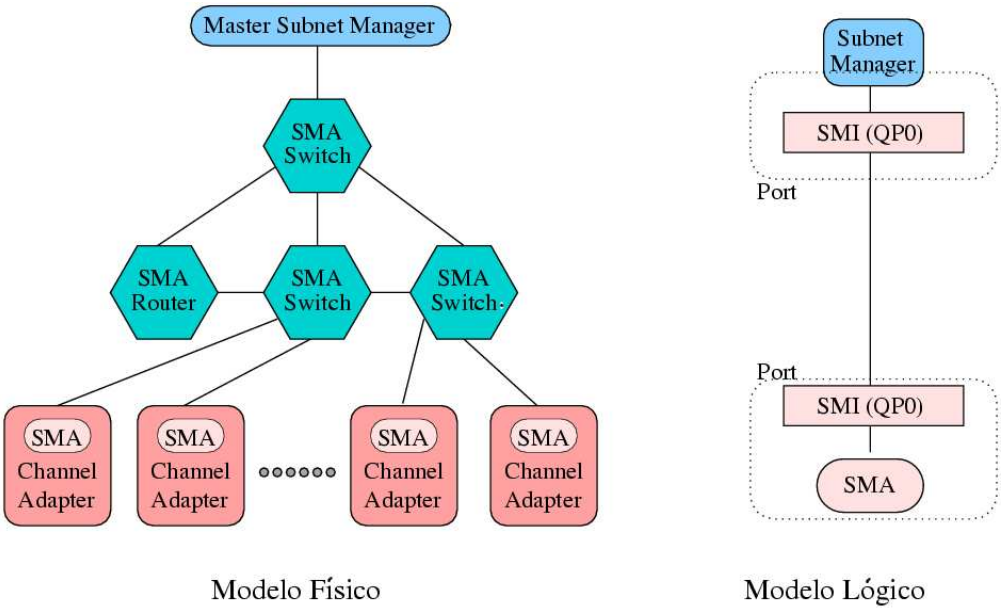


Figura 3.15: Modelo de gestión de la subred en InfiniBand.

Los paquetes de control sólo se envían desde y hacia un SMI, y no se pueden enviar otro tipo de paquetes a un SMI. Los paquetes de control nunca saldrán de la subred (los encaminadores no los reenvían) y siempre utilizan el canal virtual VL_{15} que está totalmente dedicado a este tipo de tráfico, siempre tiene prioridad sobre el resto de canales virtuales y no está sujeto a control de flujo.

Pueden existir varios SMs, pero sólo uno de ellos (llamado *Master*) puede estar activo en un momento dado. Si un SM descubre otro SM utiliza un protocolo específico para determinar cuál de ellos actuará como Master, quedando el resto a la espera por si el master dejara de funcionar, pasar a actuar como tal. Solamente el Subnet Manager Master es el autorizado para configurar los nodos y conmutadores.

El Subnet Manager Master se asegura el control exclusivo sobre los SMA al establecer una clave de control (M_Key) en cada puerto. Posteriormente, los paquetes de control que intercambie con ese puerto contendrán la clave que se les suministró. Sólo serán válidos los paquetes de control que contengan la clave correcta, ignorándose el resto de paquetes de control. La clave M_Key expira transcurrido un cierto período, tras el cual, el SMA vuelve a permitir establecer una nueva clave. De este modo, si el SM Master muere otro SM puede actuar como Master.

3.4.5. SUBNET ADMINISTRATOR

Sólo puede haber un *Subnet Administrator* (SA) en cada subred, y se supone que debe estar situado en el mismo nodo que el Subnet Manager Master. De hecho, los nodos localizan el SA mediante el envío de paquetes de la clase SubnAdm al nodo SM. Si el SA no estuviera en el mismo nodo que el SM, éste reenviaría ese paquete al nodo donde el SA residiera.

El administrador de la subred actúa como gestor de clase para la gestión de la subred. Este SA proporciona a los clientes y aplicaciones de control los medios para obtener y enviar información de control. Este SA es el encargado de interactuar con el Subnet Manager Master. El SA no utiliza los típicos mensajes de control, sino que utiliza un formato genérico de paquete.

La clase de administración de subred (SubnAdm) define los métodos para interactuar con el SA y obtener las características de la subred. Por ejemplo, el SA permite que un nodo encuentre otros nodos, averigüe el camino hasta ellos y determine las características de ese camino, para así poder establecer una conexión con ellos.

Otro servicio que proporciona el SA a los nodos es la posibilidad de que éstos le soliciten recibir notificación sobre las traps. Cuando el SM recibe una trap, el SA lo reenvía a todos los nodos que tenga registrados.

3.4.6. SERVICIOS GENERALES

Las especificaciones de InfiniBand describen una infraestructura de servicio general para las tareas de control llamadas no críticas. Como tareas no críticas se entiende todo lo necesario para el control de la subred excepto la configuración y su mantenimiento posterior. Todas las clases de servicios (excepto la SM) usan el modelo de servicio general (Fig. 3.16).

El modelo de servicios generales se basa en intercambiar información entre:

- Un gestor de clase y clientes/aplicaciones de control.
- Un gestor de clase y un agente de servicio general (*Global Service Agent*, GSA).
- Directamente entre un cliente/aplicación de control y un GSA.

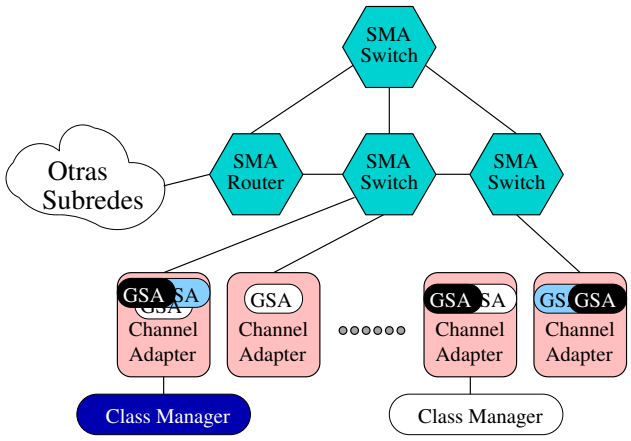


Figura 3.16: Modelo de servicios generales para la gestión en InfiniBand.

Lo normal es que para cada clase haya varios clientes o aplicaciones de control, un gestor de la clase, y muchos GSAs (al menos uno por puerto). El GSA es la entidad de un puerto que habilita esa clase de gestión para ese puerto. Un GSA responde a peticiones relativas a su clase del gestor de la clase o de aplicaciones de control. Un ejemplo podría ser el agente de gestión de dispositivo que tienen todos los nodos de E/S para responder al gestor de recursos de E/S de cada host.

Las especificaciones de InfiniBand definen una interfaz para los servicios generales llamada *General Service Interface* (GSI), que viene representado por un QP reservado (el QP1). Cualquier petición, sea ésta de la clase que sea, se envía al QP1 del destino, y es el GSI quien se lo hace llegar al GSA correspondiente. El GSI permite redirigir la petición a otro QP y/o otro puerto.

Las aplicaciones de gestión, los gestores de clase y los GSAs pueden estar ubicados en cualquier nodo (Fig. 3.17).

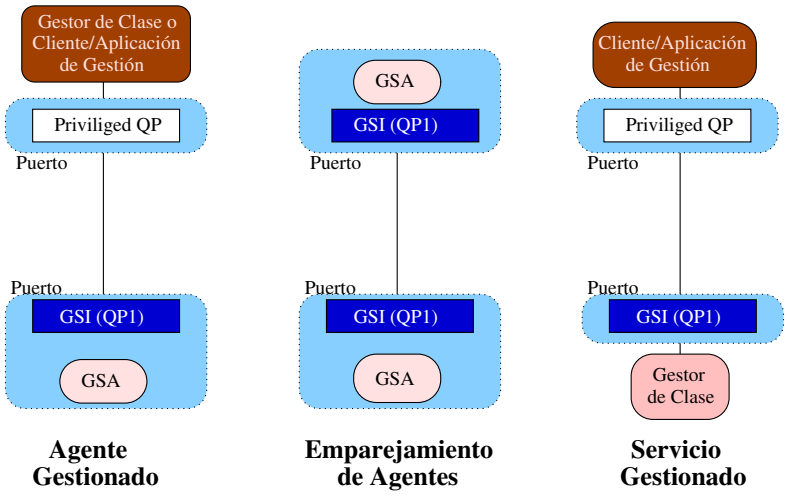


Figura 3.17: Modelo lógico de servicios generales en InfiniBand.

Como ejemplo de agente gestionado se podría citar un gestor de recursos de E/S en un host que accede al agente de gestión de dispositivo en una unidad de E/S. Como ejemplo de emparejamiento de agentes se podría citar el intercambio entre gestores de comunicación cuando se establece una conexión entre dos adaptadores. Por último, un ejemplo de un servicio gestionado podría ser el SA como un gestor de la clase de administración de la subred, que responde a los hosts sobre información de gestión.

3.4.7. COMUNICATION MANAGEMENT

La gestión de comunicaciones especifica los protocolos y mecanismos para establecer, mantener y liberar conexiones. También se encarga de la resolución de identificadores, que permite a los usuarios del servicio no fiable y sin conexión identificar el QP que soporta un servicio en particular.

En cada adaptador hay un *Communication Manager* (CM) accesible por cualquiera de sus puertos por medio del GSI de dicho puerto. Un CM interactúa con otros CMs por medio de los paquetes de la clase CommMgt. Las especificaciones de InfiniBand definen el comportamiento que deben tener los CM que interactúan. Sin embargo, la interfaz entre el QP cliente y el CM dependerá del sistema operativo y queda fuera de las especificaciones de InfiniBand.

3.5. CALIDAD DE SERVICIO

InfiniBand proporciona varios mecanismos que permiten al administrador de la subred gestionar distintas garantías de calidad de servicio, tanto para servicios con conexión como sin conexión. En los servicios con conexión se puede intentar garantizar calidad de servicio. Sin embargo, en los servicios sin conexión no es posible dar garantías, pues no se conoce a priori los requisitos que necesitan las aplicaciones. Para este tipo de servicio se puede configurar la subred con algún tipo de mecanismo similar a lo que hacen los servicios diferenciados.

Los mecanismos que proporciona InfiniBand para conseguir proporcionar calidad de servicio son básicamente: niveles de servicio, canales virtuales, arbitraje en dichos canales virtuales y particiones.

3.5.1. NIVELES DE SERVICIO

InfiniBand define el atributo nivel de servicio que deben incorporar todos los paquetes. Se permiten 16 niveles de servicio distintos, aunque el propósito de cada uno de ellos no está especificado. De esta forma, dependerá de las implementaciones o del administrador de la subred cómo distribuir los tipos de tráficos existentes entre los niveles de servicio disponibles.

Los niveles de servicio permiten segregar los distintos tipos de tráfico existentes de forma que se pueda señalar cada tipo de tráfico distinto, para luego ser capaz de proporcionar a cada tipo de tráfico un trato distinto, en función de sus necesidades.

3.5.2. CORRESPONDENCIA SL A VL

Tal y como se indicó en la Sección 3.2.3 los puertos en InfiniBand tienen canales virtuales. Un canal virtual representa un conjunto de buffers para transmisión y recepción en un puerto.

Cada puerto puede tener un número de canales virtuales diferentes (Fig. 3.18). Durante la fase de configuración ambos extremos del enlace deben ponerse de acuerdo para funcionar con el número mínimo de enlaces de los dos.

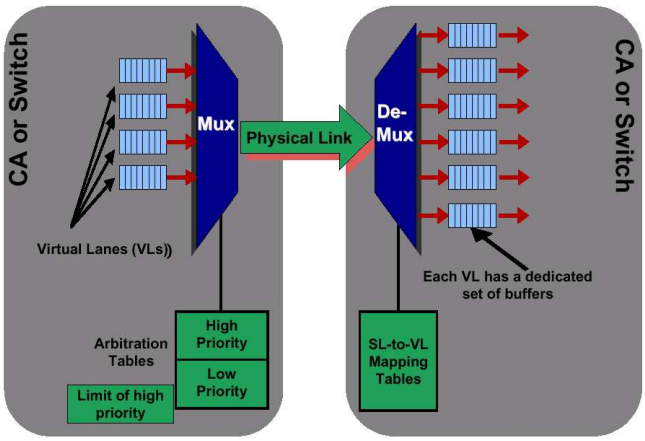


Figura 3.18: Funcionamiento de los canales virtuales en un canal físico.

Por otra parte, cada paquete contiene un Service Level (SL) que determina qué VL se usará en cada uno de los enlaces por donde pasa. De esta forma, es posible que puedan funcionar en la misma subred dispositivos que implementan distinto número de canales virtuales.

Cada puerto de salida (en conmutadores, encaminadores y nodos terminales) tiene una tabla de correspondencia entre los SLs y los VLs (*SLtoVLMappingTable*) configurada por las entidades de gestión de la red. Una correcta configuración de esta tabla hará que cada tipo de tráfico, previamente segregado en distintos niveles de servicio, use canales virtuales diferentes.

3.5.3. ARBITRAJE DE LOS PUERTOS DE SALIDA

Los puertos de salida (tanto en los conmutadores, como en los encaminadores y en los nodos terminales) incorporan una tabla de arbitraje que permite especificar la prioridad que tendrá cada canal virtual a la hora de enviar paquetes a través de ese puerto de salida.

Es obligatorio implementar esta tabla si el puerto tiene más de dos canales virtuales. El arbitraje se realiza sólo entre los canales virtuales de datos, pues el canal virtual del tráfico de control siempre tiene preferencia.

Cada tabla de arbitraje (Fig. 3.19) está formada por dos tablas, una para gestionar el tráfico de los canales virtuales de alta prioridad y otra para los canales virtuales de baja prioridad. Sin embargo, InfiniBand no especifica qué es alta y baja prioridad, quedando este aspecto a criterio del administrador de la subred.

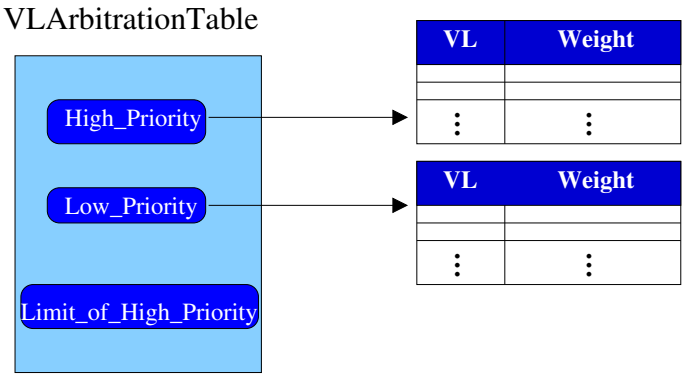


Figura 3.19: Estructura de la tabla de arbitraje.

Cada una de esas tablas tiene un funcionamiento cíclico y ponderado. Tanto la tabla de alta prioridad como la de baja prioridad tienen un máximo de 64 entradas. Cada entrada especifica un canal virtual y un peso. El peso indica la cantidad de unidades de 64 bytes a enviar por ese canal virtual. El peso debe estar entre 0 y 255, y siempre se redondea a un paquete completo.

El valor *LimitOfHighPriority* determina la cantidad máxima de información que

puede sacarse de los canales virtuales de alta prioridad, antes de pasar a enviar un paquete de un canal virtual de los considerados de baja prioridad. En concreto, los canales virtuales situados en la tabla de alta prioridad pueden enviar $LimitOfHighPriority \times 4096$ bytes antes de enviar un paquete de baja prioridad. Una vez enviado el paquete de un canal virtual de baja prioridad (si había alguno listo para enviarse), se vuelve a la tabla de alta prioridad y se continúa por la entrada donde se quedó y con el peso que le restaba.

Una característica importante del funcionamiento de la tabla de arbitraje es que si no hay paquetes de alta prioridad para ser transmitidos, se puede pasar a transmitir paquetes de baja prioridad. De esta forma no se desperdicia ancho de banda, y cuando no haya tráfico de alta prioridad se puede pasar a transmitir otro tipo de tráfico de menos prioridad aprovechando el ancho de banda disponible.

3.5.4. PARTICIONES

Tal y como se ha explicado en la Sección 3.2.3, InfiniBand permite establecer particiones en la subred. De esta manera se puede conseguir aislar determinadas zonas de la subred del tráfico destinado a otras zonas.

Para utilizar las particiones como mecanismo para mejorar la calidad de servicio, se pueden dedicar algunos SLs a ciertas particiones. Esto, junto con lo anteriormente expuesto, permite aislar el tráfico de las distintas particiones en distintos canales virtuales, y garantizar a cada partición una calidad de servicio distinta.

Capítulo 4

PRODUCTO INFINIBAND

En este capítulo se incluye una descripción del producto comercial diseñado de acuerdo con las especificaciones de InfiniBand [9] que ha sido utilizado para desarrollar este proyecto fin de carrera. La descripción cubrirá tanto aspectos hardware como aspectos software. En el primer caso, indicando los elementos que forman la configuración disponible, y en el segundo comentando los paquetes y aplicaciones instaladas. Señalar que el producto ha sido adquirido a **Mellanox Technologies Ltd.** [11] y está ubicado en el laboratorio del grupo de investigación RAAP en el I^3A.

4.1. HARDWARE

Cada elemento hardware se estudia individualmente para destacar su diseño y estructura, destacando sus características más importantes. No obstante, se puede recurrir a la documentación del fabricante [11] para encontrar una descripción más extensa y profunda de cada uno. En el último punto se puede ver como queda el sistema final con todos los elementos montados.

4.1.1. SERVIDOR

El servidor es la máquina que alberga el Subnet Manager y hace de servidor de ficheros para los nodos. Además, él mismo es un nodo de la subred. En definitiva será el punto de acceso a la subred por parte de los usuarios.

A continuación se exponen sus características hardware más destacables. De todos

modos no es un objetivo de este proyecto el conocimiento exhaustivo del servidor, por lo que si se quiere una descripción más completa se puede recurrir a la documentación de los respectivos fabricantes, como Intel [3], Kingston [2], Seagate [10], etc. Las características más destacables se enumeran a continuación:

1. Placa base ATX INTEL-SE7505VB2 ATA.
2. Dos procesadores Intel XEON 2,4 GHz.
3. Dos módulos DIMM DDR de 512 MB PC2100 KINGSTON.
4. Dos HDD SATA150 de 160GB Seagate ST3160023AS.
5. Controladora RAID Intel-SRC514L.

Todos los componentes serán montados en el servidor. Puede apreciarse su composición final con una fotografía (Fig. 4.1).

Figura 4.1: Foto del servidor una vez montados sus componentes.

4.1.2. TARJETA INFINIHOST

En este apartado se describe la tarjeta InfiniHost (Cougar[1]) que se puede instalar en el servidor. La tarjeta identificada como **MTPB23108-CE128** es el dispositivo de interconexión entre el servidor y el chasis (Sección 4.1.3). Para la especificación es un Target Channel Adapter (TCA). En la Figura 4.2 se muestra una imagen de la tarjeta.

[1]El nombre de Cougar es utilizado en la bibliografía para hacer referencia a la tarjeta InfiniHost de Mellanox. Ambos términos son utilizados indistintamente.

Figura 4.2: Imagen de la tarjeta InfiniHost (Cougar).

Las características más destacables de la tarjeta InfiniHost son las siguientes:

1. Se conecta al servidor a través de uno de sus conectores PCI-X.
2. Proporciona dos puertos InfiniBand a 4X.
3. Soporta hasta 2 GBytes de memoria DIMM SDRAM DDR.
4. Soporta hasta un millón de QPs, EEs y CQs.
5. Soporta multicast.
6. MTU programable de 256 Bytes hasta 2 KBytes.
7. Soporta ocho canales virtuales, de los cuales uno será utilizado para la gestión.
8. Soporta los mecanismos de transporte de InfiniBand: UC, UD, RC, RAW.

4.1.3. CHASIS

El chasis es el armazón donde los componentes de la red InfiniBand van a ser montados, excepto el servidor que por sus dimensiones resulta imposible instalar dentro del chasis, aunque no por ello deja de ser un nodo más dentro de la subred InfiniBand. En la Figura 4.3 se muestra una imagen del chasis.

Su nombre oficial es **MTEK4X14-BC**. Mantiene una subred Infiniband 4X Server Blade (ISB) completa. Proporciona tolerancia ante fallos y alta disponibilidad. Sus características principales se resumen en:

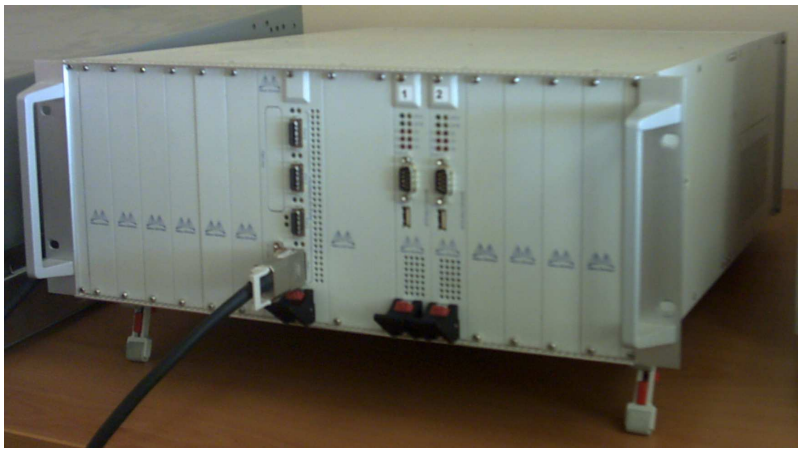


Figura 4.3: Fotografía del chasis.

- Topología de estrella dual en *backplane* [1] (Fig. 4.4)
- Soporte para doce Server Blades o slots de E/S.
- Soporte para dos Server Switches.
- Sistema de alimentación integrado.
- Sistema de refrigeración integrado.

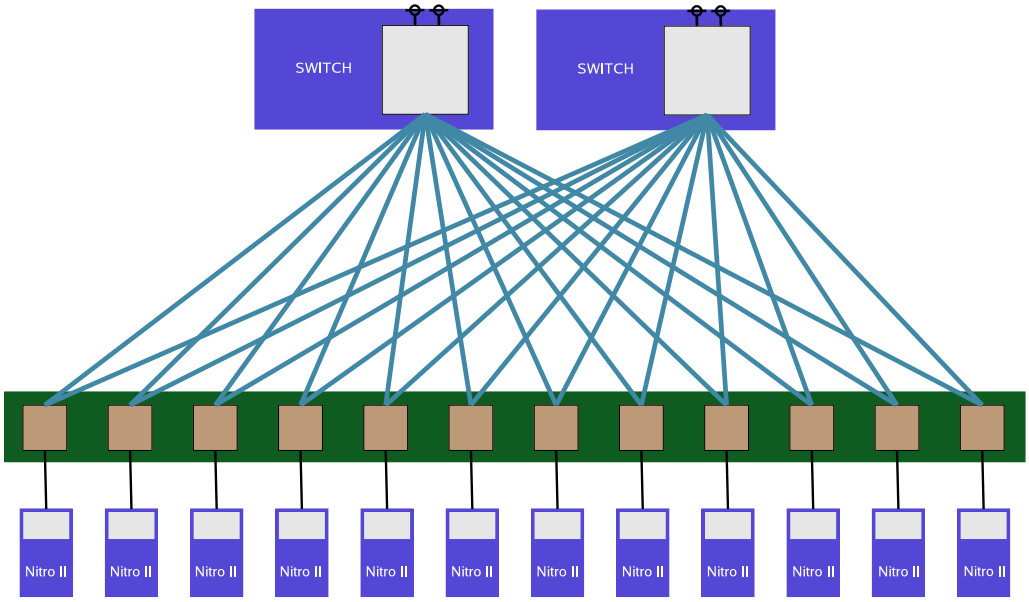


Figura 4.4: Topología en estrella dual.

Todas las tarjetas que pueden ser instaladas en el chasis deben tener unas dimensiones de 144,5mm x 280mm. La facilidad que posee el chasis para modificar la posición

[1] Se llama backplane a la placa que contiene toda la circuitería que conecta a los blades en la parte trasera del chasis.

de los nodos dentro de los dieciséis slots permite establecer diversas configuraciones, con lo que se permite balancear la carga.

El sistema de alimentación está integrado en el chasis. Es modular y suministra una corriente de 12V en el blackplane. El sistema de ventilación está integrado en el chasis. El aire entra por debajo del chasis y sube a través de los blades para terminar siendo expulsado por los extractores de la parte trasera del chasis.

4.1.4. SWITCH BLADE

El *Switch Blade* se identifica con el nombre **MTEVB43132-MSX4-4X**. Es el dispositivo de interconexión entre el chasis y el servidor y con otros dispositivos exteriores. También tiene forma de tarjeta que se integra dentro del chasis. En la Figura 4.5 se muestra una imagen del Switch Blade.

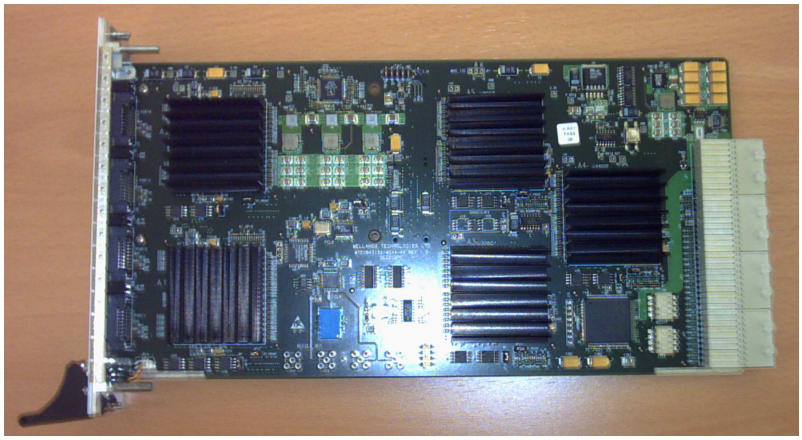


Figura 4.5: Imagen de un Switch Blade.

La parte frontal de un Switch Blade (Fig. 4.6) posee cuatro puertos InfiniBand 4X para conectar el Switch Blade con otros elementos válidos en una subred InfiniBand.

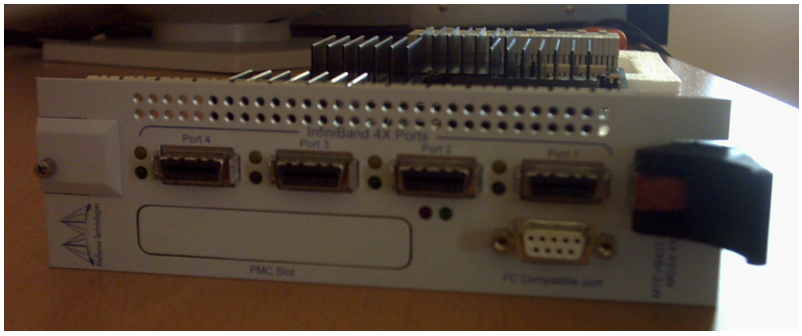


Figura 4.6: Imagen de la parte frontal de un Switch Blade.

Lógicamente, un Switch Blade, también conocido por **Nitrex II**, es un switch InfiniBand de 16 puertos (12 para nodos en el blackplane y 4 al frontal exterior), pero física y realmente es un conjunto de cinco switches InfiniBand. Cada switch interno es del tipo **InfiniScale**, posee ocho puertos físicos y un chip MT43132. Los switches InfiniScale están organizados en dos etapas para formar una estructura 4+12 en estrella dual (Fig. 4.4).

Figura 4.7: Cable InfiniBand Standar 4X.

A los 4 puertos externos del frontal del Switch Blade Nitrex II (Fig. 4.6) se conectan cables InfiniBand 4X (Fig. 4.7). Los cinco switches InfiniScale están organizados en una estructura de dos etapas usando conexiones 4X. Cada nodo tiene dos conexiones 4X con el Switch Blade. Esto dobla el ancho de banda y proporciona un punto de protección ante posibles fallos. Los cuatro puertos externos proporcionan un alto nivel de escalabilidad. Por ejemplo, se puede montar una red con 5 chasis, que sumarán unos 60 Server Blades, sin necesidad de ningún dispositivo auxiliar.

4.1.5. SERVER BLADE

Un *Server Blade* representa un nodo de procesamiento dentro de la subred InfiniBand. No son los únicos nodos en la subred, por ejemplo switches, servidores, dispositivos de E/S y de almacenamiento también son nodos dentro de una subred InfiniBand.

La utilización de Server Blades proporciona grandes ventajas a bajo coste. Un blade es un dispositivo de cómputo dispuesto en una pequeña placa que aprovecha el espacio ocho veces más y requiere ochenta veces menos consumo eléctrico que un servidor convencional [16]. Junto a la emergente generación de software de gestión, a las múltiples posibilidades de configuración de los blades, a la consolidación, compartición y expansión de elementos de red y almacenamiento, se apunta que esta filosofía será la dominante en los próximos años.

Para la realización de este proyecto se dispone de dos Server Blades **MTEVB23108-MSB** de Mellanox, también conocidos como **Nitro II**. Un Nitro II es un tipo de Server Blade para InfiniBand 4X que soporta chips InfiniHost MT23108 y ServerWorks Grand Champion. Para la especificación es un Target Channel Adapter (TCA). El Server Blade (Fig. 4.8) opera como una máquina *diskless* completamente funcional.

Una máquina *diskless* es aquella en la que su sistema operativo se encuentra localizado en otro dispositivo exterior, a modo de ejemplo, un Nitro II es una máquina sin disco duro físico, pero que su sistema de ficheros se encuentra accesible a través de la red en el servidor. Gracias al protocolo NFS este proceso es transparente para el usuario.

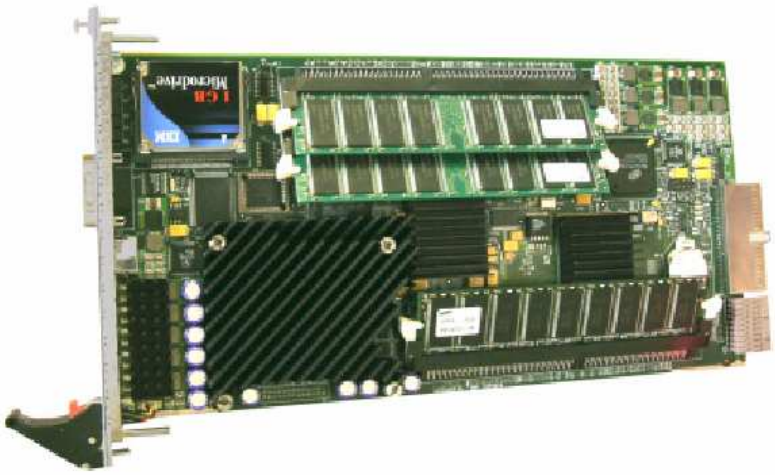


Figura 4.8: Imagen de un Server Blade.

A nivel lógico, un Nitro II presenta un diagrama de bloques como el de la Figura 4.9.

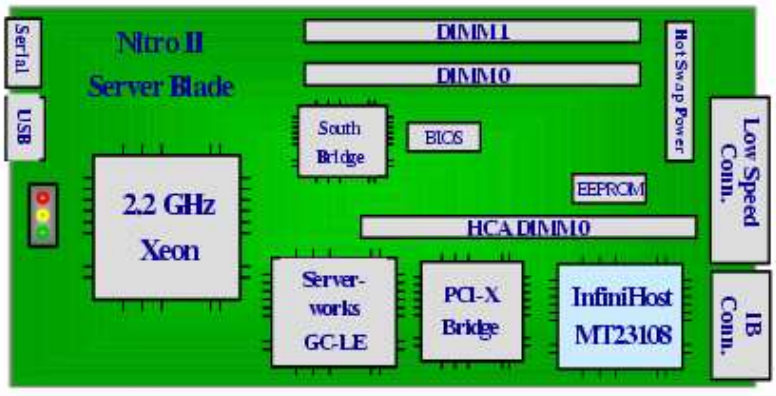


Figura 4.9: Diagrama de bloques de un Nitro II.

La interfaz blackplane consiste en dos conectores: uno para señales de baja velocidad y alimentación, y otra para señales InfiniBand de alta velocidad. El chip MT23108 y

el blackplane están unidos por dos enlaces InfiniBand 4X. La interfaz 133Mhz/64bit PCI-X conecta los chips MT23108 y ServerWorks GC-LE Northbridge que sirven de unión del nodo con la subred.

El socket uPGA478 soporta procesadores **Intel Pentium IV Xeon** de tecnología de 0.13μ y frecuencia configurable a través de la BIOS. La BIOS es de la marca Phoenix.

Cada Server Blade contiene una tarjeta de memoria flash del tipo CompactPCI. Este tipo de memoria *flash* se presenta como un dispositivo PCI tradicional pero en un nuevo *form factor*. Junto al soporte PICMG que posee el Server Blade, la tarjeta flash aparecerá como si fuera un dispositivo conectado al bus como Maestro Primario. Se puede consultar la web del grupo de desarrollo de la especificación CompactPCI y PICMG para más información [7].

Los dos sockets DIMM proporcionan hasta 4 GBytes de memoria RAM DDR PC2100 ECC. La memoria del HCA se cubre con un socket de 256 MBytes de memoria DIMM DDR. El MT23108 soporta hasta 4 DIMMs con 16 GBytes de memoria pero con sólo un DIMM, aunque en la práctica el límite actual de densidad de la memoria es 1 GByte.

Figura 4.10: Imagen frontal de un Nitro II.

Además proporciona un puerto USB y otro serie, ambos en el frontal de la tarjeta, para labores de depuración y monitorización (Fig. 4.10). Si se produce cualquier problema

en el arranque y el acceso al nodo no se configura por la red, estos puertos son la única forma de poder acceder al nodo.

4.1.6. EL SISTEMA MONTADO

Todos los elementos que se han analizado en los subapartados anteriores se han montado formando una configuración entre todas las posibles, puesto que según el slot donde los nodos Nitro II se pinchen, los caminos entre el resto de nodos se verá condicionado. Este sistema final conforma una subred InfiniBand. El sistema formado por todos los elementos hardware disponibles en este momento puede verse en la Figura 4.11. El sistema puede verse alterado si en cualquier momento se añadiesen nuevos nodos como consecuencia de una ampliación del número de elementos, pero esto no supone un problema porque el sistema es capaz de adaptarse a la nueva situación.

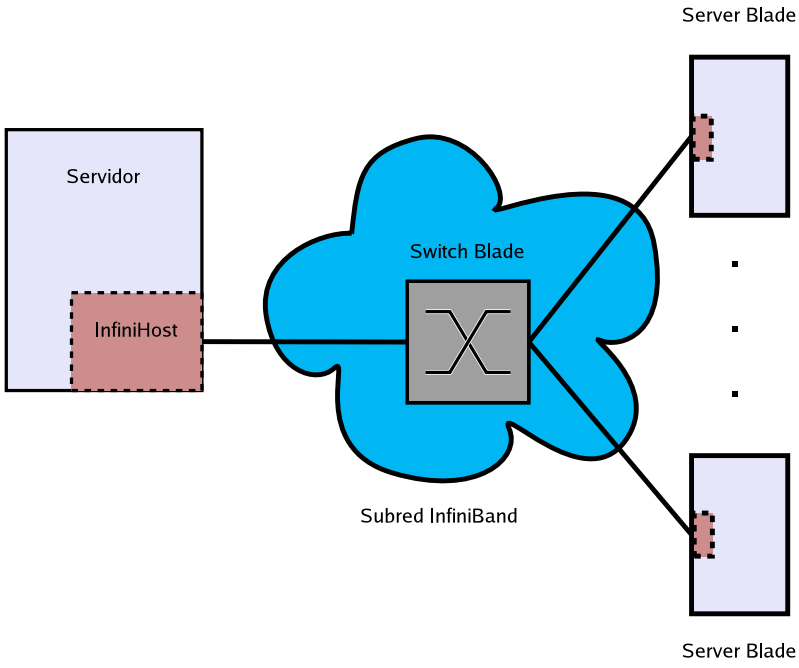


Figura 4.11: Sistema final con todos los elementos hardware.

4.2. SOFTWARE

Cada elemento software se estudia individualmente destacando sus características más importantes. La mayor parte de los trabajos futuros versarán sobre el software. Para evitar problemas se debe conocer qué paquetes se han de instalar para hacer uso del hardware de la mejor forma posible. El software debe ser el adecuado al hardware utilizado.

No obstante, se puede recurrir a la documentación del fabricante [11] para encontrar una descripción más extensa y profunda de cada uno.

4.2.1. MST

En esta sección se describe el paquete de herramientas MST (Mellanox Software Tools). Las herramientas que contiene el paquete son independientes unas de otras, aunque Mellanox las agrupa dentro de un paquete para mayor comodidad.

Se puede decir que mediante dichas herramientas se tendrá acceso tanto a los dispositivos de Mellanox como a los drivers de tales dispositivos. Para una descripción exhaustiva de todos sus componentes se pueden consultar los manuales y guías oficiales que proporciona Mellanox [15, 14]. Algunas de las herramientas principales que se pueden encontrar en este paquete son las siguientes:

InfiniBurn La funcionalidad de la tarjeta InfiniHost depende de la información de configuración y del *firmware* [1] que se carga en su proceso de arranque desde su pastilla de memoria flash/EEPROM. En el proceso de arranque la tarjeta hace 4 acciones:

1. Carga los valores de la configuración inicial de la memoria en los registros de configuración de la tarjeta InfiniHost.
2. Lee el firmware residente en la memoria flash/EEPROM.
3. Copia el firmware en la memoria DDR local de la tarjeta.
4. Ejecuta el firmware de la memoria DDR local de la tarjeta.

La tarjeta sale de fábrica con una versión del firmware por defecto (en nuestro caso 1.18) que debe ser actualizada antes de poner en uso la tarjeta. Precisamente, para actualizar el firmware y en sí toda la información de configuración del dispositivo, Mellanox proporciona a los usuarios esta herramienta. Con ella se puede tanto leer la información de configuración como escribirla.

InfiniVision Esta herramienta se proporciona para la tarjeta InfiniHost MT23208 [2]. Provee una visión compacta y conveniente de los módulos, registros y subcampos. Puede usarse para editar directamente los valores descritos en el manual de referencia del programador de la InfiniHost [12].

[1] Es un software que no puede modificarse por encontrarse en una memoria de sólo lectura o una mezcla entre el hardware y el software, es decir tiene parte física y una parte de programación consistente en programas internos implementados en memorias no volátiles, Un ejemplo típico de Firmware lo constituye la BIOS.

[2] Mellanox dispone de tarjetas InfiniHost con diferentes chips y de aquí la distinción.

iTrace Esta herramienta sirve para monitorizar las trazas generadas por los procesadores iRISC de la tarjeta InfiniHost. Los mensajes con las trazas informan a los desarrolladores de drivers sobre el estado interno, eventos, errores críticos, etc, de cada procesador. Los mensajes se almacenan en la memoria DDR de la tarjeta. iTrace extrae los mensajes y los muestra por la consola.

FLINT Esta herramienta sirve para actualizar la memoria flash desde una imagen en binario, la cual se prepara antes con la herramienta InfiniBurn. También se utiliza para realizar varias operaciones (lectura/escritura de palabras, borrado de un sector, verificación el estado de la flash/EEPROM) sobre la memoria flash/EEPROM.

emt Esta herramienta sirve para actualizar el firmware y la configuración de arranque que reside en la memoria flash/EEPROM de un dispositivo InfiniScale, pero que en cualquier momento puede interesar actualizar, en cuyo caso esta herramienta y las siguientes resultarán de un gran interés. También se puede preparar una imagen para ser utilizada en la línea de comandos con la herramienta *eburn*.

eburn Esta herramienta sirve para actualizar la memoria flash/EEPROM de un dispositivo InfiniScale en la línea de comandos.

adevmon Esta herramienta se utiliza para la monitorización de los dispositivos InfiniScale.

4.2.2. SDK

En esta sección se describe el paquete de herramientas SDK (Software Development Kit). Las herramientas que contiene el paquete son independientes unas de otras, aunque Mellanox las agrupa dentro de un paquete para mayor comodidad.

Se puede decir que mediante dichas herramientas se dispondrá de las librerías y aplicaciones necesarias para configurar la subred. Para una descripción exhaustiva de todos sus componentes se puede consultar los manuales y guías oficiales que proporciona Mellanox [13]. Algunas de las herramientas más importantes que se pueden encontrar dentro de este paquete son las siguientes:

Drivers Los drivers de la tarjeta InfiniHost del servidor para la plataforma Linux x86, que es la que se está utilizando en este proyecto fin de carrera.

Script VAPI Este script se utiliza para poner en marcha la tarjeta en el servidor y controlar el estado de la misma. Antes de realizar cualquier operación en la subred

es necesario utilizar este script para, por ejemplo, configurar la tarjeta InfiniHost haciendo que el kernel cargue en memoria todos los módulos necesarios para configurar el estado de los puertos de la tarjeta.

MiniSM Un Subnet Manager "mini" para poner en marcha la subred. Cumple con todas las tareas de un Subnet Manager, pero sólo toma decisiones por compromiso asignando valores triviales. Ignora todo lo que no es estrictamente necesario para poner la subred en marcha. Mellanox no lo mantiene en desarrollo por lo que para los trabajos futuros es necesario buscarle una alternativa como puede verse en los capítulos posteriores.

Driver IBNice Este driver es específico de los productos Mellanox y crea una interfaz Ethernet virtual sobre la tarjeta InfiniHost. Con ella se puede utilizar cualquier aplicación TCP/IP sobre la subred InfiniBand ya que el driver enmascara la infraestructura de InfiniBand por completo.

Diferentes tests Se pueden encontrar varias utilizadades para testear la subred y sus protocolos:

- El test RC es utilizado entre dos nodos para comprobar el transporte fiable RC entre un par de nodos de la red.
- El test de eventos se utiliza para observar los eventos de sincronización que arrojan los nodos mostrándolos en pantalla.
- Los tests de rendimiento implementados sobre los *IB-verbs* de Mellanox (VAPI) utilizados para medir ancho de banda y latencia en el intercambio de una cierta cantidad de información sobre distintos protocolos (UD y RC), usando operaciones de RDMA (WRITE y READ).

4.2.3. OSM-MVAPI

El paquete de herramientas OSM-MVAPI ha sido construido por Mellanox. Contiene una versión de OpenSM (Capítulo 6) adaptada para funcionar sobre los *IB-verbs* de Mellanox (VAPI).

OpenSM fue iniciado por Intel como parte de "InfiniBand Linux Driver"(IBAL) [22] aproximadamente hace dos años en Source Forge. Mellanox, junto a otros fabricantes de productos InfiniBand, se han unido a Intel para desarrollar OpenSM [18].

OpenSM se apoya en el stack IBAL de Intel para el acceso al hardware, esta es la capa sobre la que funcionan las aplicaciones, como por ejemplo los Subnet Managers.

Este acceso se realiza desde OpenSM a través de *complib*. *Complib* no es más que un conjunto de librerías sobre las que se apoya OpenSM. A su vez, el software de Mellanox se apoya sobre VAPI, a diferencia de Intel que usa IBAL. Cada fabricante usará el suyo propio. A modo informativo, la tercera semana de mayo de 2004 se ha anunciado el inicio de la integración del proyecto de Source Forge en el proyecto OpenIB. Este hecho será muy beneficioso para todos porquea partir de él todas las ventajas del proyecto de Source Forge pasarán al proyecto de OpenIB.

Al existir ciertas incompatibilidades en complib, Mellanox introdujo OpenSM sobre VAPI (llamándolo MVAPI). La primera versión oficial (release 0.2.0) puede ser descargada desde [18] aunque actualmente se está trabajando en la versión 0.3.0 de la misma.

La versión 0.2.0 contiene tres utilidades básicas, que son un subconjunto de las que se pueden encontrar en el proyecto original de Intel [22]:

opensm Es el Subnet Manager y Administrator que se puede encontrar en el proyecto de Intel. Como se puede ver en los capítulos posteriores esta herramienta resulta de gran interés para la mayoría de los investigadores ya que mediante ella se controla la configuración de la subred. Va a ver el sustituto de la herramienta *minism* de Mellanox, ya que posee mayores funcionalidades y su desarrollo está orientado a la filosofía *Open Source* la cual es perfecta para trabajar en grupo.

opensh Es un shell Tcl que extiende el conjunto completo de comando de opensm y su modelo de datos. Es perfecta para leer y asignar datos en tiempo de ejecución. Se espera que sea utilizable para desarrollar versiones gráficas del OpenSM.

opentest Es una utilidad para testear el funcionamiento de opensm y opensh. Es capaz de realizar muchas de las peticiones al adminitrador y devolver información de utilidad.

En la Figura 4.12 se muestra como los paquetes anteriores se sitúan en la pila de software. El OpenSM se sitúa sobre las MST y SDK instaladas y que forman la capa VAPI de Mellanox, los cuales son equivalentes a IBAL para Intel o al que corresponda para otro fabricante. Todos ellos están sobre el sistema operativo. En el futuro se planea que todas estas capas de VAPI, IBAL, etc. formen parte del kernel de Linux, de esta forma se alcanzaría el soporte nativo para InfiniBand en el kernel, lo que incrementaría las posibilidades de éxito de esta tecnología.

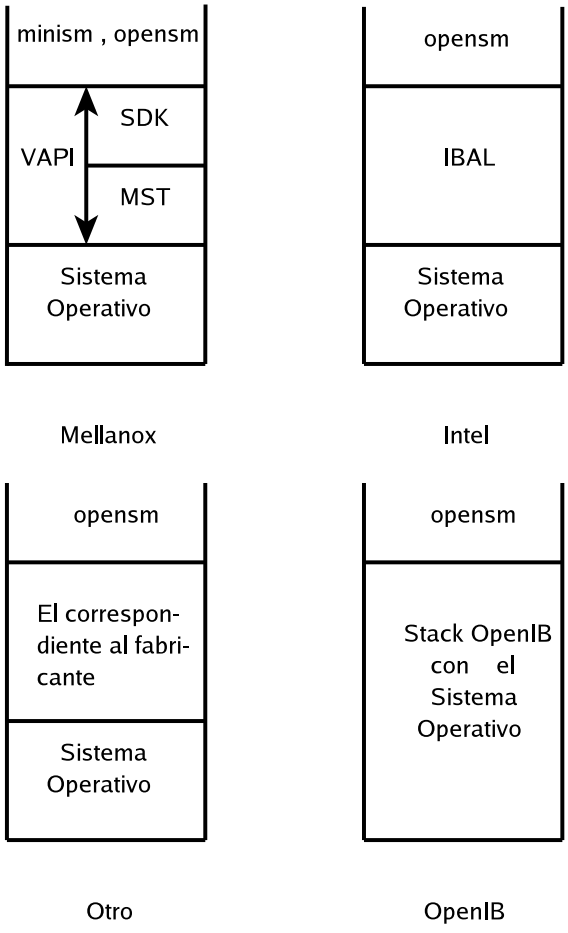


Figura 4.12: Capa de software sobre la que se sitúa el OpenSM.

Capítulo 5

PUESTA EN MARCHA

En este capítulo se indican los pasos más importantes que hay que seguir para poner en marcha el hardware y software descrito en el Capítulo 4. Del hardware se analiza cómo actualizar su memoria flash. En lo que se refiere al software, se describe toda la parte de instalación del sistema operativo y paquetes de software específicos para los productos de Mellanox. Para aprovechar todas las ventajas de una red de computadores se explica cómo actualizar los sistemas operativos de los nodos de procesamiento Nitro II que se pueden encontrar en la subred. Para terminar, se presentan algunos resultados obtenidos para probar el funcionamiento correcto de la subred y del sistema en su conjunto.

5.1. PRERREQUISITOS

Para poder disfrutar de todas las ventajas que proporciona el hardware se hace necesario realizar unas tareas de preparación del mismo.

La tarjeta InfiniHost ejecuta un software específico, *firmware*, que se almacena en una memoria EEPROM que posee la propia tarjeta y que se carga en la etapa de arranque de la tarjeta. Mellanox graba el *firmware* en la memoria EEPROM cuando fabrica la tarjeta, pero este software cambia con el paso del tiempo por lo que se hace necesaria su actualización. En el caso de que no se lleve a cabo la actualización de este software la tarjeta no detectará el Switch Blade por lo que la subred no podrá ser accedida.

Para la actualización del software se debe utilizar la herramienta *InfiniBurn* con la última imagen disponible en la web de Mellanox [11].

5.2. SISTEMAS OPERATIVOS

En esta sección se describirá el sistema operativo a instalar en la máquina servidor (Sección 4.1.1) y en los nodos de procesamiento Nitro II. Se analizarán todas las opciones, el sistema operativo en concreto que se ha elegido, las características que debe soportar dicho sistema operativo y las razones que han influido en la toma de esta decisión.

El sistema operativo es la piedra base sobre la que se trabaja en cualquier máquina. Las características del sistema operativo influirán en la forma de trabajar y hacer las cosas. Este proyecto no tiene como objetivo analizar lo que es un sistema operativo y cómo se estructura, por lo que no se profundizará más en ello.

En este proyecto se trabaja con elementos ensamblados por el fabricante Mellanox Technologies Ltd. [11], por tanto, el rango de elección quedará acotado a lo que el fabricante disponga. Entre todas ellas, se escogerá la que mejor satisface las necesidades concretas de este proyecto, aunque problamente en el futuro podrá variar para adaptarse a las necesidades de los usuarios.

5.2.1. EN EL SERVIDOR

A continuación se exponen las razones que han llevado a la elección del sistema operativo para la máquina servidor que se dispone.

Plataforma

La subred Infiniband estará formada por nodos pertenecientes a la plataforma Intel i686, Pentium IV Xeon en particular.

Sistema operativo

Una vez escogida la plataforma se buscarán los sistemas operativos que corren sobre dicha plataforma y tienen soporte de Mellanox. Los dos principales sistemas operativos son GNU/Linux y Windows 2000.

- Windows 2000 tiene el problema de ser un sistema propietario cerrado. Todos los problemas derivados de este hecho son bien conocidos. No obstante, Mellanox da

soporte para esta plataforma.

- GNU/Linux no tiene el problema de ser un sistema propietario. En el ambiente universitario y de investigación se ha introducido con gran aceptación. Se están realizando grandes esfuerzos para llevar a Linux el soporte para Infiniband: [22] y [18].

Distribución

Mellanox aconseja el uso de GNU/Linux y en particular de distribuciones de Linux RedHat o SuSe para las máquinas que actúan como servidor. Además, todos los ficheros fuentes que Mellanox dispone para sus clientes están escritos en lenguaje C. El código está escrito a bajo nivel, como por ejemplo, drivers, librerías, etc. por lo que la versión del compilador de C (gcc) es vital. Primará la elección de una distribución testeada por Mellanox para evitar futuros problemas a la hora de compilar e instalar los componentes software. Algunas de las distribuciones que a priori Mellanox aconseja son las siguientes:

- Red Hat 7.2, kernel 2.4.7.10 (UP y SMP[1]) con gcc 2.96-98
- Red Hat 7.3, kernel 2.4.18-10 (UP, SMP y bigmem) con gcc 2.96
- Red Hat 7.3, kernel 2.4.18-19.7.x (UP, SMP y bigmem) con gcc 2.96
- Red Hat Advanced Server 2.1, kernel 2.4.9-e.12 (UP, SMP y enterprise) con gcc 2.96
- Red Hat 8.0, kernel 2.4.18-14 (UP, SMP y bigmem) con gcc 3.2
- SuSe 8.0, kernel 2.4.18-64GB-SMP con gcc 2.95.3
- SuSe SLES 8.0, kernel 2.4.19-64GB-SMP con gcc 3.2

Elección

En base a lo indicado anteriormente se ha elegido la plataforma Intel i686 y el sistema operativo GNU/Linux con Red Hat 7.3. Actualmente no se configura el soporte SMP por problemas de incompatibilidad con la tarjeta Controladora RAID de los discos duros. La versión del compilador de C (gcc) es *2.96* que viene instalada por defecto en

[1] UP significa “soporte para un procesador” y SMP significa “multiprocesamiento simétrico”.

dicha distribución. Sin embargo, la versión del kernel no es una versión oficialmente soportada por Mellanox, por lo que se tendrá que migrar a la versión *2.4.18-14* que sí lo está.

5.2.2. EN LOS NITRO II

A continuación se exponen las razones que han llevado a la elección del sistema para los Nitro II.

Las máquinas Nitro II son un caso particular. Un Nitro II trae de fábrica un sistema operativo totalmente funcional. Así pues, la elección ya ha sido tomada por Mellanox: plataforma Intel i686 y sistema GNU/Linux Red Hat 7.2.

No obstante, en determinadas ocasiones resulta deseable la actualización del sistema operativo, en particular del núcleo del sistema operativo en los Nitro II, caso que será estudiado en la Sección 5.3.

5.3. SISTEMA OPERATIVO DE UN NITRO II

En esta sección se describe el sistema operativo que Mellanox proporciona para que sea instalado en los nodos Nitro II, se analiza el proceso de arranque de un Nitro II y se explica los pasos necesarios para construir un sistema operativo propio capaz de ser ejecutado por un Nitro II. Esta tarea resulta de gran interés para los trabajos posteriores que necesiten modificarlo, puesto que Mellanox no proporciona el fichero *.config* del kernel, utilizado para generar la imagen del kernel que utilizan los Nitro II por defecto.

5.3.1. SISTEMA OPERATIVO DE MELLANOX

Mellanox proporciona a sus clientes un sistema operativo que puede ser instalado en sus Server Blade Nitro II. En concreto, se trata de un Linux Red Hat 7.2 con un kernel 2.4.7-10 y gcc 2.96. El sistema operativo propiamente dicho se distribuye en dos partes. Una primera parte se compone del kernel de Linux que junto a una imagen *initrd* se guardan en una pequeña memoria flash dentro del Nitro II. La misión de ambas es la de configurar la subred InfiniBand para poder acceder al servidor de ficheros por NFS y montar el sistema de ficheros del Nitro II. La segunda parte es el sistema de ficheros que

se encuentra en el servidor de ficheros.

5.3.2. PROCESO DE ARRANQUE

El proceso de arranque en un nodo Nitro II es del tipo arranque sin disco o *diskless*. Este tipo de arranque está destinado a las máquinas que no tienen un disco local donde se encuentra el sistema operativo y es muy común en sistemas formados por Server Blades.

Un Nitro II dispone de una memoria flash que simula un dispositivo IDE. En concreto, el dispositivo es un disco primario maestro (*/dev/hda* en Linux) gracias al soporte PICMG. Esta memoria contiene el kernel y una imagen *initrd*. El kernel montará la raíz del sistema de ficheros desde un servidor de ficheros a través de NFS. Antes de esto debe procesar una imagen *initrd* para que configure el acceso a la subred. El sistema de ficheros una vez que está montado es transparente al usuario.

5.3.3. CONSTRUCCIÓN DE UN SISTEMA OPERATIVO NUEVO

Mellanox proporciona un sistema operativo funcional. Todo esto no descarta la posibilidad de construir otro sistema alternativo que pueda ser instalado en los nodos Nitro II. Para poder construir un sistema operativo nuevo será necesario definir, compilar e instalar tanto el kernel del sistema como una imagen *initrd* que sea capaz de configurar el acceso a la red InfiniBand. También es necesario construir el sistema de ficheros que dará soporta al kernel. Este soporte consistirá en proporcionar los ficheros de datos, binarios y scripts vitales para hacer al sistema utilizable para los usuarios.

A continuación, se exponen las principales etapas que deben realizarse para construir un sistema operativo nuevo para los nodos Nitro II. Todos los pasos siguientes se han realizado en la máquina servidor.

El kernel

Una vez que se disponga de los ficheros fuente y de cabecera apropiados para la versión 2.4.18-14 del kernel de Linux que se desea construir, el proceso de construcción no difiere del proceso estándar para construir un kernel Linux. Sin embargo, hay que destacar que para la construcción posterior de la imagen *initrd* es necesario que el kernel que está en ejecución disponga de:

- Soporte para sistema de ficheros EXT3.
- Soporte para drivers de tarjetas Ethernet.
- Soporte de modversion.
- Soporte de loopback.

Si el kernel actual posee el soporte anterior se puede pasar a la definición del kernel destinado a los Nitro II. Éste debe ser definido con las siguientes características:

- Soporte para el sistema de ficheros en red NFS (Network File System).
- Soporte para cliente NFS.
- Soporte para el sistema de ficheros raíz sobre NFS.
- Soporte para la consola en el puerto serie.
- Soporte para la autoconfiguración del kernel.

Concluida la construcción del kernel de los Nitro II se pasa a la construcción de su imagen *initrd*.

La imagen initrd

Una imagen initrd es un fichero comprimido en *zip* que contiene un sistema de ficheros Linux. El kernel, una vez que ha arrancado, crea un RAMDISK en memoria donde descomprime la imagen initrd y ejecuta un script llamado *linuxrc* que se encuentra en el directorio raíz de la imagen initrd y por tanto en la RAMDISK. Este script tendrá como función configurar el acceso a la red InfiniBand por medio de los scripts y drivers existentes en la imagen initrd y por tanto en la RAMDISK. Un RAMDISK es la forma que tiene el sistema operativo de reservar un espacio de memoria principal y simular en ella un sistema de ficheros.

La memoria flash puede ser accedida por medio del dispositivo */dev/hda* dentro de un nodo Nitro II. El kernel nuevo, junto a la imagen *initrd* nueva, deberán ser depositados en el directorio *boot* del sistema de ficheros en el memoria flash. Se configurará LiLo [1] en la flash para poder seleccionar el kernel nuevo cuando los Nitro II arranquen.

[1] LiLo son las iniciales de Linux Loader.

Cuando el nodo Nitro II arranque desde la memoria flash, se podrá seleccionar el kernel nuevo para que sea cargado. La red será configurada y a continuación el kernel intentará acceder a su sistema de ficheros que se encuentra en el servidor, con lo que se hace necesario construir el sistema de ficheros apropiado para el kernel nuevo.

El sistema de ficheros

Para la construcción de un sistema de ficheros para los Nitro II se puede partir del sistema de ficheros que proporciona Mellanox o de uno nuevo partiendo de cero. Para utilizar el que ya existe debe adaptarse al kernel nuevo, copiando las librerías del kernel nuevo en el directorio */lib/modules* que montará el Nitro II y a su vez adaptando el resto de ficheros de configuración, por ejemplo, */etc/modules.conf*. Cuando arranque por primera vez se deberá observar qué problemas se presentan para posteriormente actualizar los ficheros de configuración y de este modo eliminar los errores que se produzcan en el proceso de arranque del sistema operativo de los Nitro II.

El sistema de ficheros construido se encuentra en el servidor conectado a la red y se deberá exportar a través del protocolo NFS. El sistema de ficheros conformará el directorio raíz en el nodo Nitro II. Cuando el Nitro II concluye el arranque está preparado para que los usuarios accedan a él como si se tratara de un ordenador estándar.

5.4. SISTEMA DE FICHEROS

Esta sección muestra como se prepara el servidor para exportar el sistema de ficheros que un Nitro II tiene que montar por NFS. Cuando el kernel nuevo se ejecute en los Nitro II necesitará acceder a este sistema de ficheros, de ahí su importancia.

5.4.1. PRERREQUISITOS

Para llevar a cabo la instalación del software se necesitará tener los siguientes componentes instalados correctamente:

1. Un host o server que tenga instalada la tarjeta HCA en una ranura de expansión PCI-X.
2. El host tendrá instalados los paquetes software MST y SDK.

3. Tener configurado el servicio NFS en el servidor.

5.4.2. RAÍZ DEL SISTEMA

El servicio NFS va a exportar el sistema de ficheros que se encuentra en un directorio en el servidor. Los siguientes pasos están encaminados a conocer el contenido de este directorio y la configuración necesaria del servidor NFS en el servidor. Para preparar el sistema de ficheros raíz, se realizan los siguientes pasos:

1. Mellanox suministra con el chasis un CD con un sistema de ficheros completo. Estos ficheros son: *nitro2_bsp-081.tgz*, *nitro2-sb01-110.tgz*, *nitro2-basefs-110b-new.tgz*.

2. Se crea un directorio */sbfs* donde se instala la base del sistema de ficheros. Esto creará a su vez tres directorios: *user*, *home* y *opt*.

```
mkdir /sbfs
cd /sbfs
tar zxvf nitro2-basefs-110b-new.tgz
```

3. Se instala el sistema de ficheros del primer blade bajo */sbfs*. Esto creará a su vez el subdirectorio *sb01*.

```
tar zxvf nitro2-sb01-110.tgz
```

4. Se instala el sistema de ficheros del segundo blade.

```
mkdir tmp
cd tmp
tar zxvf <ruta>/nitro2-sb01-110.tgz
mv sb01 /sbfs/sb02
```

5. Se edita el fichero */sbfs/sb02/etc/sysconfig/network* y se asigna la variable *HOSTNAME*:

```
HOSTNAME=sb02
```

6. Los pasos anteriores se repetirían con cada uno de los blade que se tuvieran.

7. Se edita el fichero */etc/exports* y se añaden las siguientes entradas:

```
/sbfs/usr *(rw,no_root_squash)
/sbfs/opt *(rw,no_root_squash)
/sbfs/home *(rw,no_root_squash)
/sbfs/sb01 *(rw,no_root_squash)
/sbfs/sb02 *(rw,no_root_squash)
```

8. Se leventa el servicio *vapi*.

```
vapi start
```

9. En un terminal diferente debe lanzarse un Subnet Manager para que descubra la topología y configure los enlaces. Este Subnet Manager puede ser tanto *minism* como *opensm*. Por ejemplo, se puede invocar a *minism* para que lleve a cabo esta tarea.

```
bash: minism InfiniHost0
minism> x 5
```

10. En un terminal diferente se declara la variable *HOSTADDR* y se lanza el driver IBNICE. De esta forma la interfaz eth1 recibe una dirección IP que se enlaza con la red. La interfaz eth1 representa la tarjeta InfiniHost.

```
export HOSTADDR=10.5.8.1
IBsNICE.sh start
```

11. Se habilita el servicio de NFS para que los nodos Nitro II puedan montar su sistema de ficheros en el servidor.

```
/etc/init.d/nfs start
```

5.4.3. INSTALACIÓN DE MINICOM

Cada blade tiene un puerto serie. Una aplicación para interaccionar con este puerto es *minicom*. Una vez instalada puede lanzarse para configurarla. Se configura con los parámetros *9600* de velocidad, *8N1* de paridad y */dev/ttyS0* como dispositivo serie. Para que escuche del puerto basta con lanzarla con el parámetro -s.

5.5. INICIALIZACIÓN DEL SISTEMA

En esta sección se describen los pasos principales para poner en marcha todo el sistema. En particular se describen los ajustes en el hardware, principalmente chasis y nodos, para terminar de poner en marcha la red.

5.5.1. PRIMEROS PASOS

En las secciones anteriores se ha explicado cómo instalar los componentes software en el servidor. A continuación, en esta sección el objetivo será instalar los componentes hardware de la red, es decir: el chasis con switches y nodos. Los pasos principales a dar para conseguir que el sistema arranque por completo son los siguientes:

1. El chasis se colocará en una superficie firme. Es aconsejable dejar una distancia aproximada no menor de dos centímetros para facilitar la entrada de aire por debajo del chasis. La refrigeración es muy importante para salvaguardar la integridad física de cualquier aparato electrónico.

2. Cada uno de los nodos y los switches serán apagados mediante los botones frontales para evitar que la corriente entre directamente causando daños. Una vez apagados, se conectará a la red eléctrica el chasis y a continuación se enchufarán los switches y los nodos. El orden no es relevante.

3. Tanto en los switches como en los nodos el LED rojo se deberá ver encendido. Esto indicará que el elemento recibe corriente.

5.5.2. CONFIGURACIÓN DE LOS NODOS

Cada nodo viene con un disco Compact Flash preprogramado que contiene un kernel linux, en concreto RedHat 7.2. El objetivo es que el nodo arranque con este kernel preprogramado y mediante NFS monte el sistema de ficheros en la máquina servidor. Por defecto, el nodo mantiene la siguiente información para que el arranque se pueda ejercer:

- IP del server (ej. *10.5.8.1*)
- IP del propio nodo (ej. *10.5.8.11*)

- Directorio montado por NFS en el servidor (ej. */sbfs/sb01*)

Estos parámetros sólo sirven para el primer nodo, pero si se dispone de más se aconseja que se revisen estos parámetros y se modifiquen en caso necesario. Mellanox ya preconfiguró los nodos con una configuración apropiada para cada nodo dependiendo del total adquirido.

El proceso de reconfiguración en el caso que sea necesario estará compuesto por los siguientes pasos:

1. Conectar el cable serie entre el nodo y el servidor. Lanzar un programa para escuchar el puerto serie (ej. minicom).
2. Enchufar el nodo dentro del chasis. Se debe ver el LED L6 en rojo.
3. Tras unos segundos, el LiLo que se encuentra en la memoria flash del Nitro II se lanza y muestra su menú para que el usuario seleccione el kernel que desea que ejecute el Nitro II.
4. El kernel seleccionado se empieza a cargar. El usuario verá un *prompt* que le invita a entrar en modo interactivo "*Press Y within 3 seconds to change into interactive mode*". Presionar la tecla y.
5. Una vez que se termine de arrancar lanzamos el script *setup.sh* que guiará al usuarios a través del proceso de configuración. Los tres parámetros que se podrán modificar son:
   - Dirección IP del nodo.
   - Dirección IP del servidor NFS.
   - Punto de montaje del sistema de ficheros en el servidor.
6. Tras grabar los cambios, se reiniciá el nodo tecleando *exit*. El fabricante aconseja que se mantenga sin corriente eléctrica durante unos momentos.
7. Una vez reiniciado ya no es necesario entrar otra vez en modo interactivo. La consola del terminal serie tampoco es necesaria ya, salvo para monitorizar el proceso de inicio.
8. Cuando el nodo pase el *prompt* del modo interactivo, instalará automáticamente los drivers del HCA y levantará el enlace físico. En el frontal el LED LK-A GREEN se encenderá. Después de que el Subnet Manager descubra el nodo, le asigne un LID y levante el enlace, el LED LK-A YELLOW se encenderá.

9. Tras la configuración del Subnet Manager, el script de IBNICE se ejecuta y el usuario podrá ver la dirección asignada en la interfaz de red eth0 en el Nitro II.

10. A continuación el nodo intentará encontrar el sistema de ficheros NFS para montar su sistema. Esto será inmediato tras ver el mensaje de inicialización de Red Hat (o de la distribución que se haya usado salvando las distancias).

11. Si todo va bien, ya sólo bastará con entrar el *login* y *password* correctos. Por defecto es *root* y *nitro*.

5.5.3. ENTRADA EN LOS NODOS

Una vez que los nodos están funcionando correctamente se puede entrar en ellos mediante una sesión SSH. Desde el servidor, por ejemplo, se lanzará desde una consola el siguiente comando:

```
# ssh -l root 10.5.8.xx
```

Por defecto (*xx:sb01=11,sb02=12,...*) aunque ya se ha visto en el apartado anterior como se puede cambiar las direcciones IP de los nodos.

5.6. TESTS

Puesto que uno de los objetivos de este proyecto es la puesta en marcha de la subred, se debe comprobar que la subred funciona sometiéndola a algunas pruebas. No se pretende obtener las causas por las que se obtienen los resultados, ni el análisis de esos resultados, sino simplemente comprobar que los tests funcionan correctamente.

Las pruebas se han realizado sin tráfico de red y el resto de condiciones no se han tenido en cuenta. Por otro lado, se han utilizado los dos Server Blade que existen en la subred **ceuta** y **melilla** [1] para realizar las pruebas.

[1] ceuta y melilla son los nombres de los Nitro II

5.6.1. PING

El primer test que se realiza en una red para comprobar su funcionamiento es la utilización de la utilidad **ping**. Esta utilidad envía paquetes "ICMP request" y recibe la respuesta en paquetes "ICMP echo" midiendo el tiempo transcurrido entre el envío y la llegada.

ping -c 10 melilla

El tiempo obtenido es 68μs mínimo, 531μs máximo y 79μs de media. Este test es suficiente para asegurar que la red está funcionando.

5.6.2. SCP

Otra prueba que se ha realizado en la red es la transferencia de ficheros de un host hacia otro. En este caso se ha escogido el protocolo de transferencia de ficheros sobre SSH.

scp fichero melilla:.

El fichero tiene un tamaño de 417 MBytes y la transferencia ha tardado 44 segundos, así que la tasa de transferencia es aproximandamente de 9 MBytes/s. El ancho de banda teórico de cada enlace es 1 GByte/s en cada dirección por lo que el resultado no es óptimo.

Aquí hay que destacar que el fichero está montado por NFS en el servidor por lo que no es una transferencia de ficheros propiamente dicha, en la que un fichero se transmite por una red de una máquina a otra. Sirve para medir el comportamiento de la red junto a la característica de tener el sistema de ficheros accesible por NFS.

5.6.3. PERF_MAIN

La última prueba que se ha realizado en la subred es con una utilidad propia de los productos Mellanox y que viene incorporada en el paquete software SDK, *perf_main*.

La batería de prueba consiste en envíar 15000 paquetes de 1 KByte, otros 15000 paquetes de 2 KBytes, 4 KBytes, 8 KBytes y así hasta paquetes de 8192 KBytes, variando distintos parámetros: protocolo de transporte y tipo de RDMA.

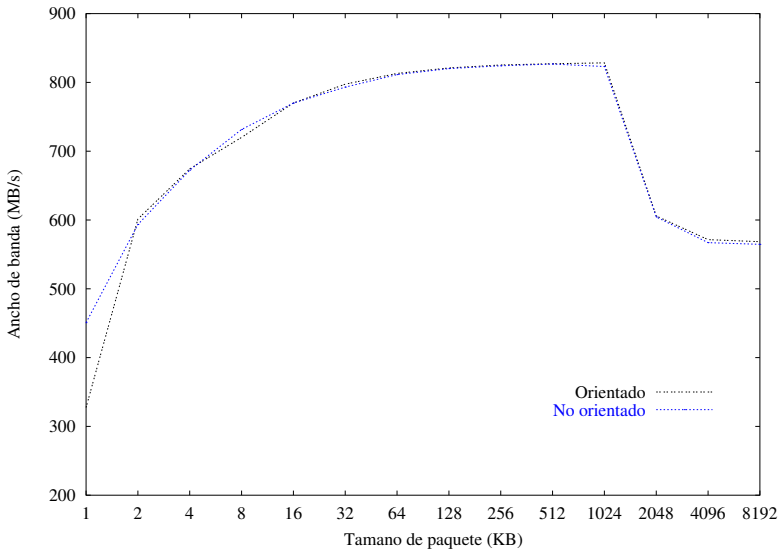


Figura 5.1: Orientado a conexión vs. no orientado.

En primer lugar, se configuran dos baterías, una con transporte orientado a conexión y otra con no orientado a conexión. El tipo de RDMA se configura como “Write” por escoger uno. En la Figura 5.1 se muestra el resultado.

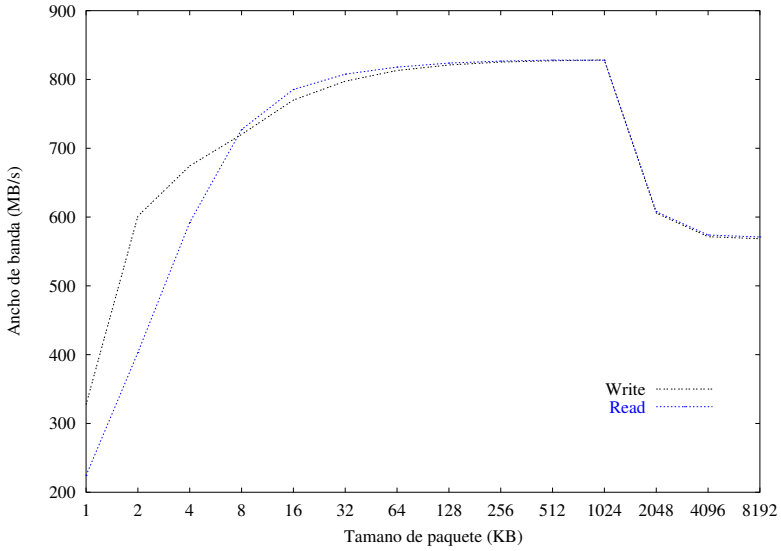


Figura 5.2: Tipo RDMA Write vs. Read.

En segundo lugar, se configuran dos baterías, una el tipo de RDMA “Write” y otra con el “Read”. El protocolo de transporte se configura orientado a conexión por escoger uno. En la Figura 5.2 se muestra el resultado.

Mediante las dos baterías anteriores se puede observar como se puede alcanzar un ancho de banda de 850 MBytes/s, el cual se aproxima al ancho de banda teórico de los enlaces, 1 GByte/s.

Capítulo 6

OPEN SUBNET MANAGER

En este capítulo se analiza el gestor de subred OpenSM. Se comienza con un repaso a la estructura interna del mismo para terminar por las tareas propias de un gestor de subred o subnet manager. El análisis está enfocado a conocer cómo se ha implementado en OpenSM las funcionalidades que indica la especificación.

6.1. INTRODUCCIÓN

Hace aproximadamente dos años que Intel comenzó un proyecto software para dar soporte en Linux a sus productos InfiniBand [4]. El proyecto echó a andar en SourceForge [22] bajo una combinación de licencias copyright y GPL.

Con el paso del tiempo y la aparición de más y más fabricantes de productos InfiniBand este proyecto se fue convirtiendo en un referente importante en la comunidad InfiniBand. Muchos de estos fabricantes empezaron a aportar su granito de arena al proyecto (programadores, diseñadores, drivers, etc) cuando vieron que podían aprovechar todo lo que estaba ya hecho.

Dado que el proyecto inicial estaba orientado a productos de Intel, esta empresa y el resto de fabricantes se han decidido por crear otro proyecto, llamado OpenIB [18] y que se presenta como el referente de InfiniBand en Linux. Es por esto que muchos módulos presentan incompatibilidades ante productos de otros fabricantes.

Durante cierto tiempo los dos proyectos han caminado en paralelo pero a principios de mayo del 2004 se ha iniciado la convergencia del proyecto inicial hacia OpenIB.

Paralelamente Mellanox, como participante en ambos proyectos, ha liberado el subnet manager del proyecto OpenIB, llamado OpenSM. Principalmente ha realizado una depuración de incompatibilidades en las librerías que dan apoyo a OpenSM para acceder a las librerías específicas de Mellanox que a su vez acceden al hardware. De este modo ya es posible ejecutar OpenSM sobre productos del propio Mellanox utilizando la capa *VAPI*.

OpenSM está escrito en lenguaje C sin objetos a pesar de que las estructuras y las funciones se agrupan a semejanza de lo que sería un objeto con sus métodos.

A continuación se analizan los aspectos más importantes que presenta la versión actual de OpenSM 0.3.0. Para este proyecto se ha utilizado la especificación de InfiniBand 1.0.a [9], aunque la versión 1.1 es compatible.

6.2. MÓDULOS PRINCIPALES

OpenSM está estructurado en una serie de módulos. Todos los módulos comparten una estructura interna pero recogen una funcionalidad distinta. A continuación se presenta una breve descripción de cada módulo.

LOG El módulo Log encapsula la información necesaria para que OpenSM lleve una traza de todo lo que realiza. Estas trazas resultan de gran ayuda para los desarrolladores porque muestran en detalle todas las decisiones tomadas.

SUBNET El módulo Subnet encapsula la información necesaria para que OpenSM pueda gestionar la subred. La información recogida es sobre todos los nodos de la subred, sus puertos, valores de configuración, tablas de encaminamiento, etc.

VENDOR El módulo Vendor encapsula la información necesaria para que OpenSM pueda acceder al hardware de un modo independiende del hardware específico. En tiempo de compilación será elegido el módulo a usar. Hay módulos para *Mellanox* [11], *TopSpin* [1], *Intel* [3], etc.

MAD_POOL El módulo MAD Pool encapsula la información necesaria para que OpenSM pueda gestionar una *pool* (será como un almacén) de objetos MAD. Toda la comunicación se realiza mediante MADs y este módulo es fundamental para el seguimiento de dichas MADs.

VL15 El módulo VL15 encapsula la información necesaria para que OpenSM pueda instanciar la interfaz VL15. OpenSM asigna un objeto VL15 por subred IBA. El

objeto VL15 transmite MADs al cable a un ritmo acelerado, pero sin sobrecargar los buffers de los componentes de la subred. Los módulos podrán enviar MADs a la interfaz VL15 tan rápido como sea posible.

SM El módulo SM (Subnet Manager) encapsula la información necesaria para que OpenSM instancie un gestor de la subred.

Los **CONTROLLERS** (Controladores) son los encargados de la gestión de la información de gestión recibida en el SM desde los direfentes agentes (SMA) situados en los nodos.

Los **MANAGERS** (Gestores) son los encargados de la gestión de los distintos componentes del SM.

SA El módulo SA (Subnet Administrator) encapsula la información necesaria para que OpenSM instancie un administrador de la subred. Es el encargado de responder a las peticiones que desde el resto de nodos se mandan al gestor de la subred.

Los **RECORDS** son los encargados de responder a las peticiones recibidas en el gestor.

6.3. HILOS DE TRABAJO

En esta sección se explicarán los hilos de trabajo que componen OpenSM. Se trata de dar una breve descripción de cada uno, puesto que son la parte activa de la aplicación.

El número de hilos de trabajo que componen OpenSM no es fijo. Pueden añadirse más hilos para realizar una tarea específica en un determinado momento. No obstante se pueden identificar un número de ellos que permanecerán estables dentro de OpenSM, entre los cuales podemos encontrar los siguientes:

- Hilo principal o *main*.
- Hilo de barrido o *sweeper*.
- Hilo de interfaz vl15 o *poller*.
- Hilo de transporte o *umadt*.

6.3.1. MAIN

El hilo principal o main es el hilo que inicia el sistema operativo cuando el usuario teclea “opensm” en un shell. El hilo tiene su inicio en el fichero *main.c*. A continuación, se indican las tareas que realiza:

1. Declarar variables auxiliares de configuración y del objeto *opensm*.
2. Esperar las órdenes del usuario en la entrada estándar.
3. Inicializar el objeto *opensm*, función *osm_opensm_init*.
4. Asignar el nivel de log, función *osm_opensm_set_log_flags*.
5. Si el usuario no indicó un puerto donde asignar el *opensm*, entonces le pedirá al usuario que escoga uno entre todos los posibles.
6. Iniciar el barrido de la subred, función *osm_opensm_sweep*.
7. Terminar.

Hay dos funciones que se deben destacar sobre las otras: *osm_opensm_init* y *osm_opensm_sweep*.

osm_opensm_init

Esta función inicia el objeto *opensm* para su uso. Las tareas que realiza se pueden resumir en las siguientes:

1. Construir el objeto opensm, *osm_opensm_construct*.
2. Iniciar el log, *osm_log_init*.
3. Bloquear el objeto para el acceso exclusivo.
4. Iniciar el objeto subnet para esta subred, *osm_subn_init*.
5. Iniciar el adaptador específico del vendedor, *osm_vendor_new*.
6. Iniciar la pool de MADs, *osm_mad_pool_init*.
7. Iniciar la interfaz VL15, *osm_vl15_init*.

8. Iniciar el subnet manager, *osm_sm_init*.

9. Iniciar el subnet administrator, *osm_sa_init*.

10. Crear los grupos de multicast, *osm_opensm_create_mcgroups*.

osm_opensm_sweep

Esta función inicia el barrido sobre la subred. El primer barrido de la subred se refiere a la configuración de la subred. La función está redirigida a otra función, *osm_sm_sweep*, que es la encargada de mandar la señal apropiada al gestor del *opensm* (Sección 6.4.1) para iniciar el barrido de la subred, señal **SWEEP**. Véase la Sección 6.4 para más información sobre la configuración de la subred.

6.3.2. SWEEPER

Este hilo se sitúa en el objeto subnet manager. Tras construir e iniciar los objetos que componen el propio objeto subnet manager (sm) se crea el hilo en cuestión. A dicho hilo se le indica que ejecute la función *_osm_sm_sweeper*. El algoritmo que sigue dicha función es el siguiente:

```
/* Si el hilo fue iniciado puede empezar a ejecutarse. */
SI  (estado del hilo es INIT) ENTONCES
 Establecer una máscara con la señal SIGINT activa;
 Asignar el estado del hilo a RUN;
FINSI

MIENTRAS (estado del hilo es RUN) HACER
 SI (estado del SM es MASTER) ENTONCES
   Enviar la señal SWEEP al gestor del SM;
 FINSI

 Esperar eventos según máscara y configurar un timeout;

 /*
  * Terminó de esperar porque ha ocurrido algo
  * y tiene que comprobar qué ha sido.
```

```
  */
 SI (hubo un evento) ENTONCES
   Registrar el evento en el log;
 SI_NO
   SI (No ocurrió el timeout) ENTONCES
     Registrar el fallo en el log;
 FINSI
FINMIENTRAS
```

6.3.3. POLLER

Este hilo se sitúa en el objeto que representa a la interfaz VL15. El constructor del objeto VL15 crea el hilo en cuestión. A dicho hilo se le indica que ejecute la función *__osm_vl15_poller*. El algoritmo que sigue dicha función es el siguiente:

```
/* Si el hilo fue iniciado puede empezar a ejecutarse. */
SI (estado del hilo es INIT) ENTONCES
  Establecer una máscara con la señal SIGINT activa;
  Asignar el estado del hilo a RUN;
FINSI

MIENTRAS (estado del hilo es RUN) HACER
  Adquirir exclusión mutua sobre el objeto vl15;

  /*
   * Se comienza sirviendo MAD wrappers y pasándolos
   * a la interfaz de transporte.
   * La cola ufifo (Unicast) tiene prioridad porque
   * hay alguien que espera su respuesta en un período
   * de tiempo determinado.
   * Los elementos de la cola rfifo (Response) esperan
   * una respuesta no marcada por el tiempo.
   */
  SI (hay MAD wrappers en la cola ufifo) ENTONCES
    p_fifo = Identificador de la cola ufifo;
  SI_NO
    p_fifo = Identificador de la cola rfifo;
  FINSI
```

```
  /*
   * Extrae el elemento de la cabeza de la cola seleccionada.
   * Este elemento en un envoltorio con la MAD e información
   * adicional sobre dicha MAD.
   */
  p_madw = cabeza_cola(p_fifo);

  Liberar exclusión mutua sobre el objeto vl15;

  /* Enviar la MAD y esperar. */
  SI (la cola p_fifo no estaba vacía) ENTONCES
    /*
     * Hay que evitar que se vea la respuesta a una MAD
     * antes que el resultado de enviarla y por eso no
     * se espera a que llegue la respuesta para actualizar
     * los contadores. En caso de error se arreglarán
     * posteriormente dichos contadores.
     */
    Actualizar contadores de MADs enviadas;
    Enviar la MAD guardada en p_madw;

    SI (error al enviar) ENTONCES
      Reparar los contadores de MADs enviadas;
    FINSI
  SI_NO
    /*VL15 vacía, no hay nada que hacer*/
    Esperar hasta la próxima petición de envío sin timeout;
  FINSI

  MIENTRAS (estado del hilo es RUN y
    MADs en el cable >= Máximo_permitido) HACER
    Esperar hasta la próxima petición de envío sin timeout;
  FINMIENTRAS
FINMIENTRAS
```

6.3.4. UMADT

Este hilo se sitúa en la capa que representa a la interfaz del componente de transporte UMADT. Dicho componente contiene grandes dependencias con el hardware real. El soporte específico se determina en tiempo de compilación.

El hilo es creado cuando se obtiene un manejador de la capa UMADT, *osm_vendor_bind*. El hilo ejecuta el código de la función *__mad_recv_processor*, que tiene el cometido de procesar las MADs. Su carácter dependiente de las capas específicas *Vendor* hace que el estudio del algoritmo que implementa escape de los objetivos de este proyecto.

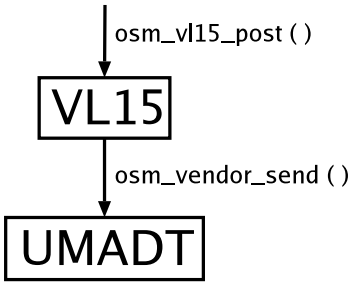


Figura 6.1: Relación entre VL15 y UMADT.

La Figura 6.1 muestra la relación existente entre el módulo VL15 y el componente UMADT. Al módulo VL15 se entra por medio de la llamada *osm_vl15_post* cuando se quiere enviar una MAD, por ejemplo, esto se produce cuando el OpenSM quiere mandar una petición Get() o Set(). Esta llamada a su vez accede al componente UMADT a través de la llamada *osm_vendor_send*.

6.4. CONFIGURACIÓN DE LA SUBRED

En el apartado 14.1 de la especificación se dice que el gestor de la subred (SM) es el elemento clave en la inicialización y configuración en una subred InfiniBand. En una subred pueden coexistir múltiples subnet manager pero sólo uno puede ejercer de *Subnet Manager Master*. Suyas son las responsabilidades de:

- Descubrir la topología física de la subred.
- Asignar identificadores locales (LIDs) a los nodos finales, switches y routers.
- Establecer posibles caminos entre los nodos finales.

- Barrer la subred para la búsqueda de cambios en la topología. Unos ejemplos de posibles cambios en la subred pueden ser cuando se enciende un nodo en la subred o si alguno se apaga.

La comunicación entre el SM Master [1] y los SMAs [2] se realiza mediante paquetes especiales de gestión llamados SMP. Existen dos tipos de paquetes SMP:

LID routed Estos paquetes son enviados a través de la subred por los switches basándose en el LID destino.

Directed routed Estos paquetes son enviados a través de la subred por los switches basándose en un vector de números de puerto que define un camino a través de la subred. Se usan para implementar varias funciones de gestión, en particular, antes de que los LIDs sean asignados a los nodos.

SMP Directed Routed

OpenSM usa paquetes SMP *directed routed* para llevar a cabo sus tareas de gestión en la subred. En el apartado 14.2.2 de la especificación se describe los algoritmos que deben seguir todos los nodos en la subred y las respectivas interfaces de gestión (SMI) para construir y manejar paquetes SMP de consulta y respuesta.

Es un mecanismo para enviar paquetes de gestión a través de una subred configurada, sin configurar o parcialmente configurada. Son usados para descubrir nodos en la subred, realizar diagnósticos o verificar la conectividad de enlaces.

Existen dos componentes que soportan este mecanismo en la subred:

LID Permissive Cuando un nodo, inclusive switches, recibe un paquete con esta dirección lo envía a su SMI. En el SMI el paquete es procesado, asegurando así que el vector de números de puerto se va rellenando cuando el SMP atraviesa los nodos en su camino.

Directed routing Permite la definición de una ruta explícita, basada en números de puerto de switches, para un paquete que atraviese la subred.

Una ruta en la subred tiene tres partes:

[1] Si no se indica lo contrario en los sucesivo se hablará del SM Master
[2] Los agentes existentes en cada nodo y que atienden las peticiones del SM

- Desde el nodo fuente al switch fuente. Esta parte usa *LID routing*, el nodo y el switch fuente están identificados por sus LIDs. Pueden existir switches intermedios pero de existir la porción de subred estará configurada para permitir *LID routing*.

- Desde el switch fuente al switch destino. Esta parte usa *direct routing*. La ruta está especificada por el número del puerto que el paquete debe usar para dejar el switch. Esta porción de la subred no necesita estar configurada para soportar *LID routing*.

- Desde el switch destino al nodo destino. Esta parte usa *LID routing*, ambos se identifican con su LID. Pueden existir switches intermedios pero la subred estará configurada para permitir *LID routing*.

Cada parte puede estar vacía, lo que da 8 combinaciones de las cuales sólo 4 son significativas. Un estudio más exhaustivo de estas combinaciones se puede encontrar en el apartado 14.2.2 de la especificación.

6.4.1. MÁQUINAS DE ESTADOS

OpenSM y el Subnet Manager (SM), contenido en OpenSM, son gestionados por medio de máquinas de estados. Dichas máquinas de estados están implementadas por medio de variables que guardan el estado en el que se encuentra la máquina y transitan entre estados a la llamada de funciones que envían mensajes a la máquina. En base al estado de la máquina y la señal recibida, se realizan ciertas acciones y en el caso correspondiente se cambia de estado.

La máquina de estado que controla al SM propiamente dicho, es la máquina de estados SMInfo (Fig. 6.2) descrita en el apartado 14.4.1 de la especificación. Tiene su ubicación dentro del código en el fichero *osm_sm_state_mgr.c*.

La máquina de estados que controla al OpenSM propiamente dicho, es una máquina de estados desarrollada por los programadores de OpenSM. Su objetivo es la gestión interna de OpenSM y del SM. Tiene su ubicación dentro del código en el fichero *osm_state_mgr.c*. Sirve de complemento a la máquina del SM, ayudando en cada estado a realizar distintas tareas y a su vez siendo ayudada por la máquina del SM.

Las tareas más importantes que controlan las máquinas de estados coinciden con las tareas de las que es responsable un Subnet Manager. En la Figura 6.3 los conjuntos de estados tienen su correspondencia con las tareas siguientes, que en las secciones

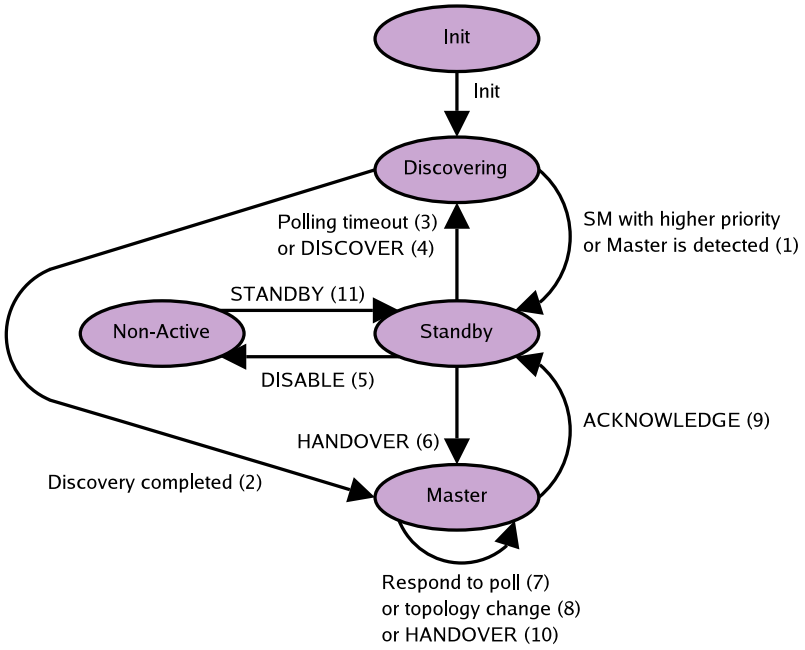


Figura 6.2: Máquina de estados SMInfo.

posteriores se estudiarán en más profundidad:

1. SM parado a la espera de ponerse a funcionar. Por ejemplo, este es el estado del SM que no es Master.

2. Descubrimiento de la topología física.

3. SM en espera después de configurar la subred.

4. Asignación de LIDs a los nodos.

5. El SM tiene que pararse.

6. Se ha detectado otro SM Master.

7. Envío de la trap 64 a los puertos descubiertos.

8. Configuración del encaminamiento *linear*.

9. Configuración del encaminamiento *multicast*.

10. Configuración del estado de los puertos.

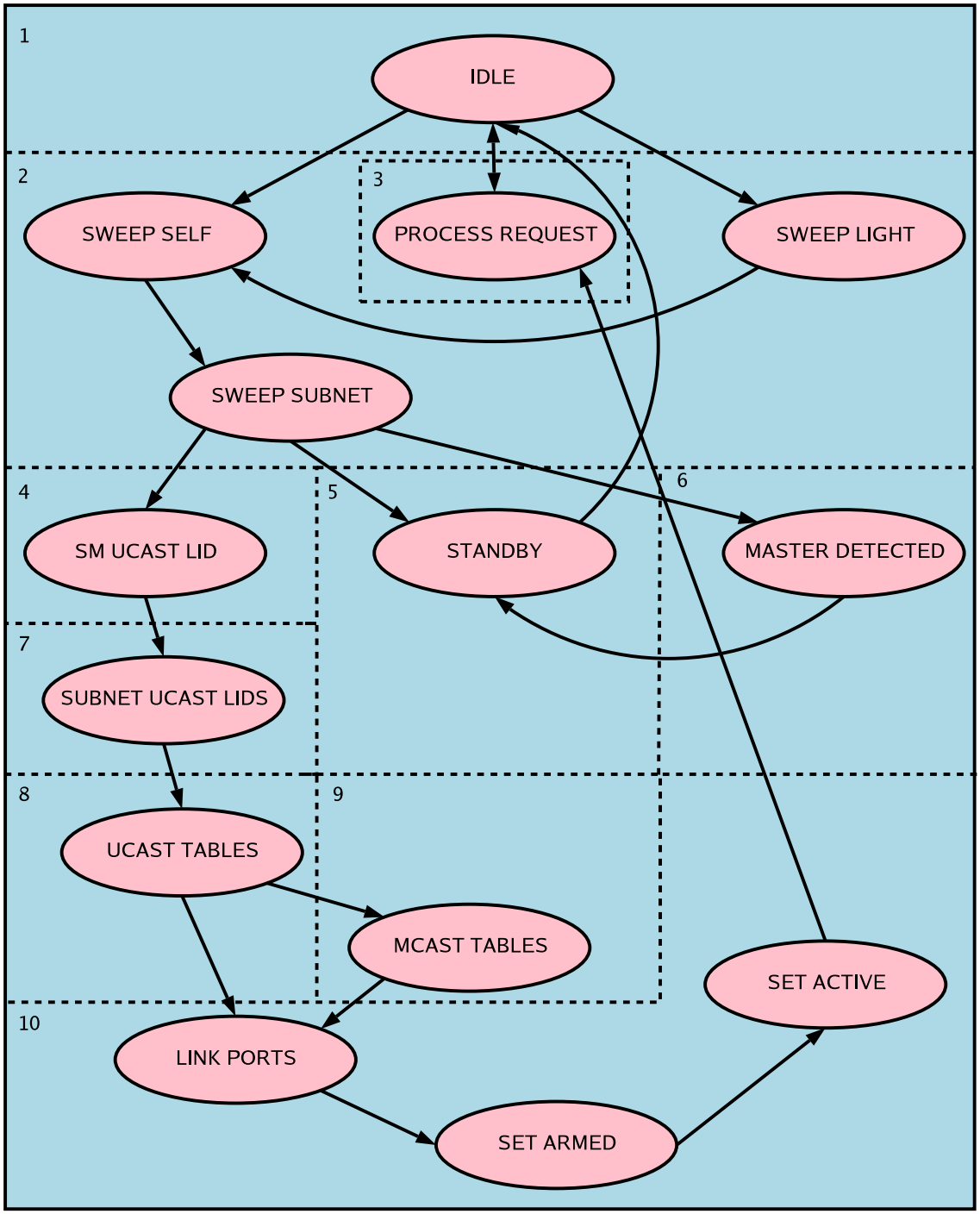


Figura 6.3: Máquina de estados del OpenSM.

6.4.2. DESCUBRIMIENTO DE LA TOPOLOGÍA FÍSICA

El SM utiliza paquetes SMP *directed routed* para encaminar los paquetes destinados a descubrir los nodos en la subred. El procedimiento a seguir es la exploración de nodos en función de los saltos. Atravesar un enlace es un salto. Se empieza con 0 saltos (nodo sobre el que se encuentra el SM) y se pasa a 1, 2, 3,..., N saltos. N es finito porque el número de nodos es finito.

Cuando el SM envía un SMP especifica el camino que debe seguir el SMP (Initial Path) y el número de saltos. Cuando un SMP llega con la respuesta al SM el camino de vuelta se encuentra en el paquete SMP (Return Path). Esto significa que el camino de ida y el de vuelta desde el SM a un nodo no tiene que ser el mismo en sentido inverso.

El SM lee y escribe información mediante métodos Get() y Set() sobre los atributos de los nodos. Estas operaciones y los atributos importantes en el proceso de descubrimiento de la topología se definen en la especificación y son los siguientes:

- El atributo *NodeInfo* provee información de gestión común a todos los CAs, switches y routers. Para descubrir un nodo el SM manda una petición Get() a dicho nodo para obtener el atributo *NodeInfo*. Importantes son los campos *NodeType* donde se define el tipo del nodo y *Numports* para el número de puertos físicos del nodo.

- El atributo *SwitchInfo* provee información de gestión específica de un switch.

- El atributo *PortInfo* provee información de gestión específica de un puerto.

El SM del OpenSM puede localizarse tanto en CAs como en switches, pero no en routers a pesar que la especificación si que recoge este caso como válido. El primer nodo que el SM debe descubrir es el nodo donde se encuentre él mismo. Para ello las peticiones se realizan a cero saltos y el camino de ida se define vacío.

Cuando se procesa los SMP de respuesta se realizan a su vez nuevas peticiones Get(). Esto significa que el OpenSM no lleva de un modo explícito la tarea del descubrimiento, simplemente él tira la "primera piedra" y espera una reacción en cadena que produzca el descubrimiento de toda la topología física.

Existen dos tipos de barridos. Un barrido "light" consiste en descubrir sólo el nodo donde se encuentra el OpenSM y un barrido "heavy" es el barrido que cubre toda la subred.

El proceso de descubrimiento se centra por tanto en el procesamiento de los SMPs con respuestas a peticiones Get() sobre los atributos *NodeInfo*, *SwitchInfo* y *PortInfo*. La reacción en cadena se inicia cuando se procesa el SMP de respuesta con un atributo *NodeInfo* enviado por el SMA del nodo donde se encuentra el OpenSM. Las comparaciones y las acciones tomadas por la lógica de OpenSM siguen el siguiente esquema:

- El atributo *NodeInfo* corresponde a un nodo no descubierto hasta ahora:

- Envía al gestor del OpenSM la señal CHANGE_DETECTED para notificar que algo ha cambiado en la subred.
- Añade el nodo nuevo a la base de datos del OpenSM.
- Inserta el nodo en la lista de nodos nuevos que necesitan recibir la *trap 64* para ponerse en servicio (apartado 14.2.5.1 de la especificación).
- Establece el enlace con el nodo vecino que ya fue descubierto. Si el nodo vecino no existe no hace nada, cuando el vecino sea descubierto establecerá el enlace entre ambos.
- Mira el tipo de nodo:
  - Si es un channel adapter entonces continúa su proceso como un channel adapter nuevo.
  - Si es un switch entonces continúa su proceso como un switch nuevo.
  - Si es un router entonces continúa su proceso como un router nuevo.

- El atributo *NodeInfo* corresponde a un nodo descubierto anteriormente:
  - Mira el tipo de nodo:
    - Si es un router no hace nada porque por ahora no hay soporte para este caso.
    - Si es un channel adapter entonces continúa su proceso como un channel adapter existente.
    - Si es un switch entonces continúa su proceso como un switch existente.
  - Establece el enlace con el nodo vecino que ya fue descubierto. Si el nodo vecino no existe no hace nada, cuando el vecino sea descubierto establecerá el enlace entre ambos.

Channel Adapter nuevo Lanza una petición Get() sobre el atributo *PortInfo* al puerto que respondió.

Switch nuevo Lanza una petición Get() sobre el atributo *SwitchInfo* al switch que respondió.

Router nuevo Lanza una petición Get() sobre el atributo *PortInfo* al puerto que respondió.

Channel Adapter existente Si la petición llegó al nodo por medio de uno de sus puertos que no se ha descubierto previamente entonces se añade el puerto a la lista de puertos descubiertos y que necesitan recibir la *trap 64* para ponerse en servicio, par lo que se lanza una petición Get() sobre el atributo *PortInfo* al channel adapter que respondió.

Switch existente Aunque se tiene constancia del switch, durante el barrido actual no se ha encontrado antes, por lo que se lanza una petición Get() sobre el atributo *SwitchInfo* al switch que respondió, pero esto sólo se hace la primera que OpenSM redescubre este switch, el otro caso no hace nada.

La cadena tiene otro eslabón cuando llega una respuesta con un atributo *SwitchInfo*. La respuesta se procesa mediante un esquema muy parecido al anterior. Principalmente se comprueba si el atributo corresponde a un switch ya descubierto o por el contrario se acaba de descubrir.

- En el caso de que el switch ya exista las acciones tomadas por el SM son:
  - Guarda la información contenida en el atributo *SwitchInfo* en la base de datos del SM.
  - Lanza una petición Get() sobre el atributo *PortInfo* a cada puerto del switch.
- En el caso de que el switch sea nuevo:
  - Guarda la información contenida en el atributo *SwitchInfo* en la base de datos del SM.
  - Lanza una petición Get() sobre el atributo *PortInfo* a cada puerto del switch.
  - Pide las tablas de encaminamiento del switch.
  - Pide las tablas de encaminamiento multicast del switch si la opción de multicast ha sido habilitada. Actualmente no se piden porque produce mucho tráfico de red inicial, pero el código que implementa esta funcionalidad se encuentra operativo.

La cadena tiene otro eslabón cuando llega una respuesta con un atributo *PortInfo*. La respuesta se procesa mediante un esquema muy parecido al de los anteriores. Si el atributo corresponde a un puerto nuevo se debe descartar puesto que el SM sólo solicita atributos *PortInfo* de puertos descubiertos previamente. En el caso de que el puerto ya haya sido descubierto las acciones son:

- Si es una respuesta a un Set() se guarda la información del atributo *PortInfo* en la base de datos del SM. Esto es posible porque la especificación define que la respuesta a un Set() también es un GetResp(). Esta información se guarda en el contexto de la MAD.

- Si es una respuesta a un Get() entonces si el puerto es válido se añade el puerto al nodo al que pertenece el puerto si no existía, de lo contrario se actualiza el *path* a este puerto puesto que el anterior ya no es válido.
- Se mira el tipo de nodo:
  - Si es un channel adapter entonces continúa su proceso como un puerto de channel adapter.
  - Si es un switch entonces continúa su proceso como un puerto de switch.
  - Si es un router entonces continúa su proceso como un puerto de router.
- Obtiene la tablas P_Key, SLVL, VL Arbitration. Actualmente sólo recoge la tabla P_Key. Como se verá en la Sección 6.6 el resto de tablas no se usan actualmente.

Puerto de un channel adapter Actualiza la información del puerto en la base de datos del SM y si el nodo final contiene a un SM entonces lanza una petición Get() sobre el atributo SMInfo al channel adapter que respondió.

Puerto de un switch Se ignora el puerto 0 y se mira el estado del puerto:

- Para el estado DOWN se desconecta el enlace en este puerto y en su vecino. Un puerto puede quedar fuera de servicio si recibe la *trap 65*.
- Para los estados INIT, ARMED, ACTIVE se realizan las acciones siguientes:
  - Copia el camino inicial del nodo padre en los campos de la MAD que está preparando para enviarla.
  - Extiende el camino inicial al próximo salto en el puerto.
  - Lanza una petición Get() sobre el atributo *NodeInfo* al nodo que se encuentra al pasar el puerto que respondió.
  - Guarda la información contenida en el atributo *PortInfo* en la base de datos del SM.

Puerto de un router Guarda la información contenida en el atributo *PortInfo* en la base de datos del SM.

6.4.3. ASIGNACIÓN DE IDENTIFICADORES LOCALES

Para llevar a cabo la asignación de identificadores locales (LID) OpenSM realiza una categorización de todos los puertos en tres listas. La razón para las listas es mantener los LIDS que están asignados tan cerca como sea posible a los LIDs ya iniciados para los puertos.

foreign Incluye todos los puertos cuyo LID es desconocido. Incluirá los puertos con un LID mayor que el tamaño de la tabla de LIDs y los puertos que no existen en la tabla de puertos. La tabla de LIDs es donde OpenSM guarda los LIDs y la tabla de puertos es la tabla donde guarda todos los puertos.

conflict Incluye todos los puertos que tienen algún problema en sus LID:

1. Puertos cuyo LMC es mayor que el LMC definido para OpenSM.
2. El objeto puerto es diferente del objeto puerto guardado en la tabla de puertos. Esto significa que ha reconocido algún puerto con este LID (puede ocurrir cuando se unen subredes).

unassigned Incluye todos los puertos que no tienen un LID asignado.

En todo este proceso el puerto donde se encuentra el SM queda fuera, aunque se le asigna el LID después de completarse el descubrimiento y antes de empezar a asignar LIDs. Los puertos *conflict* reciben el mismo trato que los pertenecientes a *unassigned*. El SM coloca el LID en el puerto por medio de una petición Set() sobre el atributo *PortInfo* de cada puerto.

6.4.4. TRAP 64

En la sección 14.2.5.1 de la especificación se define la trap 64. Cuando un puerto ha sido descubierto por primera vez el SM le envía esta trap para ponerlo en servicio. OpenSM tiene una lista donde guarda los puertos descubiertos y a los que no ha puesto en servicio, en este caso, OpenSM recorre linealmente esta lista y a cada puerto le envía la *trap 64*.

6.4.5. TABLAS DE ENCAMINAMIENTO

Cada switch en la subred usa tablas de encaminamiento para enrutar los paquetes que recibe. La especificación considera tres tipos de encaminamiento: *linear*, *random* y *multicast*. La tabla *random* actualmente no tiene soporte, pero las otras dos sí. El soporte *linear* está activo siempre y el soporte *multicast* es opcional.

El usuario también dispone de la posibilidad de introducir rutas dentro de las tablas de encaminamiento.

El algoritmo que sigue el SM para construir la tabla de encaminamiento *linear* de un switch es el siguiente:

1. Se parte de una tabla vacía.
2. Se añade una entrada que a cero saltos manda los paquetes al mismo switch.
3. Se añade una entrada por cada nodo final que está conectado a uno de lo puertos del switch.
4. En cada iteración el switch aprenderá la información *lid/puerto/saltos*. Al final cada switch conocerá cómo enrutar todos los paquetes de la subred según el LID destino.

El algoritmo utilizado por el SM para construir las tablas de encaminamiento *linear* es el siguiente:

```
bucle (total switches - 1)
{
   para i=1 hasta (total switches)
   {
      Procesa los vecinos del switch i;
   }
}

Carga rutas predefinidas por el usuario;
Procesa las tablas de cada switch;
```

Para procesar los switches vecinos (se encuentran a un salto de distancia de este switch) se realizan las acciones siguientes:

```
para j=1 hasta (total puertos del switch)
{
   Procesa el switch vecino en el puerto j;
}
```

Para procesar cada switch vecino se realizan las acciones siguientes:

```
para lid_ho=1 hasta (LID máximo en la subred)
{
   saltos = camino más corto desde este switch al nodo lid_ho;
   si (existe camino) {
      saltos++;  // salta al vecino
      if (saltos < saltos para la intersección LID/puerto) {
         saltos para la intersección LID/puerto=saltos;
      }
   }
}
```

Como se puede ver, el algoritmo de encaminamiento *linear* implementado por OpenSM no asegura que las rutas generadas estén libres de bloqueos. Además, el algoritmo asigna las rutas en base al camino más corto.

Las tablas de encaminamiento *linear* construidas por OpenSM para la subred pueden verse a continuación. Estas tablas no consideran el Server Blade **ceuta**. En la subred como GUID sólo se ha considerado los dos últimos dígitos por comodidad. En la Figura 6.4 se muestra el mapa de la subred.

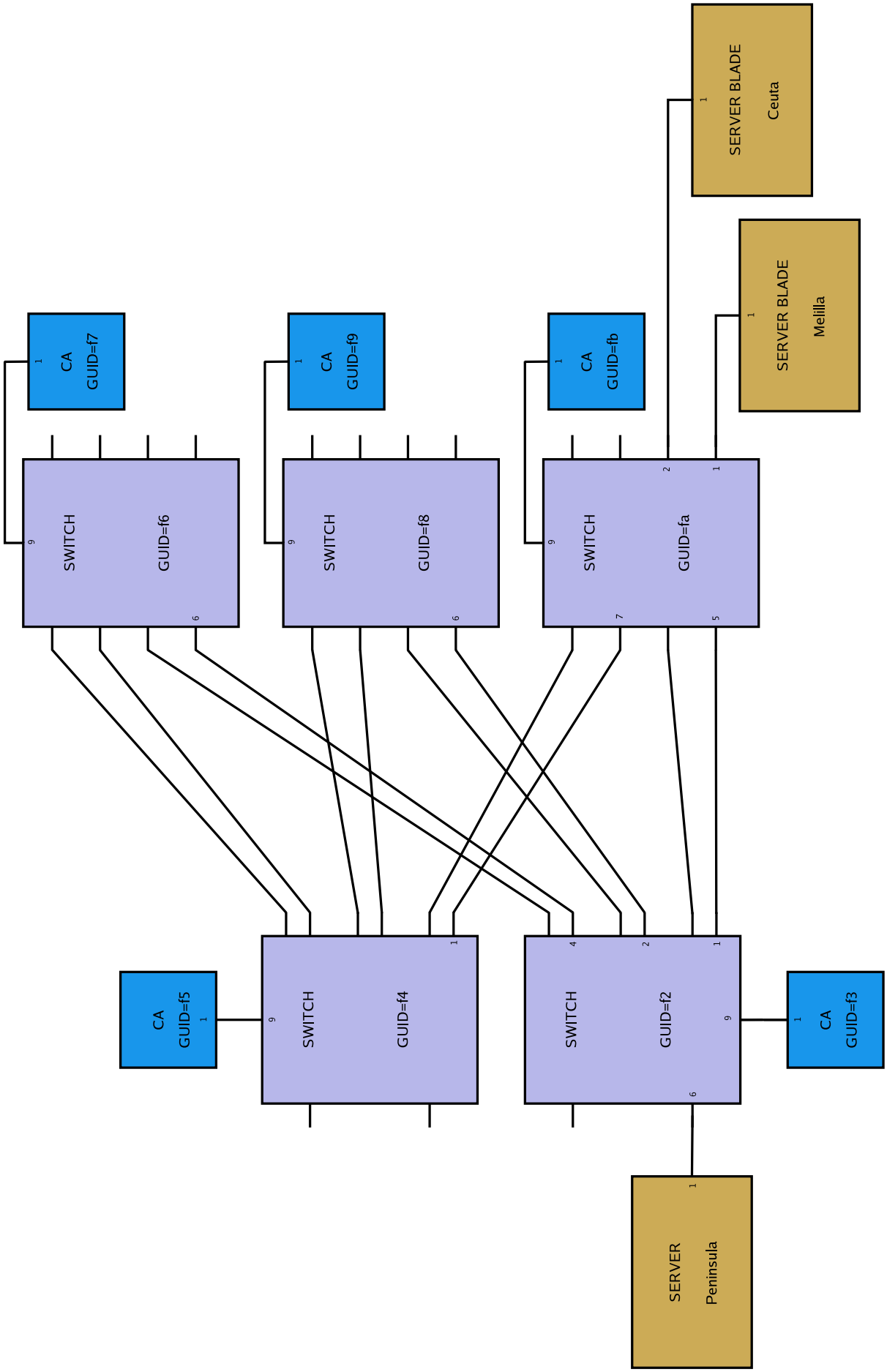


Figura 6.4: Mapa de la subred descubierta por OpenSM.

```
Switch 0xf2              Switch 0xf4              Switch 0xf6
LID    : Port : Hops     LID    : Port : Hops     LID    : Port : Hops
0x0001 : 006  : 01       0x0001 : 001  : 03       0x0001 : 005  : 02
0x0002 : 000  : 00       0x0002 : 002  : 02       0x0002 : 005  : 01
0x0003 : 001  : 01       0x0003 : 001  : 01       0x0003 : 006  : 02
0x0004 : 002  : 02       0x0004 : 000  : 00       0x0004 : 007  : 01
0x0005 : 002  : 01       0x0005 : 002  : 01       0x0005 : 008  : 02
0x0006 : 003  : 02       0x0006 : 003  : 02       0x0006 : 008  : 03
0x0007 : 004  : 01       0x0007 : 004  : 01       0x0007 : 000  : 00
0x0008 : 004  : 02       0x0008 : 004  : 02       0x0008 : 009  : 01
0x0009 : 002  : 03       0x0009 : 009  : 01       0x0009 : 007  : 02
0x000A : 008  : 02       0x000A : 001  : 02       0x000A : 006  : 03
0x000B : 009  : 01       0x000B : 002  : 03       0x000B : 005  : 02
0x000C : 001  : 02       0x000C : 008  : 02       0x000C : 006  : 03

Switch 0xf8              Switch 0xfa
LID    : Port : Hops     LID    : Port : Hops
0x0001 : 005  : 02       0x0001 : 005  : 02
0x0002 : 005  : 01       0x0002 : 005  : 01
0x0003 : 006  : 02       0x0003 : 000  : 00
0x0004 : 007  : 01       0x0004 : 007  : 01
0x0005 : 000  : 00       0x0005 : 005  : 02
0x0006 : 009  : 01       0x0006 : 005  : 03
0x0007 : 008  : 02       0x0007 : 008  : 02
0x0008 : 008  : 03       0x0008 : 008  : 03
0x0009 : 007  : 02       0x0009 : 007  : 02
0x000A : 006  : 03       0x000A : 009  : 01
0x000B : 005  : 02       0x000B : 006  : 02
0x000C : 006  : 03       0x000C : 002  : 01
```

En la Figura 6.4, el Subnet Management Agent en un InfiniScale está conectado al switch a través de una interfaz de tipo Channel Adapter. Esta es la razón por la que se ve un CA en el puerto 9 (interno) de cada InfiniScale.

6.4.6. MULTICAST

En la subred hay definidos grupos de multicast. Cuando un miembro de un grupo de multicast envía un paquete al grupo, los switches deben reenviar una copia de este paquete a todos los miembros (excepto al productor) del grupo. Para poder hacer lo anterior

el SM define las entradas de la tabla de multicast para cada switch que soporta multicast. Si una parte de la subred está detrás de switches que no soportan multicast dicha parte se quedará sin conexión de multicast.

El algoritmo empleado por el SM para construir la tabla de multicast para un switch cualquiera es el siguiente:

```
para i=1 hasta (todos los grupos de multicast)
{
   Crear un árbol de expansión para el grupo i;
}

para j=1 hasta (todos los switches de la subred)
{
   Enviarle la tabla de multicast al switch j;
}
```

Para cada grupo de multicast se construye un *árbol de expansión*. Este árbol tiene como objetivo ordenar la subred para extraer las rutas de multicast. El árbol se construye a partir de un *switch raíz*. Las ramas del árbol son los switches restantes y las hojas los nodos finales.

El criterio para elegir al switch raíz del árbol es buscar el que tiene el valor mínimo en el cálculo

$$\sum_{p=1}^{puertos\ miembro} \frac{saltos\ desde\ este\ switch\ al\ puerto\ miembro_p}{puertos\ miembro\ en\ el\ grupo}$$

y en caso de empate se elige el primero en ser encontrado. De este modo se busca el switch más cercano al resto de miembros para que los caminos sean mínimos (es un método aproximado).

El árbol se construye recursivamente. En cada etapa, los puertos alcanzables se dividen en subconjuntos disjuntos de puertos miembro que pueden ser alcanzados a través de un puerto de un switch dado. La construcción se mueve hacia abajo por cada rama calculando para su propio subconjunto de puertos miembro. La recursividad queda asegurada en la profundidad de 64. La especificación define que el camino entre dos posibles nodos es como máximo de 64 saltos, por lo que el algoritmo no profundizará más allá de este nivel.

El algoritmo construye las tablas de multicast para que cuando un switch reciba

un paquete por un puerto miembro de un grupo multicast, dicho switch reenvíe el paquete por todos los puertos miembros de ese grupo, excepto por el puerto de llegada para evitar bucles. En un switch cualquiera si el paquete llega desde el nodo padre entonces se reenviará a todos los nodos hijos; si llega desde un nodo hijo se reenviará a todos los nodos hermanos y al padre, propagando de este modo un paquete por todos los puertos miembro del grupo.

El algoritmo define rutas libres de bucles por la separación de nodos en conjuntos disjuntos. Explícitamente también separa los enlaces que unen a los nodos. Si de un nodo A hay enlaces a los nodos B y C, y a su vez, un nodo D tiene enlaces con B y C, el algoritmo dirá que a D los paquetes multicast se enrutan por B o por C. Las dos rutas son excluyentes como puede verse en la Figura 6.5.

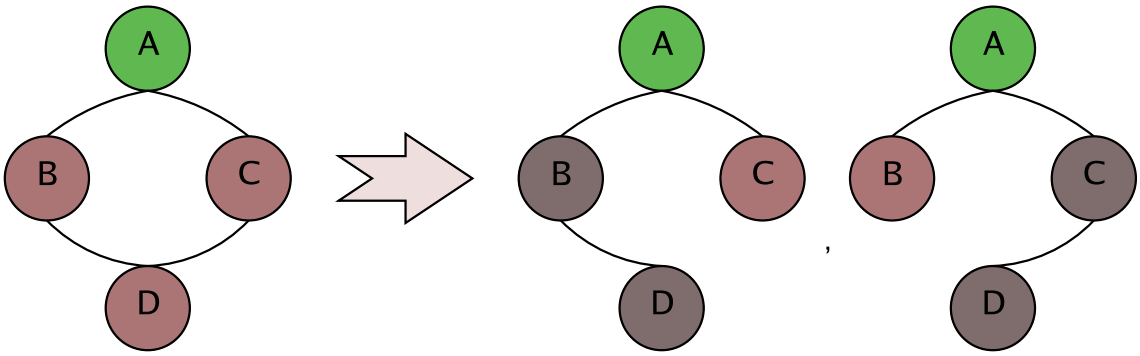


Figura 6.5: Ejemplo de separación de caminos.

Por el momento las tablas multicast existentes se borran, pero en el futuro el árbol se debe construir a partir de las tablas multicast existentes si el usuario configura el SM para preservar las mismas.

6.4.7. ASIGNACIÓN DEL ESTADO DE LOS PUERTOS

Para que pasen paquetes de datos por los puertos el SM debe configurarlos. La configuración de los puertos no se termina una vez que el SM ha enviado la información de configuración a través de un método Set() sobre el atributo *PortInfo*. Es necesario que el SM envíe las ordenes necesarias para que los puertos se pongan en estado “activo”. Para todos los puertos configurados en la subred el SM envía las ordenes siguientes:

- IB_LINK_NO_CHANGE éste le dice al puerto que no va a recibir más información de configuración.
- IB_LINK_ARMED éste le dice al puerto que se prepare para recibir paquetes de datos.

- IB_LINK_ACTIVE éste le dice al puerto que ya puede recibir paquetes de datos.

6.4.8. SUBRED CONFIGURADA

Una vez que el SM da por configurada la subred lanza el evento *SUBNET UP* para que el resto de módulos conozcan este hecho. El SM transita a un estado de espera activa.

6.5. RECONFIGURACIÓN DE LA SUBRED

Cuando el estado del enlace de un puerto en un switch cambia, el switch envía una trap número 128 al SM. Antes, en la etapa de configuración, el SM ha activado el bit *PortInfo:CapabilityMask:IsTrapSupported* para habilitar el mecanismo de traps en el switch. La trap 128 está definida en la tabla 115 de la especificación y en la sección 14.3.4 se define el envío de la trap.

Cuando el SM recibe una trap 128 fuerza un barrido completo de la subred para absorver cualquier cambio.

El SM se puede configurar para realizar un barrido de la subred automáticamente transcurrido un período de tiempo determinado, aunque también puede deshabilitarse. Por defecto el barrido automático se inicia cada diez segundos.

6.6. CALIDAD DE SERVICIO

La especificación define mecanismos para proporcionar calidad de servicio en InfiniBand, a saber, *Niveles de Servicio*, *Particiones*, *Mapas SLtoVL* y *Tablas de Arbitraje*. El Subnet Manager debe configurarlos antes de que los paquetes de datos atraviesen la subred, aunque también puede hacerlo mientras lo hacen.

Sin embargo, OpenSM no configura actualmente la subred para proporcionar calidad de servicio. ¿Cómo, entonces, la subred llega a estar configurada? La respuesta se encuentra en el apartado 7.6.5 de la especificación, donde se puede leer que si sólo se soporta un canal virtual (VL) entonces no es necesario mantener ni el mapa SLtoVL ni la tabla de arbitraje en los puertos. En el caso de definir más de un canal virtual de datos sí es necesario que los puertos soporten los dos mecanismos.

OpenSM actualmente implementa exactamente la primera opción, configurando los puertos con un sólo canal virtual de datos operativo, a pesar de que los puertos soportan 8 canales virtuales. En realidad, existe un canal virtual de datos VL_0, configurado por el OpenSM y el canal virtual de gestión VL_{15} obligatorio. Con los dos canales se define un esquema de prioridad simple donde todos los paquetes del canal VL_{15} se envían antes de cualquier paquete del canal VL_0. Como puede verse, con este planteamiento existe el riesgo de inanición en los paquetes del canal menos prioritario VL_0.

Para el resto de mecanismos, OpenSM utiliza un valor por defecto de SL_0. El nivel de servicio es obligatorio en la cabecera LRH de los paquetes que atraviesan la subred. Respecto a las particiones, OpenSM no segrega la subred en particiones.

Se ha planificado para versiones posteriores a la 0.2.0 el uso de más canales virtuales operativos en los puertos de la subred para proporcionar calidad de servicio.

Capítulo 7

CONCLUSIONES Y TRABAJO FUTURO

Hay que indicar, antes de nada, que los objetivos inicialmente planteados han sido cubiertos en la medida en que el propósito principal del trabajo ha sido alcanzado. Es decir, se ha obtenido la suficiente información como para que cualquier persona pueda empezar a trabajar con la red e intente trasladar propuestas a OpenSM.

El objetivo se ha conseguido haciendo uso de una metodología habitual en trabajos de similares características. En cualquier caso, y como el objetivo que era, debía ser cubierto de forma satisfactoria, como finalmente así ha sido. A ello ha contribuido, sin duda, la ayuda facilitada por varias personas.

Al margen de la dificultad implícita del proyecto, el factor que más ha condicionado el trabajo en el tiempo ha sido el ser los primeros que trataban con este tipo de redes respecto al hardware y software. Este hecho, se ha traducido, eso sí, en una valiosa esperiencia profesional.

En este último capítulo se incluyen las principales conclusiones del trabajo aquí presentado así como algunas de las líneas y tareas que podrían dar continuidad al mismo.

7.1. CONCLUSIONES

Respecto a las conclusiones del proyecto, algunas pueden parecer demasiado obvias, pero se ha decidido incluirlas, bien por estar directamente relacionadas con el trabajo realizado, bien porque pueden ser de ayuda a futuros investigadores. Las más destacables se resumen a continuación:

- Poner en marcha un sistema experimental entraña un enorme esfuerzo, aun más, cuando inicialmente no se tiene la experiencia suficiente para ello. A pesar de estos problemas se han conseguido los resultados esperados.

- La documentación del fabricante está destinada a profesionales del sector y no para usuarios noveles, lo que se tradujo en una dificultad añadida al comienzo del trabajo. A ésto debe añadirse el haber sido los primeros y únicos, por el momento, en el Instituto, que han trabajado con dispositivos InfiniBand comerciales.

- La gestión de los nodos Server Blade ha sido excesivamente complicada. Aunque no se pone en duda sus grandes ventajas, que han podido ser comprobadas, en el campo de la investigación, el hecho de tener sólo dos nodos de los doce posibles en el chasis, no permite obtener en absoluto el beneficio merecido a tanto esfuerzo.

- La plataforma GNU/Linux es la base de los trabajos que se desarrollan para estos dispositivos. Una vez más se ha demostrado que es idónea y versátil para este tipo de trabajos.

- OpenSM es la herramienta más adecuada para utilizarla como base en la implantación de nuevas propuestas en el ámbito de los Subnet Managers.

- OpenSM implementa de forma muy sencilla mecanismos tales como encaminamiento, multicast, calidad de servicio, y en general, todos las tareas de un Subnet Manager. A partir de ahora, los investigadores pueden estudiar sus propuestas en un Subnet Manager real. En el caso de conseguir resultados es posible que OpenSM incluya en su código estas propuestas, lo que puede dar prestigio al que las aporte.

- Actualmente existe un esfuerzo importante de desarrolladores a nivel mundial, que con el apoyo de grandes empresas del sector, están dando soporte a InfiniBand.

- Este trabajo ha supuesto para el autor una fuente de experiencia considerable en el campo de la administración de sistemas, particularmente en sistemas GNU/Linux y en redes InfiniBand. El estudio del código de un Subnet Manager ha proporcionado el camino para comprender un proyecto de gran envergadura, apreciar el trabajo en

grupo realizado por otros desarrolladores, y por lo tanto, una experiencia real fuera de la universidad.

- Con el estudio realizado en este proyecto es posible, y más sencillo, abordar la tarea de mejorar los algoritmos o mecanismos implementados, tales como encaminamiento, calidad de servicio, etc, en la medida en que se podrán llevar a la red real y comprobar su funcionamiento. Esta es, sin duda, una línea de trabajo con grandes posibilidades de ser desarrollada en un futuro inmediato pues se han puesto las bases para que así sea.

7.2. TRABAJO FUTURO

La metodología seguida para el desarrollo de este trabajo, así como las características de los resultados obtenidos han permitido observar detalles que invitan a plantear una continuidad al mismo. No sería atrevido decir, a la vista de los resultados, que las tareas que podrían surgir como consecuencia directa de este trabajo supondrían no uno, sino varios estudios de la misma o mayor envergadura.

Se indican a continuación algunas de esas líneas de trabajo que podrían dar continuidad al estudio que aquí se ha presentado:

- Una de las posibilidades sería ampliar la red, en cuyo caso, bastaría con adquirir nuevos componentes con características diferentes, para conseguir formar una red heterogénea, lo que sin duda, dará mayores opciones de configuración de la red.
- Como es de suponer, todos los sistemas informáticos son actualizados, e InfiniBand está en contínua evolución. Debido a ésto, cualquiera que trabaje en este campo, debe conocer y probar las últimas versiones de software de sistema para estos dispositivos.
- La última línea de trabajo del proyecto OpenIB es el soporte abierto en los nuevos núcleos de Linux 2.6. Su implantación, así como una posible inserción de éste dentro del kernel en un futuro, podrían favorecer la adopción de InfiniBand como una tecnología estándar, al igual que Ethernet. No hace falta decir, que el trabajo de todos los investigadores para llevar a cabo las posibles pruebas, y su evaluación, es una ayuda imprescindible para la comunidad.

- Con el estudio realizado en este proyecto, es posible abordar la tarea de mejorar los algoritmos en aspectos de implementación tales como encaminamiento, calidad de servicio, etc.

Algunos de estos trabajos futuros están ya en marcha por parte de algunos investigadores del grupo RAAP, mientras que otros de ellos comenzarán a ser desarrollados en breve dentro del I^3A.

Bibliografía

[1] TopSpin Communications. http://www.topspin.com.

[2] Kingston Technology Company. http://www.kingston.com.

[3] Intel Corporation. http://www.intel.com.

[4] Intel Corporation. *Linux System Software for the InfiniBand Architecture*. 2002.

[5] The Fibre Channel Industry Association (FCIA). http://www.fibrechannel.org.

[6] William T. Futral. *InfiniBand Architecture: Development and Deployment–A Strategic Guide to Server I/O Solutions*. Intel Press, 2001.

[7] PCI Industrial Computer Manufacturers Group. http://www.picmg.org.

[8] InfiniBand Trade Association. *InfiniBand Architecture Specification Volume 1. Release 1.0*, Octubre 2000.

[9] InfiniBand Trade Association. *InfiniBand Architecture Specification Volume 1. Release 1.0.a*, Junio 2001.

[10] Seagate Technology LLC. http://www.seagate.com.

[11] Mellanox Technologies Ltd. http://www.mellanox.com.

[12] Mellanox Technologies Ltd. *InfiniHost MT23108 Programmer's Reference Manual, Rev. 0.98*. Mellanox Technologies Ltd., 2000.

[13] Mellanox Technologies Ltd. *InfiniHost SDK User's Guide - Linux x86 Platforms, Rev. 1.00*. Mellanox Technologies Ltd., 2002.

[14] Mellanox Technologies Ltd. *MST User's Guide, Rev. 0.10*. Mellanox Technologies Ltd., 2002.

[15] Mellanox Technologies Ltd. *Mellanox Software Tools Release Notes, Rev. 1.4.8*. Mellanox Technologies Ltd., 2003.

[16] Reunión mundial de fabricantes de Blades. http://www.serverbladesummit.com.

[17] Inc. Myricom. http://www.myri.com.

[18] Open Source. Proyecto OpenIB. http://www.openib.org.

[19] PCI Special Interest Group (PCI-SIG). http://www.pcisig.com.

[20] Tom Shanley. *InfiniBand Network Architecture*. Addison Wesley Professional, 2002.

[21] Inc. Voltaire. http://www.voltaire.com.

[22] Intel y Otros. Proyecto Linux InfiniBand. http://infiniband.sourceforge.net.