UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Técnicas de Control de Flujo y Encaminamiento para Mejorar las Prestaciones en Redes en Chip

Tesina final de Máster en Ingeniería de Computadores

Curso 2013/2014

Escuela Técnica Superior de Ingeniería Informática

Miguel Gorgues Alonso

Director:

José Flich Cardo

Septiembre de 2014

Agradecimientos

A mis padres por darme todo su cariño, todo su apoyo y brindarme todas las oportunidades que he tenido en mi vida con su esfuerzo y sacrifio.

A Salva, que allá donde estés, en la distancia sé que estas cuidando de mí y dándome la fuerza necesaria para seguir adelante y poder conseguir todos mis objetivos.

A Pepe y José María, por haberme dado la oportunidad de trabajar con ellos y por todas las horas de trabajo realizado con el fin de que esta tesina haya tenido éxito.

A Vicent, compañero inseparable en estos dos últimos años, con quien he compartido grandes momentos en el trabajo, máster, almuerzos..

A todos los miembros del GAP, por los momentos amenos y su ayuda en el trabajo.

A Hector, Luis, Sergio, José, Javi, José, Carlos y el resto de compañeros de la carrera, por estos años de alegrías.

A María, Dani, Carles, Mari, ... y el resto de familia y amigos, por todo el apoyo recibido y por soportarme durante todos estos años.

A todos vosotros, gracias de corazón.

Índice de contenidos

1 Introducción. 11

1.1 Descripción del problema. 11

1.2 Objetivos. 14

1.3 Estructura del Trabajo. 14

2 Conceptos Preliminares. 17

2.1 Conceptos de Redes. 17

2.2 Trabajo Relacionado. 27

2.2.1 Mecanismos de Control de Flujo y Encaminamiento Adaptativo. 27

2.2.2 Técnicas de Alivio de la Congestión. 28

2.3 Entorno de Simulación. 29

3 Descripción de los Mecanismos. 31

3.1 TBFC: Type Based Flow Control. 31

3.1.1 TBFC con Flit-Level Crossbar Switching. 31

3.1.2 TBFC con Packet-Level Crossbar Switching. 35

3.2 Algoritmo de encaminamiento Safe/Unsafe. 36

3.2.1 Justificación Algoritmo Libre de Bloqueo. 40

3.2.2 Ejemplo de TBFC+SUR. 42

3.3 Filtro End-Point-Congestion (EPC). 44

3.3.1 Estimación del Coste de EPC. 48

4 Evaluación de los Mecanismos. 49

4.1 TBFC+SUR. 49

4.1.1 Parámetros del análisis. 49

4.1.2 Resultados de rendimiento. 50

4.1.3 Equilibrado de la utilización VCs. 54

4.2 EPC. 54

4.2.1 Parámetros del análisis. 55
4.2.2 Resultados de Rendimiento. 56

5 Conclusiones y Trabajo Futuro **61**
5.1 Conclusiones. 61
5.2 Trabajo Futuro. 62
5.3 Artículos Publicados y Relación con la Tesina. 62

Índice de figuras

1.1 Ejemplo problema utilización desequilibrada de colas. 12
1.2 Ejemplo problema de propagación de la congestión en el uso de encaminamiento adaptativo. 13

2.1 Red en el chip (NoC).Copyright[BM06] 17
2.2 Arquitectura conmutador. Copyright[DT01] 18
2.3 Flit-level Crossbar Switching. 22
2.4 Packet-level Crossbar Switching. 23
2.5 Red de medio compartido. 23
2.6 Red conmutada directa. 24
2.7 Red conmutada indirecta. 25

3.1 Control de flujo tradicional basado en créditos asumiendo *flit-level crossbar switching*. 32
3.2 Control de flujo TBFC asumiendo flit-level crossbar switching. 33
3.3 Ejemplo de la lógica del conmutador en el puerto de entrada. 35
3.4 Control de flujo TBFC asumiendo *packet-level crossbar switching*. . . . 36
3.5 Ejemplo de *deadlock-freedomness* en una red tipo toro 1D 42
3.6 Ejemplo TBFC+SUR. 42
3.7 Dos conmutadores conectados para el ejemplo de la Figura 3.6 43
3.8 Ejemplo de la ramificación de la congestión. 45
3.9 Arquitectura de conmutador base incluyendo EPC. 46
3.10 Información de control del puerto de salida y EPC. 47

4.1 Resultados de rendimiento en mallas 8×8, tráfico bit reversal y transpose para TBFC+SUR. 51
4.2 Resultados de rendimiento en mallas 8×8, tráfico uniform y hotspot para TBFC+SUR. 52

4.3 Resultados de rendimiento en toros 8×8, tráfico bit-reversal y transpose para TBFC+SUR. 53
4.4 Resultados de rendimiento en toros 8×8, tráfico uniforme y hotspot para TBFC+SUR. 54
4.5 Ejemplo malla 2D. 55
4.6 Utilización de canales. 55
4.7 Resultados de rendimiento para distribución de tráfico *hotspot* en una malla 4×4 para EPC. El porcentaje de *hotspot* es 30%. 57
4.8 Resultados de rendimiento para distribución de tráfico *hotspot* en una malla 4×4 para EPC. El porcentaje de hotspot es 70%. 58
4.9 Resultados de rendimiento para la distribución de tráfico uniforme en una malla 4×4 para EPC. 59
4.10 Resultados de rendimiento para distribución de tráfico *hotspot* en una malla 8×8 para EPC. El porcentaje de hotspot es 30%. 59

Índice de tablas

4.1 Parámetros y valores usados para los experimentos en la malla 2D. . . . 50
4.2 Parámetros y valores usados para los experimentos en el toro 2D. . . . 50
4.3 Parámetros y valores usados para los experimentos de EPC. 56

Chapter 1

Introducción.

1.1 Descripción del problema.

Hoy en día las redes en el chip (*network on chip*; NoC) [GL06] han pasado de ser un concepto a una tecnología. Las NoC permiten una comunicación eficiente dentro del chip en términos de productividad, área y consumo. Las NoC permiten remplazar tanto redes de comunicación complejas (*fully connected*), como soluciones simples, *buses* o *crossbars*, que no presentan una buena escalabilidad.

El concepto de NoC proviene del dominio de red fuera del chip, donde las interconexiones de altas prestaciones han sido diseñadas para construir grandes instalaciones de HPC o *datacenters*. El cambio de entorno (de HPC o *datacenters* al chip) plantea un desafío en el diseño de las NoC, ya que los ingenieros deben enfrentarse a limitaciones muy restrictivas en el diseño del chip. La limitación del área impone que tengamos que realizar diseños con los mínimos recursos posibles, optimizando al máximo dichos diseños. Sin embargo, lo más importante es proporcionar un diseño con un consumo de energía bajo. Al disponer de una cantidad acotada de energía dentro del chip aparecen severas limitaciones en los recursos a utilizar. Los componentes con mayor consumo de energía (principalmente memorias) deben reducirse al mínimo o incluso eliminarse por completo. Sin embargo, los ingenieros deben enfrentarse a estas limitaciones sin dejar de ofrecer el rendimiento esperado. Estos dos son, por lo general, requisitos opuestos.

Un ejemplo claro de este problema es el hecho de utilizar diseños basados en *wormhole switching*, donde los mensajes bloqueados en la red se almacenan a lo largo de su ruta, ocupando las colas en los diferentes conmutadores. Ésto permite reducir el tamaño de la cola (pudiendo ser más pequeño que el tamaño del mensaje). Sin embargo, este mecanismo de conmutación supone un gran impacto en el rendimiento. En efecto, el bloqueo del mensaje en varios conmutadores puede causar graves proble-

mas de saturación. Además, el uso de *wormhole switching* impone más restricciones arquitectónicas. Por ejemplo, las operaciones de comunicación colectiva no pueden implementarse a menos que se utilicen más colas. Ésto es necesario para evitar las posibilidades de inducir bloqueos (*deadlocks*).

Además, la forma en que se gestionan los componentes de las colas puede conducir a un consumo excesivo de energía. De hecho, su uso debe ser lo más equilibrado posible con el fin de economizar energía. Por lo general, los algoritmos de encaminamiento se basan en el uso de diferentes canales virtuales, especialmente en conmutación *wormhole*. Además, se utilizan varios canales virtuales para evitar los bloqueos a nivel de protocolo, que pueden ser inducidos por las dependencias entre los mensajes generados por los protocolos de coherencia del nivel superior. Ésto nos lleva al uso de un gran número de colas y, por tanto, a una ineficiencia de recursos si estos no se utilizan equilibradamente.

El desequilibrado en el uso de las colas también se produce al utilizar algoritmos de encaminamiento adaptativos. Este tipo de encaminamiento permite mejorar las prestaciones de las NoCs, debido a que nos ofrecen una mayor flexibilidad a la hora de encaminar los mensajes a través de la red. Sin embargo, este tipo de encaminamiento conlleva principalmente dos problemas: el uso desequilibrado de las colas y la ramificación de la congestión en la red. El primer problema aparece debido a que se diferencia entre colas de encaminamiento adaptativo y colas de encaminamiento de escape. Las colas de escape solo se utilizan cuando se inyecta una alta carga de mensajes en la red. Por tanto, la mayor parte del tráfico de la red pasa por las colas adaptativas. El segundo problema que aparece es la ramificación de la congestión en la red. El tener una mayor flexibilidad en el encaminamiento de la red facilita que en caso de que exista congestión debido a un *hotspot* ésta se extienda rápidamente por toda la red, haciendo que la mayoría de las colas estén ocupadas por mensajes con destino al *hotspot*.

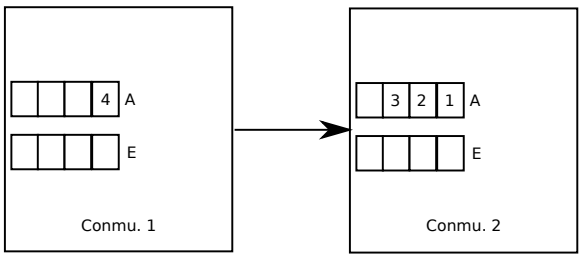


Figura 1.1: Ejemplo problema utilización desequilibrada de colas.

La Figura 1.1 muestra un ejemplo de la utilización desbalanceada de las colas adaptativas, en este caso asúmase que estos dos conmutadores son parte de una red mayor, el conmutador 1 (Conmu.1) decide enviar el mensaje número 4 al conmutador 2

(Conmu.2). El problema del desbalanceado de la carga hace que como queda suficiente espacio en la cola adaptativa, el conmutador 1 lo envíe a esta cola. Dejando sin utilizar la cola de escape.

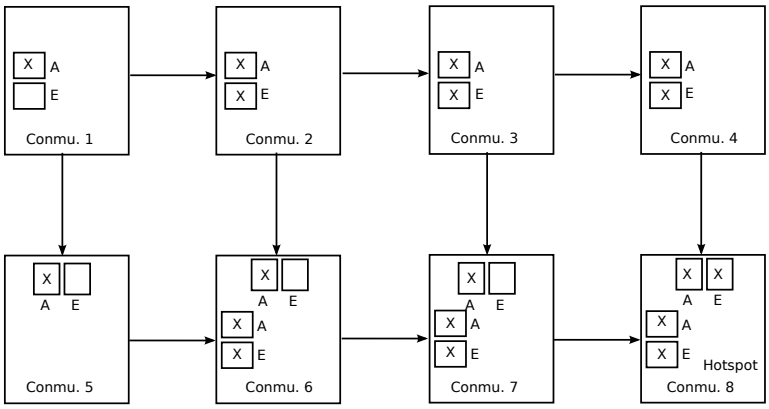


Figura 1.2: Ejemplo problema de propagación de la congestión en el uso de encaminamiento adaptativo.

La Figura 1.2 muestra un ejemplo de cómo un algoritmo adaptativo ha llenado las colas de mensajes con destino al *hotspot*. Asúmase que las X son mensajes con destino al *hotspot*. El algoritmo propaga por toda la red mensajes con destino al *hotspot*, utilizando y llenando todas las colas adaptativas (marcadas como A). Además, las colas de escape (E) se llenan con mensajes con destino al *hotspot* debido a que las colas adaptativas se han llenado. Al final toda la zona de la red se ha congestionado debido al *hotspot*

Estos dos problemas (desequilibrado de colas y ramificación de la congestión) repercuten negativamente en las prestaciones de la NoC. El desequilibrio producido por la imposición de que las colas de escape solo se utilizan cuando no hay posibilidad de encaminar el mensaje por alguna de las colas adaptativas, elimina la posibilidad del equilibrado de la utilización de las colas con cargas de tráfico bajas.

Por otra parte, la ramificación de la congestión en la red con encaminamiento adaptativo se produce mucho más rápido, llenando así toda la red de mensajes con destino al *hotspot*. Esto conlleva que el interfaz de red receptor del tráfico *hotspot* sea el cuello de botella de la red. Con lo cual, tanto la productividad como la latencia en la red verán afectadas sus prestaciones, debido a que, la mayoría de las colas están ocupadas por los mensajes al *hotspot* y el resto de tráfico de la red no puede avanzar o ser inyectado.

1.2 Objetivos.

En esta tesina abordamos los problemas mencionados anteriormente: conseguir el uso equilibrado de las colas y evitar ramificación de la congestión en la red. Para el primero de ellos (conseguir un uso equilibrado de las colas) proponemos un novedoso mecanismo de control de flujo, denominado *Type-Based Flow-Control* (TBFC). Este mecanismo está diseñado para usar los mínimos recursos, pero permitiendo tanto el uso de *virtual cut-through* (con las ventajas que ofrece) como de *wormhole*. Además, TBFC está preparado para un nuevo tipo de algoritmo de encaminamiento, el cual decide sobre el encaminamiento en función de cómo se etiquete un mensaje. De hecho, aplicamos este nuevo algoritmo de encaminamiento adaptativo sobre TBFC. El algoritmo, denominado *safe/unsafe routing* (SUR), clasifica los mensajes como *safe* o *unsafe* en función de las posibilidades que tiene el mensaje para inducir *deadlock*. Los mensajes *safe* se mueven a través de la red sin restricciones, mientras que los mensajes *unsafe* se encaminan solo a través de rutas libres de bloqueo. Cuando se combinan, TBFC y SUR logran un equilibrado perfecto de los recursos permitiendo un uso óptimo. Para el segundo de los problemas proponemos una nueva técnica de filtrado de mensajes, denominada *End Point Congestion* (EPC). Esta técnica limita la ramificación de los mensajes con el mismo destino aunque sigue permitiendo el encaminamiento adaptativo. Con esta limitación los mensajes que se dirigen hacia el *hotspot* solo pueden ocupar un VC por puerto de salida, dejando libre el resto de los VCs para el tráfico en la red que no se dirige a este destino.

Esta tesina esta orientada hacia trabajos de investigación sobre NoCs y protocolos de coherencia. Las técnicas propuestas en esta tesina se utilizarán como punto de partida de la tesis, orientada hacía la mejora de las NoC en entornos con protocolos de coherencia.

1.3 Estructura del Trabajo.

El resto del presente trabajo se organiza de la siguiente manera. En el Capítulo 2.2, se explican conceptos básicos de las redes en chip, se muestran los trabajos propuestos previamente para solucionar los problemas planteados anteriormente y se presenta el entorno de simulación desarrolado para el trabajo. En el Capítulo 3 se describen de una forma detallada las técnicas implementadas. En el Capítulo 4 se muestran y analizan los resultados obtenidos en los experimentos lanzados. Por último, en el Capítulo 5, se resumen las conclusiones obtenidas y se describe el trabajo futuro, así como la relación

de publicaciones obtenidas y relacionadas con la tesina.

Chapter 2

Conceptos Preliminares.

En este capítulo se describen conceptos relacionados con la investigación. Primeramente se introducen las redes de chip y los conceptos básicos. Seguidamente, se muestra un estudio de los trabajos previos relacionados con el control de flujo y los algoritmos de encaminamiento y finalmente se presenta el entorno donde se desarrolla este estudio.

2.1 Conceptos de Redes.

En esta sección primeramente se muestran los elementos básicos de las redes en el chip. Estas redes están compuestas por varios tipos de dispositivos como muy bien ilustra el documento [BM06] en la sección 2.1 del cual se ha obtenido la Figura 2.1. Estos dispositivos son:

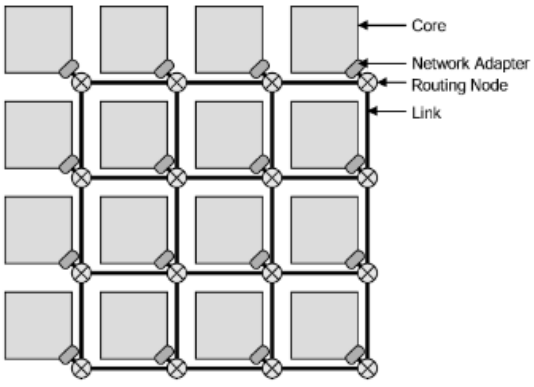
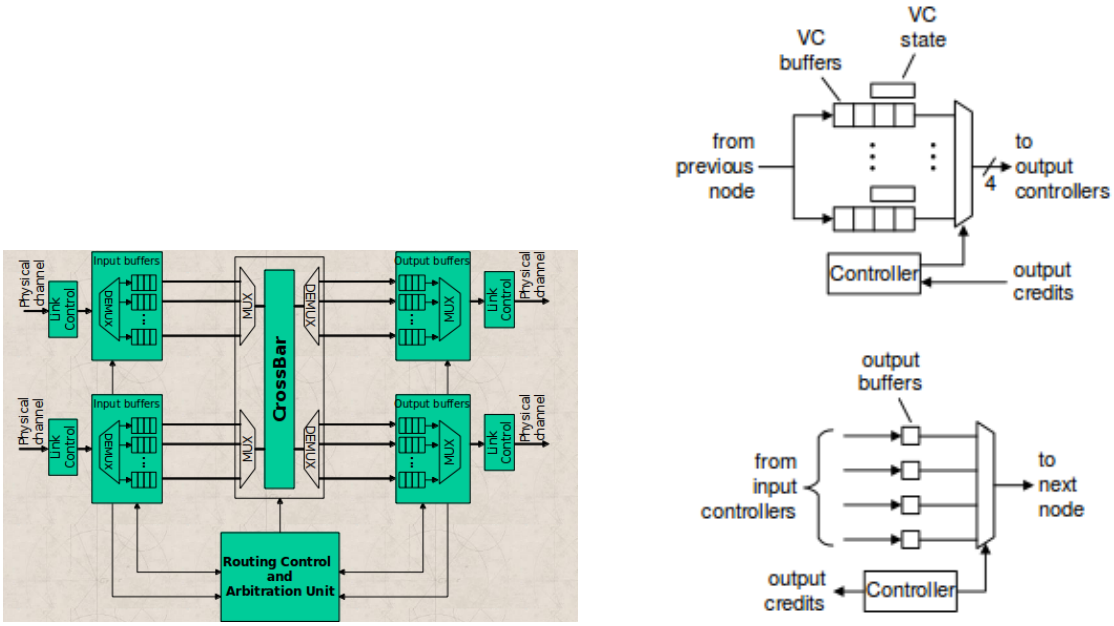


Figura 2.1: Red en el chip (NoC).Copyright[BM06]

- **Interfaces de red**. Implementan la interfaz por la cual los procesadores (*cores*) se comunican con la NoC. Su función es la de disociar el cómputo (realizado en el *core*) de las comunicaciones (en la NoC).

- **Enlaces**. Conectan los diferentes conmutadores entre sí, proporcionando el ancho de banda de la red. Los enlaces consisten en uno o más canales lógicos o físicos.
- **Conmutadores**. Éstos son los encargados de dirigir los mensajes por la red de acuerdo con el protocolo elegido. Los conmutadores implementan los algoritmos de encaminamiento.



(a) Conmutador (b) Sistema entrada

Figura 2.2: Arquitectura conmutador. Copyright[DT01]

Los conmutadores están compuestos básicamente por puertos de entrada, puertos de salida, un elemento de conmutación y la lógica asociada. La figura 2.2(b) muestra un ejemplo de los componentes de un conmutador.

Los puertos están compuestos de colas (memorias), las cuales tienen como función almacenar temporalmente las unidades de información (llamadas *flits*) que atraviesan la red. Cada cola esta asociada a un puerto ya sea de entrada o de salida. Los puertos de entrada se encargan de recibir las unidades de información que provienen del conmutador anterior, mientras que los puertos de salida están encargados de volcar las unidades de información al enlace de salida. Normalmente no se implementan las colas en los puertos de salida.

Cada puerto puede contener más de una cola (varias memorias) o una única memoria puede estar mapeada en varias colas. A cada una de estas colas se le denomina canal virtual (*virtual channel*; VC). Los canales virtuales asociados a un mismo puerto

comparten el enlace físico y el puerto es el encargado de multiplexar dicho enlace. Cuando un *flit* se bloquea en un VC otras unidades de mensaje almacenadas en otro VC pueden cruzar el conmutador. Estos VCs se organizan en redes virtuales (*Virtual Network; VN*). Las VN son un subconjunto de VCs que se utilizan para el encaminamiento de los mensajes. Los canales virtuales correspondientes a cada red virtual son disjuntos. Estas redes virtuales se utilizan por ejemplo para el encaminamiento de distintos tráficos en protocolos de coherencia o para distintas técnicas de prevención de la congestión de tráfico para evitar el *Head of Line Blocking* (HOL blocking). El *HOL blocking* se produce cuando un paquete que podría avanzar por la red, no lo hace porque el primer paquete de la cola está bloqueado.

Todos los puertos de entrada están conectados con todos los puertos de salida a través de un elemento de conmutación no bloqueante, denominado *crossbar*, como muestra la figura 2.2(a). Además, dicha figura muestra el subsistema de entrada compuesto por varias colas, una por cada canal virtual (VC), y el controlador que es el que decide si puede acceder al *crossbar* para enviar el mensaje. En el sistema de salida, las colas solo tienen una posición y es el controlador quien decide si es posible mandar el mensaje al siguiente nodo.

Otros dos elementos del conmutador que no están representados en la Figura 2.2 son la unidad de encaminamiento y la unidad de arbitraje. La primera se encarga de decodificar información de control contenida en el *flit* y a partir de esta información realizar el cómputo de cuales son los puertos de salida más adecuados para el *flit*. El segundo elemento se encarga de distribuir el acceso al *crossbar* entre múltiples peticiones al mismo puerto de salida y de configurar dicho *crossbar*.

Como se ha comentado los conmutadores son los responsables de las funciones de encaminamiento y transmisión de los datos. Éstos pueden implementarse usando un modelo de conmutador de 4 etapas, siendo estas etapas las siguientes:

- **IB**. *Input Buffer*. El conmutador recibe el *flit*, almacenando éste en el apropiado VN y VC.

- **RT**. *Routing*. En esta etapa se hace el encaminamiento del *flit*, calculando el puerto de salida para el mensaje.

- **VA_SA**. *Virtual channel Allocation* y *Switch Allocation*. Esta es la etapa de arbitraje, se modelan dos etapas en una sola. En la primera (VA) se calculan los *flits* que pueden acceder a la segunda etapa (SA), para cada puerto de entrada. En la segunda cada puerto de salida, selecciona uno de los puertos de entrada con

datos de entrada válidos que será encaminado por el puerto de salida. Entonces los puertos de entrada reciben notificación para transmitir sus datos atravesando el *crossbar* en la siguiente etapa.

- **X**. Cross. Transmisión del mensaje por la salida del conmutador.

Las técnicas de conmutación determinan cuándo y cómo se reservan los recursos de la red para un mensaje. A continuación se describen las principales técnicas de conmutación usadas en NoCs.

- **Conmutación de Circuito**. Esta técnica de conmutación establece una ruta reservada entre el nodo emisor y el nodo receptor antes de transmitir el mensaje. Esta ruta se establece mediante la inyección de un flit de cabecera que es transmitido a través de la red reservando los enlaces que atraviesa hasta llegar a su destino. Una vez se ha alcanzado el destino, se transmite el mensaje completo por la ruta establecida sin ningún impedimento. Una vez finalizada la transmisión se liberan los recursos de la ruta establecida y los enlaces pueden ser utilizados para establecer otras rutas. El tiempo de establecimiento de la red es muy elevado comparado con el tiempo de transmisión de los mensajes.

- **Conmutación de Mensaje**. En lugar de reservar toda la ruta para transmitir el mensaje, existen técnicas que conmutan la información a nivel de mensaje. Las técnicas de conmutación de mensaje son las siguientes:

  Store and Forward (SAF). Esta es la técnica más básica de conmutación de mensaje. Cuando un mensaje llega al puerto de entrada, el conmutador espera a que haya sido recibido todo el mensaje antes de empezar a procesar el encaminamiento del mensaje y poder ser transmitido. SAF requiere un tamaño de la cola igual o superior al tamaño del mensaje.

  Virtual Cut-through (VCT). Esta técnica permite empezar el computo de los puertos de salida una vez el flit de cabecera ha sido recibido en el conmutador, sin tener que esperar al resto del mensaje como en SAF. El mensaje puede avanzar por las diferentes etapas del conmutador una vez la cabecera ha sido recibida, la latencia base del conmutador disminuye con esta técnica. Sin embargo, VCT requiere al igual que SAF un tamaño de cola igual o superior a la longitud del mensaje. Esto permite que si un mensaje se bloquea en la red, todo el mensaje se almacene en el mismo conmutador. VCT es la técnica de conmutación más utilizada en las redes fuera del chip debido a que el tamaño de la cola no es un problema crítico.

Wormhole(WH) A diferencia de las técnicas anteriores, esta técnica de conmutación de mensaje no requiere espacio suficiente en la cola para almacenar todo el mensaje, es decir, en WH las colas únicamente pueden almacenar unos pocos flits. El tamaño de la cola depende del tiempo de vuelo del enlace (*roundtrip time delay*; RTT). Al igual que VCT, WH empieza a procesar el mensaje una vez se ha recibido la cabecera. Sin embargo, a diferencia de VCT, con la técnica de conmutación WH no es necesario que en la cola de entrada se almacene todo el mensaje. En el caso de que se bloquee un mensaje en la red, el mensaje se almacena a lo largo de la ruta en varios conmutadores. La mayor ventaja de WH es el bajo requerimiento de recursos de almacenamiento. Sin embargo el gran inconveniente es que puede llevar a una gran contención debido a que bloquea muchos recursos a través de la red de interconexión.

Así como las técnicas de conmutación son las encargadas de determinar como fluyen los datos, las técnicas de control de flujo son las encargadas de administrar el envío de las unidades información a través de la red de interconexión entre los conmutadores. Como se ha comentado, las colas son recursos donde se almacenan temporalmente los *flits*. Sin embargo, estas colas no son infinitas. El control de flujo es el encargado de determinar cuando un *flit* puede ser transmitido por la red, sin que esta transmisión produzca una pérdida de la información debido a que no se ha podido almacenar el *flit* porque las colas están llenas. Los mecanismos de control de flujo más usados son:

- **Stop&Go**. *Stop&Go* determina dos umbrales dentro de la cola. Cada umbral hace referencia a una posición dentro de la cola. Si la cola se llena lo suficiente como para llegar a la posición de *Stop*, el conmutador envía una señal al conmutador que le está enviando por ese puerto para decirle que pare de mandarle *flits* hasta que reciba la señal de *Go*. Esta señal será enviada cuando se vaya vaciando la cola y el número de mensajes en la cola descienda hasta la posición de *Go*.

- **Créditos.** En este protocolo de control de flujo el conmutador tiene asignado un número de créditos a un puerto de salida, cuando este manda un *flit*, decrementa el número de créditos en uno, solo pudiendo enviar si el número de créditos es positivo. Una vez el conmutador receptor ha vaciado la posición del *flit*, éste manda un mensaje al nodo emisor para que incremente el valor de créditos en uno.

- **ACK/NACK**. Cuando un *flit* llega al puerto de entrada, si existe espacio suficiente en alguna de las colas, el *flit* se almacena y se envía hacía atrás una

señal de reconocimiento (*acknowledgement signal*; ACK). Si, por el contrario, no existe espacio suficiente, el *flit* se descarta y se envía una señal negativa de reconocimiento (NACK). El *flit* no puede ser eliminado del conmutador emisor hasta que la señal ACK haya sido recibida.

Con respecto al *crossbar* podemos diferenciar dos estrategias de comutación que pueden ser implementadas en el conmutador.

- **Flit-level Crossbar Switching**. Ésta consiste en la mejora de la utilización de la cola permitiendo que el conmutador multiplexe los *flits* de diferentes mensajes para avanzar a través del *crossbar* con la misma dirección, es decir, el mismo puerto de salida, pero mapeando los distintos mensajes en canales virtuales diferentes. La figura 2.3 muestra un sencillo ejemplo de como por el puerto de salida 1 se multiplexan los *flits* de los mensajes almacenados en las colas de entrada.

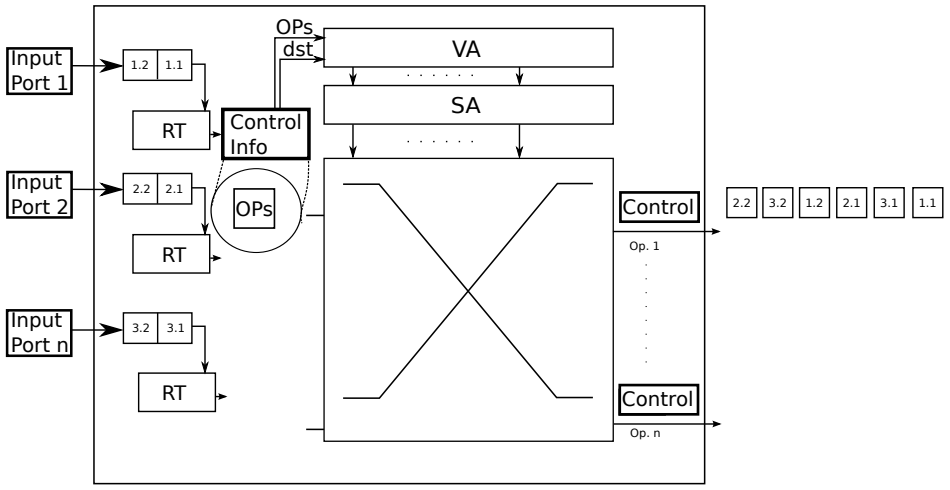


Figura 2.3: Flit-level Crossbar Switching.

- **Packet-level Crossbar Switching** y consiste en evitar que el conmutador multiplexe *flits* de diferentes mensajes al mismo puerto de salida. En esta estrategia, cuando la cabecera del mensaje accede al *crossbar*, el resto del mensaje se envía sin interrupción conservando la conexión al *crossbar*. La figura 2.4 muestra un sencillo ejemplo de cómo por el puerto de salida 1 los mensajes son transmitidos sin interrupción y sin que otros *flits* puedan intercalarse por el puerto de salida.

Flit-level crossbar está concebido para conmutación de tipo *wormhole*, mientras que *packet-level Crossbar Switching* está concebido para VCT. Sin embargo, ambas estrategias pueden usar cualquier tipo de mecanismo de conmutación. No obstante,

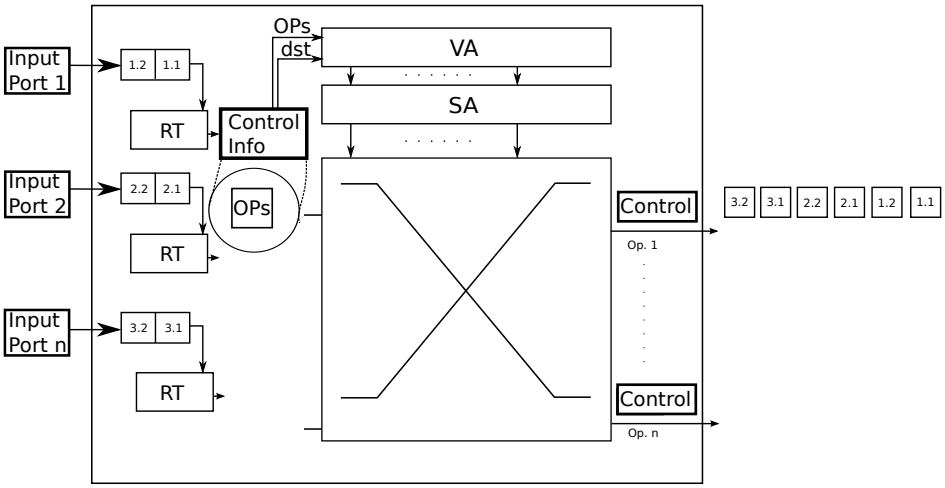


Figura 2.4: Packet-level Crossbar Switching.

el mecanismo de control debe diseñarse teniendo en cuenta si se utiliza *flit-level* o *packet-level Crossbar Switching*.

La topología de las NoCs definen la organización física de los nodos y las conexiones entre ellos, es decir, define el patrón de interconexión de los dispositivos de una red de interconexión. La topología tiene un gran impacto tanto en las prestaciones como en el coste. Las topologías se pueden diferenciar en dos grandes grupos: redes de medio compartidos y redes conmutadas.

- **Redes de medio compartidos**. Estas topologías son aquellas en la que todos los nodos comparten un medio de transmisión. Las limitaciones de estas topologías son baja flexibilidad, únicamente un nodo puede volcar información a la red de interconexión y un limitado ancho de banda. Por otra parte, la gran ventaja de estas redes es la simplicidad y el bajo coste. Además, permiten una fácil implementación de las transmisiones de mensajes a varios destinos (*Multicast*) o a todos los destinos (*Broadcast*).La Figura 2.5 muestra un ejemplo de este tipo de red.

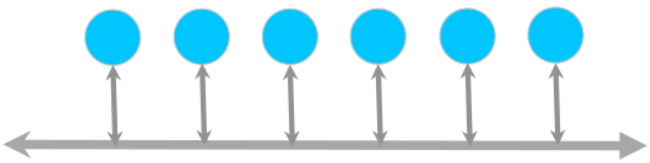


Figura 2.5: Red de medio compartido.

- **Redes Conmutadas**. En estas topologías se utilizan enlaces punto a punto y elementos de conmutación (conmutadores). Se caracterizan por tener alta flexibilidad y disponibilidad y un mayor ancho de banda. Este tipo de redes se pueden clasificar en tres tipos diferentes:

  – **Redes directas**.Cada nodo se conecta por medio de enlaces punto a punto. Algunos ejemplos de este tipo de red son las topologías tipo malla (Figura 2.6), toro o hipercubo. En algunas topologías de este tipo, como por ejemplo el toro, existen enlaces que conectan los nodos situados en los extremos de la topología, estos enlaces se denominan *wraparound link*.

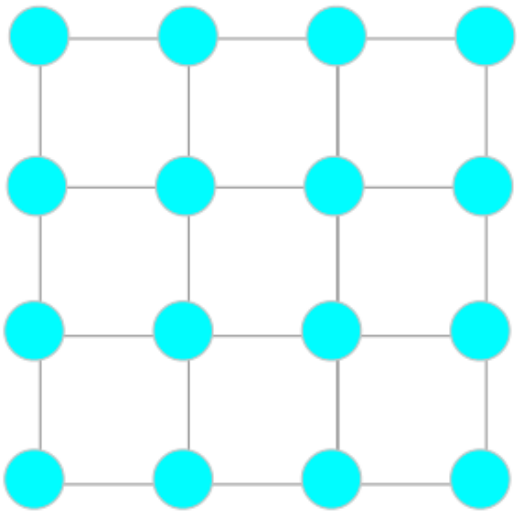


Figura 2.6: Red conmutada directa.

  – **Redes indirectas**. Cada nodo se conecta a un conmutador y los conmutadores tienen enlaces punto a punto con otros conmutadores. Un claro ejemplo de este tipo de red son las topologías *Butterfly* (Figura 2.7), *Cube* o *Baseline*.

  – **Redes hibridas** Las redes híbridas combinan los dos tipos de red anteriores, por tanto aprovechando las ventajas de cada una de ellas. En este trabajo asumimos el uso tanto de mallas 2D como toros 2D.

  En la actualidad, la topología más utilizada en redes en el chip es la Malla 2D que se trata de una topología directa.

Los algoritmos de encaminamiento son los responsables de decidir por qué puerto de salida debe encaminarse los mensajes para que se efectúe un encaminamiento efectivo y

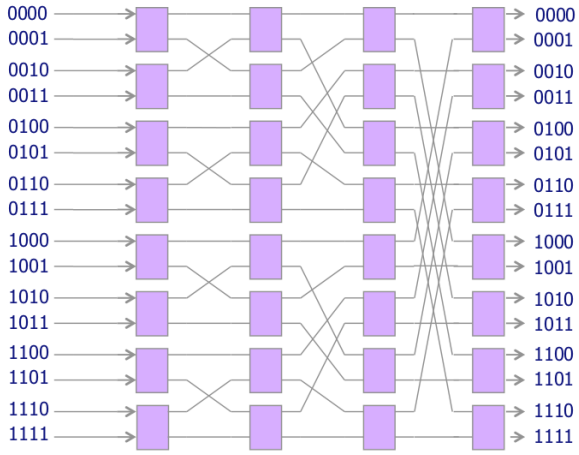


Figura 2.7: Red conmutada indirecta.

correcto. En las NoCs, a diferencia de algunas redes fuera del chip, es un requerimiento que no haya pérdida de información, es decir, pérdida de mensajes. Sin embargo, aún existiendo caminos disponibles, existen situaciones que se deben evitar. Estas situaciones son las siguientes:

- **Bloqueo (Deadlock)**. El *deadlock* ocurre cuando un conjunto de mensajes no puede avanzar por que los recursos que solicita están ya ocupados por los mismos mensajes, los mensajes están bloqueados en un ciclo.

- **Liveblock**. Esta situación se produce cuando el mensaje no se bloquea, pero debido al encaminamiento de ruta no mínima el mensaje siempre se aleja de su destino y nunca alcanza el destino o el enlace necesario, porque otro mensaje lo está utilizando.

- **Starvation**. Esta situación aparece cuando un mensaje está permanentemente bloqueado en el recurso y no puede avanzar, debido a que cuando solicita el siguiente recurso siempre hay menos prioridad. Esto puede ser resultado de un arbitraje incorrecto.

Existen dos formas principalmente de implementar los algoritmos de encaminamiento: Algoritmos de encaminamiento basados en tablas y algoritmos de encaminamiento basados en lógica.

- **Algoritmos de encaminamiento basados en tablas**. Estas tablas de encaminamiento están formadas por una estructura de filas que contiene el puerto de salida para un destino concreto. Por lo tanto, dado el destino para un mensaje

determinado, existe un circuito lógico asociado que decodifica esta información, y accede a la tabla de encaminamiento para encontrar el puerto de salida asociado a ese destino. La manera más convencional para implementar estas tablas es utilizar las estructuras de memoria. La ventaja del encaminamiento basado en tablas es la flexibilidad, ya que la información de las decisiones de encaminamiento almacenados en las tablas, puede ser la respuesta de los algoritmos de encaminamiento mucho más complejos. Por otra parte, la implementación en tablas de encaminamiento sufre de escalabilidad, área, consumo de energía y problemas de latencia.

- **Algoritmos de encaminamiento basados en lógica**. Este tipo de encaminamiento es el resultado de trasladar una función lógica o aritmética del algoritmo de encaminamiento al circuito equivalente dentro del conmutador. Cuando la cabecera del mensaje se decodifica, se calcula el puerto de salida en el hardware que representa la función de encaminamiento. Este tipo de algoritmos obtienen buenos resultados en términos de latencia, área, y consumo de energía. El inconveniente de este tipo de encaminamiento es la falta de flexibilidad. Un ejemplo de este tipo de encaminamiento XY, el cual mediante la función lógica primeramente calcula si ha el destino del mensaje requiere desplazamiento en horizontal, y una vez el mensaje ha llegado a la columna del destino se transmite verticalmente. El encaminamiento XY es un encaminamiento determinista, ya que para alcanzar un destino desde un emisor los mensajes siempre siguen la misma ruta.

Además, de estos encaminamientos deterministas, encaminamientos en que un mensaje enviado desde un nodo origen a un destino siempre sigue la misma ruta, existen los algoritmos de encaminamiento adaptativos. Estos algoritmos aportan una mayor flexibilidad a la hora de encaminar los mensajes, pero pueden producir los problemas descritos anteriormente. Además para implementar estos algoritmos se requiere más de un canal virtual, debido a que se diferencian los canales entre canales adaptativos y canales de escape. Los canales adaptativos son aquellos por los cuales un mensaje puede ser encaminado sin ningún tipo de restricción. Por otra parte, los canales de escape sí que siguen un algoritmo de encaminamiento determinado y libre de bloqueo. Esto permite que en la red no se produzca un *deadlock*.

2.2 Trabajo Relacionado.

En esta sección se describen los trabajos previos relevantes sobre los problemas que se han propuesto en el capítulo 1. En concreto, se describen mecanismos de control de flujo, algoritmos adaptativos en NoCs y técnicas para evitar la congestión en la red.

2.2.1 Mecanismos de Control de Flujo y Encaminamiento Adaptativo.

En los últimos años se han presentado muchos artículos sobre control de flujo y encaminamiento en NoCs. Aquí, se introducen algunos de ellos. Tang en [TL09] propone un mecanismo de control de flujo en el que se limita la inyección de mensajes en la red dinámicamente. Nousias en [NA06] propone una estrategia con un ratio de control adaptativo con conmutación *wormhole* utilizando canales virtuales. En esta propuesta, cuando cambia la contención, el nodo destino manda una señal al nodo fuente para que regule la inyección de forma proporcional a la congestión existente.

Avasare en [ANM+05] propone un mecanismo de control de flujo punto a punto centralizado con conmutación de mensajes. Este mecanismo requiere dos redes, una de control y una de transmisión de datos. En [PD00], se presenta *flit reservation control*. En esta estrategia de control de flujo, un flit de control avanza por la red reservando los canales virtuales. Una vez se ha alcanzado el destino, el nodo fuente envía el mensaje al nodo destino. Ambos mecanismos necesitan una red paralela de control.

Glass y Ni en [GN92] proponen un modelo que diseña un algoritmo de encaminamiento parcialmente adaptativo en mallas. Se utiliza *west-first routing* en una malla 2D, y si es necesario adaptativamente sur, norte y este. Wu [Wu03] propone un algoritmo tolerante a fallos basado en *odd-even turn model* para mallas 2D.

Dally y Seitz en [DS12] presentan las condiciones necesarias y suficientes para los algoritmos libres de bloqueo en redes de interconexión. Muchos algoritmos han sido propuestos para mallas y toros [Chi00, XL08, SDGT03]. Duato [Dua95] propone las condiciones necesarias y suficientes para implementar algoritmos adaptativos libres de bloqueos en redes con conmutación *wormhole*. Las metodologías para el diseño de algoritmos adaptativos libres de bloqueos se presentan en [DP01]. En [Pue01] se expone un conmutador adaptativo con control de flujo de burbuja para redes con conmutación VCT basadas en el protocolo de Duato. Este conmutador requiere un canal adaptativo y uno de escape por el cual se aplica encaminamiento *Dimension-Order-Routing* (DOR). Para evitar el bloqueo en el canal adaptativo se diseña una

función de control de flujo.

Los mecanismos de control de flujo anteriores necesitan más información que nuestra propuesta, algunos de ellos necesitando una red adicional. Por lo tanto, estas propuestas utilizan más recursos. Además, nuestra propuesta tampoco limita el flujo del tráfico inyectado a la red, como pasa en [TL09]. Finalmente, SUR únicamente necesita dos canales virtuales para ofrecer un algoritmo de encaminamiento totalmente adaptativo. Estos canales no están clasificados en canal de escape y adaptativo, lo cual permite una utilización de las colas más equilibrada. Además, ninguna de las propuestas anteriores se centra en balancear la utilización de los recursos del conmutador con un codiseño de mecanismos de control de flujo y algoritmo de encaminamiento. (TBFC + SUR)

2.2.2 Técnicas de Alivio de la Congestión.

También se han desarrollado muchos estudios para el control de la congestión en la red. Seguidamente, introducimos algunos trabajos relevantes sobre técnicas para el control de la congestión. En el campo de trabajo del control de la congestión en la red de interconexión existen cuatro perspectivas diferentes.

La primera de ellas trata de desarrollar técnicas que garanticen que la congestión nunca se pueda crear en la red. Nan en [YTL87] propone la construcción de un árbol mediante software para combinar los mensajes a memoria que tienen como destino el *hotspot*. Esta técnica evita la saturación en la red mediante la distribución de los mensajes por el árbol software. Los nodos de dicho árbol están situados en los módulos de memoria. El inconveniente de esta aproximación es que tiene baja productividad y utilización.

Ferrer en [FBR+09] presenta una técnica basada en la segunda aproximación a este problema. Ésta se basa en detectar congestión, notificarlo a la fuente y eliminar la congestión mediante limitación de la inyección (*injection throttling*). Aunque esta técnica es efectiva en ciertos escenarios, su efectividad depende del ancho de banda de la red y el tamaño de ésta ya que se basa en un bucle de control cerrado.

El tercer punto de vista para gestionar la congestión es atacar directamente el efecto de dicha congestión: el *Head of Line blocking* (HoL blocking) [XZZY10]. En este caso, los mecanismos detectan la congestión en la red de interconexión y dinámicamente se asignan nuevas colas para mapear en ellas los mensajes que están causando la congestión, y por lo tanto, quitar estos mensajes de las colas normales permitiendo así que estos mensajes puedan circular por la red. Un claro ejemplo de esta aproximación es el mecanismo RECN [DJF+05]. Sin embargo en este caso, el sobrecoste de la imple-

mentación no es insignificante. Otro ejemplo de esta aproximación para gestionar la congestión en la red es ICARO [EFG14] y BAHIA [EFG13], estos mecanismos detectan el tráfico congestionado en la red, una vez detectado que el tráfico a un destino esta congestionado lo notifica y finalmente lo aísla de la red pasando este tráfico a una red virtual especial.

Finalmente, la última aproximación para solucionar la congestión en la red de interconexión es el uso de algoritmos adaptativos para evitar los *hotspots*. Esta aproximación se utiliza en [GGK08]. Sin embargo, la efectividad de este tipo de aproximación depende de la intensidad de la congestión en la red. Como se ha comentado anteriormente, si la intensidad de la congestión es elevada esta aproximación provocará el efecto contrario haciendo que la congestión se ramifique por toda la red, empeorando la situación.

En [MAW99] se propone la solución de utilizar canales virtuales de salida (VOQ), para aliviar la congestión haciendo que todos los mensajes a un mismo destino salgan por el mismo canal de salida. sin embargo, este mecanismo no soporta el uso de encaminamiento adaptativo, por lo cual no se beneficia de la alta productividad conseguida por estos algoritmos de encaminamiento.

Estas son las cuatro aproximaciones en la que se centran los estudios para intentar solucionar el problema de la congestión en la red de interconexión, sin embargo, nuestro nuevo filtro EPC es un mecanismo complementario para estas soluciones.

2.3 Entorno de Simulación.

El simulador utilizado en este proyecto ha sido gMemNoCsim. Éste es un conjunto de herramientas que nos permiten el diseño y modelado de redes en el chip, así como analizar su rendimiento. Las herramientas que lo componen son:

gMemNoCsim: gMemNocSim es un simulador de redes en el chip implementado en C/C++. Es un simulador dirigido por eventos que van ocurriendo en el tiempo. La ejecución de estos eventos implica la generación de nuevos eventos relacionados con el primero, que finalmente permiten el modelado de la red. De esta forma el simulador inicializa estructuras, datos estadísticos y programa los eventos iniciales (la inyección de los primeros mensajes) en el simulador.

CEF_Parser: este es el parser para configurar los archivos que posteriormente se simularán en la herramienta anterior. Genera y lee los ficheros que describen la con-

figuración de la red.

GnoCgui: se trata de la interfaz gráfica para la lectura y escritura de los archivos CEF de configuración, además desde esta herramienta se pueden lanzar las simulaciones y generar las gráficas con los resultados de las simulaciones.

El simulador permite definir una gran cantidad de parámetros. Estos parámetros son: Primeramente el control de flujo, créditos y stop&go, junto con el tamaño de las colas, el número de mensajes a simular y los umbrales de stop&go. GMemNoCSim también permite definir la distribución de tráfico, que nodos van a transmitir, el tamaño de los mensajes cortos y largos y la cantidad de cada uno de estos que se enviarán en la simulación. Además, el simulador también permite definir la topología a simular, el número de nodos, puertos por conmutador, y número de conmutadores.

Finalmente, se han tenido que hacer un par de modificaciones generales al simulador para implementar las técnicas propuestas en esta tesina. El primer cambio significativo se trata de adaptar el simulador para que permita realizar *Flit-level Crossbar switching*, ya que solo permitía *Packet-level Crossbar switching*, para hacer esto ha sido necesario implementar un árbitro de tres fases (Request-Accept-Grant). Otro cambio general implementado para la técnica SUR ha sido modificar los interfaces de red para que sean ideales y tengan un VC por destino.

Chapter 3

Descripción de los Mecanismos.

En esta sección describimos TBFC y SUR, los cuales permiten el correcto balanceado en la utilización de las colas de los conmutadores. Posteriormente, describimos el mecanismo del filtro EPC. En el siguiente capítulo describimos los resultados obtenidos.

3.1 TBFC: Type Based Flow Control.

En las dos secciones siguientes describimos nuestro mecanismo de control de flujo para ambas estrategias de conmutación explicadas en el capitulo 2.1.

3.1.1 TBFC con Flit-Level Crossbar Switching.

La Figura 3.1 muestra la implementación tradicional del mecanismo de control de flujo basado en créditos para una pareja de puertos de entrada-salida. Asumimos *flit-level switching*. En cada puerto de salida, el conmutador necesita cierta información de control. Por cada canal virtual se necesita: un campo para el número de créditos disponibles (*CRED*), un campo para determinar si el canal está siendo utilizado (*USED*), el puerto de entrada y el canal virtual del puerto de entrada que ha sido asignado al VC de salida (se establece un enlace entre el puerto de entrada y el VC concedido).

Cuando se encamina la cabecera de un mensaje, el conmutador envía una petición para obtener el puerto de salida indicado por el algoritmo de encaminamiento. En este puerto, el *virtual channel allocator (VA)* comprueba si hay algún VC libre y con créditos suficientes en el siguiente conmutador para todo el mensaje (asumiendo VCT). Entonces, el conmutador arbitra (en *modo round-robin*) entre todas las peticiones y asigna el VC a la petición ganadora. Éste guarda el puerto de entrada y el VC ganador

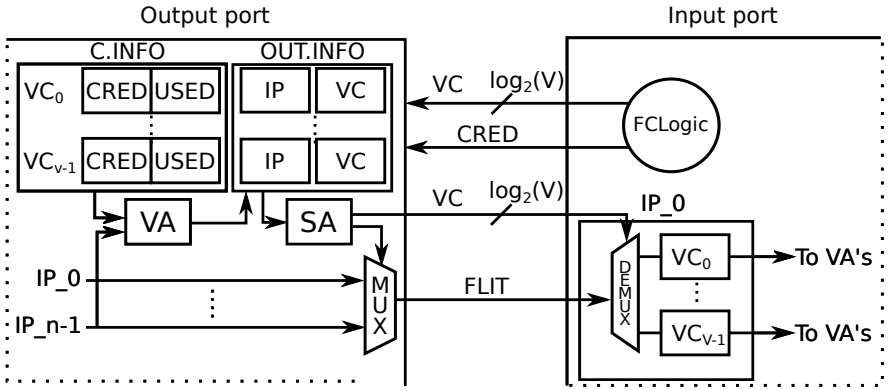


Figura 3.1: Control de flujo tradicional basado en créditos asumiendo *flit-level crossbar switching*.

en la estructura de información de control del VC asociado. El conmutador decrementa los créditos disponibles en la información de control asociada al VC.

En la etapa *Switch Allocation (SA)*, el arbitro selecciona el puerto de entrada que enviará el flit a través del puerto de salida en el siguiente ciclo. SA selecciona este puerto de entrada entre todos los que han sido asignados a este puerto de salida en la etapa de VA. Para implementar *flit-level crossbar switching*, el árbitro rota la prioridad de los puertos de entrada cada vez que un flit accede al *crossbar*, asignando la menor prioridad al puerto de entrada que acaba de enviar el flit. El conmutador envía el flit junto con el VC ID al siguiente conmutador. El siguiente conmutador utiliza este VC ID para multiplexar y almacenar el flit en el VC correspondiente. Cuando la cola de un mensaje se envía, el VC se libera y puede ser asignado a un nuevo mensaje.

Cuando se envía un flit, el modulo *Flow Control Logic* (*FCLogic*) del puerto de entrada donde está almacenado envía un crédito al conmutador anterior. Para hacer esto, el conmutador necesita al menos $log_2\ (V)+1$ cables para indicar el VC que va a recibir el crédito (señal *VC*), donde V es el número de canales virtuales en cada puerto de entrada. El conmutador también envía la señal de control *CRED*. El conmutador anterior, al recibir esta señal, incrementa el contador de créditos asociado al VC correspondiente.

La Figura 3.2 muestra TBFC cuando aplicamos *flit-level crossbar switching*. La primera diferencia entre el mecanismo de control tradicional y el de TBFC es la estructura de información del control de flujo. TBFC añade dos nuevos campos por puerto de salida: el campo *FREE*, el cual tiene una cuenta del número de canales virtuales disponibles en el siguiente puerto de entrada, y el campo *TYPE*, el cual contiene el número de mensajes etiquetados de cierta forma almacenados en el siguiente puerto de entrada (en la siguiente sección describiremos el uso de este campo por el algoritmo de

encaminamiento). Además, por cada VC, se almacena un contador asociado al número de créditos del VC en el puerto de entrada. Sin embargo, el campo *USED* se elimina.

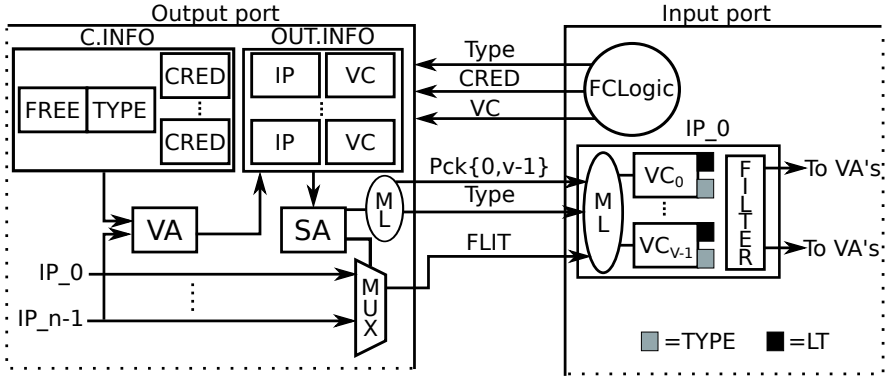


Figura 3.2: Control de flujo TBFC asumiendo flit-level crossbar switching.

Al contrario que en el mecanismo de control de flujo base, las reglas (en la etapa VA) para asignar un VC a una petición entrante son diferentes. Ahora, la etapa VA comprueba únicamente el número de VCs libres y el número de mensajes marcados (campo *TYPE*) (describimos estas reglas más en detalle en la siguiente sección). Cuando una petición obtiene el VC de salida, el puerto de entrada y el VC asociados a dicha petición se almacenan en la estructura de información de control del VC de salida. El número de *FREE* VCs se reduce en uno y si el mensaje enviado es etiquetado como el tipo que se contabiliza en el campo *TYPE*, entonces el campo *TYPE* se incrementa en uno.

En la etapa SA, el conmutador selecciona el puerto de entrada que enviará un *flit* a través del puerto de salida al siguiente conmutador en el siguiente ciclo. En la etapa *crossbar*, el conmutador no envía el VC seleccionado como ocurre en el modelo tradicional. En su lugar, éste realiza una asignación (bloque *ML*), $packet \rightarrow ID$ que asigna un identificador al mensaje. Junto con el *flit* de cabecera se envía el tipo del mensaje. El identificador, el tipo del mensaje y el flit se envían a través del enlace al siguiente conmutador. Todos los *flits* del mismo mensaje usan el mismo identificador y solo el flit de cabecera contiene la información del tipo.

Cuando el siguiente conmutador recibe el flit de cabecera, el tipo de mensaje y el identificador, se realiza una nueva asignación (bloque *ML*). En este caso, el mapeado es $ID \rightarrow VC$, asignando así al nuevo mensaje un VC libre. Después de la llegada del flit de cabecera, todos los *flits* que lleguen con el mismo identificador se almacenan en el mismo VC a través del ML.

Cada VC de entrada tiene asociado un bit, denominado *Last token* (LT). Cuando un *flit* de cabecera llega y es asignado en un VC libre, este VC pone su bit LT a '1'

y el bit LT del otro VC se configura a '0'. [1]. Este campo se utiliza para garantizar entrega en orden. En el caso de que ambos VCs en el puerto de entrada tengan un *flit* de cabecera y su destino fuera el mismo, el más viejo (el que tenga LT igual a 0) es el único que puede acceder al VA. En otro caso, ambos pueden acceder al VA. Nótese que si se implementa encaminamiento adaptativo, éste permite entrega fuera de orden, por lo que el bit LT y la lógica asociada a éste pueden eliminarse. Éste es el caso para nuestro algoritmo. Además, cada VC de entrada también contiene un bit para el tipo *TYPE* que indica si el mensaje almacenado está marcado o no. Este bit se actualiza con la información de tipo recibida con el flit de cabecera.

Siempre que se transmite un flit de cabecera al siguiente conmutador, el bit *TYPE* se transmite al conmutador anterior. Una vez recibido, el conmutador anterior decrementa el contador *TYPE* de la estructura de información de control. Además, el campo *FREE* se incrementa en uno. Nótese también que el campo *CRED* se sigue utilizando en TBFC. Esto es necesario si se asume *flit-level crossbar switching* para evitar enviar un *flit* a una cola llena.

La Figura 3.3 muestra un ejemplo de la asignación en la cola de entrada. Asúmase que ambos VCs de entrada están vacíos. También están a cero las variables locales para asignar un VC de entrada a un identificador. En el tiempo t el *flit* de cabecera del mensaje A (A_H) llega al puerto de entrada con el identificador igual a 1. Entonces, el bloque ML asigna el flit a VC_0, y guarda el identificador en la variable local. En t+1 llega un *flit* del mensaje A con identificador 1. El bloque ML sabe donde lo tiene que asignar y almacena el *flit* en VC_0. En el siguiente ciclo, el *flit* de cabecera del mensaje B (B_H) llega al puerto de entrada con identificador 0, el bloque ML asigna el nuevo mensaje a VC_1 y guarda el identificador en la variable local. En t+5 llega un nuevo *flit* de B (B_B), pero en este ciclo, el *flit* de cabecera de A atraviesa el *crossbar* y deja el puerto de entrada. El bloque ML limpia el identificador de la variable local. Si en este momento llega un mensaje nuevo, este VC puede volver a asignarse. También, el FCLogic envía hacia atrás, al conmutador anterior, la información del tipo de mensaje usando la información de control *TYPE*. Finalmente, en t+6 una nueva cabecera, en este caso del mensaje C (C_H), alcanza el puerto de entrada y el bloque ML asigna este nuevo mensaje a VC_0, guardando en la variable local de este VC el identificador del mensaje.

Nótese que los mensajes A y C comparten el mismo VC. Sin embargo, el control de flujo está implementado a nivel de mensaje, ésto significa que cuando la cabecera

[1] Si el conmutador tiene más de 2 VCs, el campo LT necesita log_2 *(V)* bits y se actualiza siguiendo un algoritmo similar al utilizado en las caches con política de remplazo *Least Recently Used (LRU).*

de un mensaje se envía este VC está disponible (aunque no esté completamente vacío). Dado que la longitud del mensaje es igual a la longitud del VC, garantiza que un mensaje no se bloqueará a mitad de una cola, ya que el mensaje anterior dejará dicha cola en un momento determinado. De hecho, si un mensaje se bloquea, se almacena completamente en una cola del puerto de entrada. Esto preserva la regla que conforma VCT.

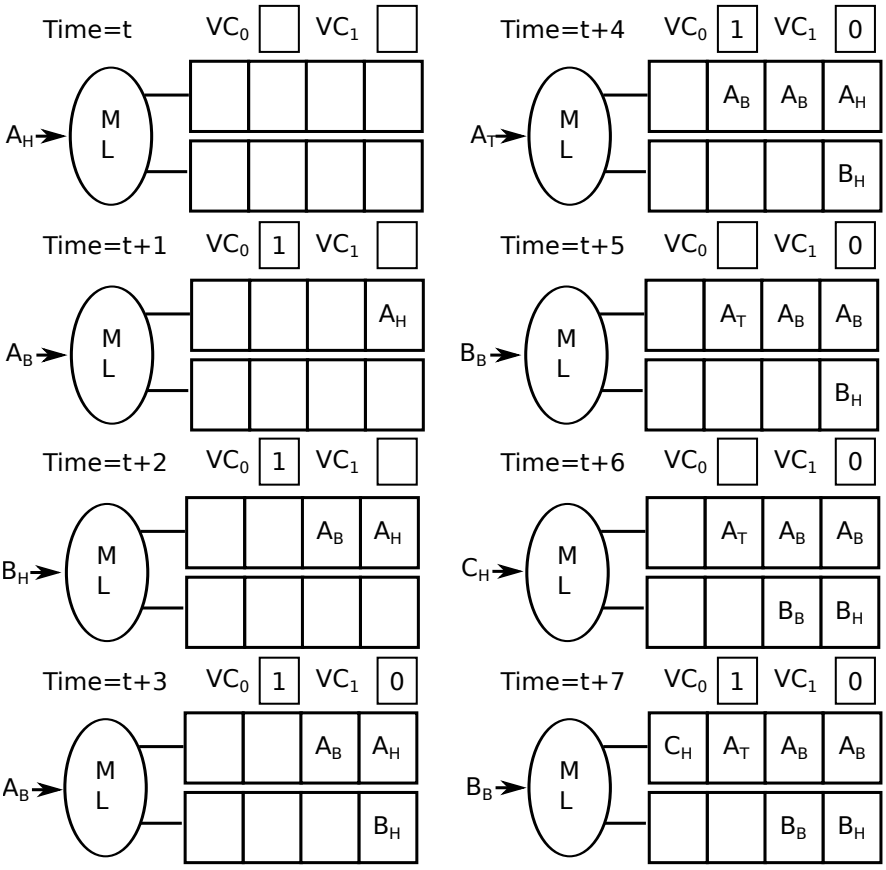


Figura 3.3: Ejemplo de la lógica del conmutador en el puerto de entrada.

3.1.2 TBFC con Packet-Level Crossbar Switching.

Ahora, nos centraremos en el mecanismo TBFC con *packet-level crossbar switching*. Nótese que en este caso los mensajes no se multiplexan en el *crossbar*. Este hecho, junto con VCT *switching* garantiza que la tasa de recepción y transmisión de los mensajes en el puerto de entrada será igual. Ésto significa que siempre que una cabecera gane el acceso al *crossbar*, todo el mensaje puede ser transmitido y por tanto no se detendrá hasta que todo el mensaje haya sido enviado al siguiente conmutador. Esto simplifica mucho el mecanismo TBFC, como mostraremos a continuación.

La Figura 3.4 muestra el mecanismo TBFC con *packet-level crossbar switching*. Lo primero que se percibe es la simplificación de la estructura de control. Ahora no necesitamos la información sobre los créditos. Solo se necesita almacenar el puerto de entrada y el VC al que se ha asignado el puerto de salida. No obstante, los campos *FREE* y *TYPE* se siguen usando. El bloque de mapeado lógico también se elimina ya que cuando un mensaje consigue el acceso al *crossbar*, el mensaje será enviado ininterrumpidamente.

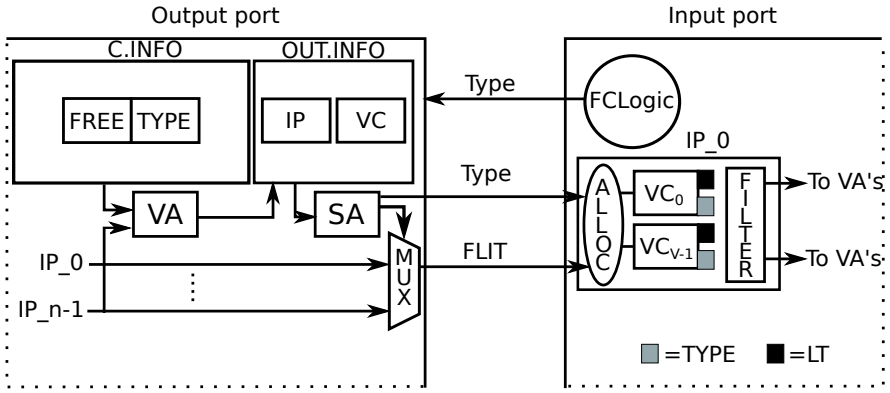


Figura 3.4: Control de flujo TBFC asumiendo *packet-level crossbar switching*.

La etapa VA se modifica levemente, la única variación es que en esta etapa solo se tienen en cuenta el número de VCs libres (campo *FREE*) y el número de mensajes etiquetados en el siguiente conmutador (campo *TYPE*). La etapa SA también se simplifica debido a que no se tienen que multiplexar *flits* en el puerto de salida. SA solo necesita arbitrar entre los mensajes que compiten por conseguir el puerto de salida y debe guardar la prioridad del puerto hasta que la cola del mensaje haya sido enviada al siguiente conmutador. Esto evita la multiplexación de los mensajes.

En el puerto de entrada del siguiente conmutador la lógica también se simplifica. Se elimina el bloque lógico *ML*. Además, cuando un mensaje sale del conmutador el modulo *FCLogic* solo envía el tipo de mensaje al conmutador anterior. El bit LT se sigue usando si se necesita entrega en orden. El campo del tipo de mensaje también es necesario para saber si el mensaje está etiquetado o no.

3.2 Algoritmo de encaminamiento Safe/Unsafe.

En esta sección presentamos el nuevo algoritmo de encaminamiento adaptativo para el mecanismo TBFC. Cada puerto de entrada contiene dos VCs, mientras que cada VC tiene el suficiente espacio en la cola para almacenar todo el mensaje. El algoritmo SUR es totalmente adaptativo y para evitar bloqueos (*deadlocks*) utiliza una ruta de

escape. El encaminamiento utilizado para implementar la ruta de escape es *Dimension Order Routing* (DOR). SUR puede trabajar usando tanto conmutadores con *flit-level crossbar switching* como con *packet-level crossbar switching*. SUR funciona en mallas de n-dimensiones y toros de n-dimensiones.

TBFC permite el etiquetado de mensajes y utiliza esta información en la etapa de encaminamiento. En este caso, el algoritmo SUR etiqueta los mensajes como *safe* o *unsafe*. Los mensajes se etiquetan y se mandan al siguiente conmutador de la siguiente forma:

- En una red n-dimensional un mensaje es enviado y almacenado en el siguiente conmutador como mensaje *safe* si el próximo salto forma parte del camino que se realizaría en el algoritmo de encaminamiento base (DOR). En cualquier otro caso, el mensaje es etiquetado como *unsafe*.

- En un toro de n-dimensiones un mensaje es enviado y almacenado en el siguiente conmutador como mensaje de tipo *safe* si cumple alguna de las siguientes condiciones:

  - El siguiente salto del mensaje en la red tiene que atravesar un *wraparound link* a través de la dimensión d, y el mensaje no necesita atravesar otro *wraparound link* con una dimensión menor a d para alcanzar su destino.
  - El mensaje no necesita atravesar ningún *wraparound link* desde el conmutador actual hasta el conmutador destino y el siguiente salto forma parte del algoritmo de encaminamiento base (DOR).

  Si alguna de estas dos condiciones no se cumplen, entonces, el mensaje es transmitido y etiquetado como mensaje *unsafe*.

Con esta clasificación, el algoritmo de encaminamiento puede decidir qué puertos de salida son elegibles para un mensaje. Los puertos que conforman un camino mínimo hacia el destino pueden ser elegidos. Los mensajes tipo *safe* pueden ser encaminados sin ninguna restricción mientras que los mensajes tipo *unsafe* pueden encaminarse si cumplen ciertas condiciones.

Para ayudar a este algoritmo de encaminamiento se define la función *check port*, que puede utilizarse en mallas y toros. El algoritmo 1 muestra la función *check-port*(f,s). Esta función evita que se llene un puerto de entrada únicamente con mensajes de tipo *unsafe*. Para un puerto de entrada dado se comprueba el número de VC libres (f) y el número de mensajes *safe* (s) de la siguiente forma:

- $f > 1$, el mensaje puede ser enviado porque hay más de un VC libre en el puerto de entrada del siguiente conmutador.
- $f = 1$ & $s > 0$, el mensaje puede ser enviado porque hay al menos un mensaje tipo *safe* en el puerto de entrada del siguiente conmutador.
- $f = 1$ & $s = 0$, el mensaje puede ser enviado solo si el mensaje será etiquetado al enviarse al siguiente conmutador como *safe*; en cualquier otro caso, el mensaje se bloquea o toma otro puerto de salida.
- $f = 0$, el mensaje se bloquea o toma otro puerto de salida.

Algoritmo 1 check-port(f,s)

Entradas:
Número de VCs libres en el siguiente conmutador, f;
Número de mensajes *safe* en el siguiente conmutador, s;
Salida:
Si el mensaje puede ser enviado al siguiente conmutador;

```
1: if f > 1 then
2:    return true;
3: end if
4: if f = 1 y s ≥ 1 then
5:    return true;
6: end if
7: if f = 1 y s = 0 then
8:    y el mensaje será etiquetado como safe en el siguiente conmutador
9:    return true;
10: end if
11: return false;
```

En Alg 2 se muestra el algoritmo de encaminamiento totalmente adaptativo propuesto para mallas 2-D donde f_i+ representa el número de VC libres y s_i+ representa el número de mensajes *safe* en el puerto de entrada del siguiente conmutador conectado al conmutador actual C a través de la dimensión i en la dirección positiva. De forma similar f_i- representa el número de VC y s_i- el número de mensajes *safe* en el puerto de entrada en la dirección negativa. El algoritmo toma como entradas cuatro valores: las coordenadas del conmutador actual y del conmutador destino, el número de colas libres y el número de mensajes *safe* en el puerto de entrada vecino. El conjunto de canales disponibles y de puertos de salida son inicializados a $\emptyset$ y *null*, respectivamente.

Si el conmutador actual es el destino, se selecciona el canal interno y se consume el mensaje. En los otros casos, si el *offset* a través de la dimensión i (dimensiones 1 y 2

en Alg. 2) es mayor que 0 y la función *check-port(f_i+,s_i+)* retorna un valor positivo, entonces el mensaje puede ser enviado en el canal c_i+. Si el *offset* en esa dimensión i es menor que 0 y la función *check-port(f_i-,s_i-)* devuelve un valor positivo, el mensaje puede ser enviado por el canal c_i-. La función *check-port(f_i,s_i)* permite agregar el canal c_i+ o c_i- a S si el mensaje puede avanzar en la dimensión i.

Algoritmo 2 safe-unsafe-2D-mallas

Entradas:

Coordenadas del conmutador actual $C : (c_1, c_2)$,
Coordenadas del conmutador destino $D : (d_1, d_2)$,
Colas libres: (f_1-,f_1+, f_2-, f_2+),
mensajes *safe* : (s_1-, s_1+, s_2-, s_2+);

Salidas:

Canal de salida seleccionado;

```
1: S=0;ch=null;
2: if C == D1 then
3:    ch=interno; return true;
4: end if
5: for i == 1 to 2 do
6:    if d_i − c_i > 0 y check − port(f_i+, s_i+) then
7:       S =← S ∪ {c_i+};
8:    end if
9:    if 0 > d_i − c_i y check-port(f_i−,s_i−) then
10:      S ← S ∪ {c_i-}.
11:   end if
12: end for
13: if S ≠ ∅ then
14:   ch = select(S);
15: end if
16: if if S = ∅ then
17:   ch = null;
18: end if
```

En Alg. 3 se muestra el algoritmo de encaminamiento totalmente adaptativo elaborado para una red de tipo toro 2-D. La principal diferencia se encuentra al computar el sentido que puede ser tomado en cada dirección. También, la función *chek-port* ha de tener en cuenta las reglas definidas anteriormente basadas en los cruces de los *wraparound links* para que un mensaje sea *safe* o *unsafe* para redes de tipo toro.

Finalmente, el algoritmo de encaminamiento propuesto selecciona de forma aleatoria el canal de salida del conjunto S, si S es nulo el mensaje se bloquea en el conmutador y será encaminado en el siguiente ciclo.

Algoritmo 3 safe-unsafe-2D-toros

Entradas:

Coordenadas del conmutador actual $C : (c_1, c_2)$,
Coordenadas del conmutador destino $D : (d_1, d_2)$,
Colas libres: (f_1-,f_1+, f_2-, f_2+),
mensajes *safe* : (s_1-, s_1+, s_2-, s_2+);

Salidas:

Canal de salida seleccionado;

1: S=0;ch=null;
2: **if** $C == D1$ **then**
3: ch=interno; return true;
4: **end if**
5: **for** $i == 1$ to 2 **do**
6: **if** $0 < d_i - c_i \leq k/2$ or $d_i - c_i < -k/2$ and $check - port(f_i+, s_i+)$ **then**
7: $S =\leftarrow S \cup \{c_i+\}$;
8: **end if**
9: **if** $d_i - c_i > k/2$ or $-k/2 \leq d_i - c_i < 0$ and check-port(f_i-,s_i-) **then**
10: $S \leftarrow S \cup \{c_i\text{-}\}$.
11: **end if**
12: **end for**
13: **if** $S \neq \emptyset$ **then**
14: $ch = \text{select}(S)$;
15: **end if**
16: **if** if $S = \emptyset$ **then**
17: $ch = null$;
18: **end if**

3.2.1 Justificación Algoritmo Libre de Bloqueo.

En esta sección justificamos que el algoritmo SUR es libre de bloqueo. Para ello utilizamos una aproximación por contradicción.

Supongamos que existe una dependencia cíclica de recursos por parte de mensajes en una malla 2D al utilizar el algoritmo SUR. Como existe el ciclo entonces existen dependencias entre los canales $x \rightarrow y$ e $y \rightarrow x$. Las dependencias $x \rightarrow y$ están permitidas por el algoritmo de encaminamiento base (DOR), pero las dependencias $y \rightarrow x$ no están permitidas. Los mensajes en los puertos de entrada en el eje Y son etiquetados como *unsafe* si este mensaje realiza una petición a un puerto de salida en el eje X. Para crear una situación de interbloqueo, los mensajes dentro del ciclo no deben poder avanzar. Esto significa o bien que todas las colas están llenas o que el algoritmo de encaminamiento no permite avanzar a los mensajes. La primera condición, que las colas estén totalmente llenas no se puede cumplir, puesto que eso significaría que el puerto de entrada en el eje Y contiene en sus dos VCs un mensaje *unsafe*. Esto no

puede pasar debido a que un mensaje etiquetado como *unsafe* solo puede ser enviado al siguiente conmutador si los dos VCs están disponibles o un VC está disponible y el otro VC contiene un mensaje etiquetado como *safe*. La segunda situación (las condiciones de encaminamiento no permiten avanzar a los mensajes) no puede suceder. De hecho, si un mensaje en el puerto de entrada en el eje Y realiza una petición del puerto de salida en el eje X, el puerto de entrada asociado a este puerto de salida contrendrá uno o dos mensajes *safe* o estará completamente vacío. En el caso de que contenga un mensaje *safe* o este vacío, el mensaje *unsafe* en el puerto de entrada Y puede avanzar, por tanto no hay *deadlock*. En el caso de que se almacenen dos mensajes seguros, ambos pueden avanzar, ya que tienen delante de ellos mensajes *safe*, que se moverán porque ellos están utilizando un camino acíclico (conformado por mensajes *safe* usando el algoritmo de encaminamiento base). Por lo tanto, no se pueden bloquear los mensajes en un ciclo. En otras palabras, los mensajes *safe* almacenados en un camino libre de bloqueo tienen siempre reservado un VC, por tanto siempre pueden avanzar. Los mensajes *unsafe* pueden cruzar ciclos pero nunca llenar un puerto de entrada, esto evita los *deadlocks*. Por tanto, para cualquier ciclo, los mensajes *unsafe* nunca llenarán todos los recursos de un puerto de entrada. Éstos podrán avanzar cuando ambos VCs estén disponibles o haya un mensaje seguro en el puerto de entrada.

Para el caso de un toro 2D seguiremos una aproximación parecida. Sin embargo, en este caso los *wraparound links* tienen un rol muy importante. Si el mensaje no tiene que atravesar ningún *wraparound link* para alcanzar su destino en la red, el mensaje sigue el comportamiento descrito arriba, por lo que el mensaje no creará *deadlock*. Si el mensaje está almacenado en un conmutador conectado al *wraparound link*, entonces si el mensaje hace una petición a un *wraparound link*, siendo éste el que menor dimensión tiene que atravesar, el mensaje sigue el algoritmo de encaminamiento base (DOR). Lo cual significa que el mensaje puede ser liberado y etiquetado como tipo *safe*. Por otra parte, si el mensaje realiza la petición a un puerto de salida conectado a un *wraparound link* y la dimensión de éste no es la menor que debe atravesar el mensaje, entonces el mensaje será etiquetado como mensaje de tipo *unsafe*. Como hemos demostrado anteriormente, esto solo pasa si el conmutador al que está conectado tiene los dos VCs vacíos o uno de ellos contiene un mensaje de tipo *safe*. Por lo cual, en ningún momento ningún puerto de entrada podrá contener dos mensajes etiquetados con el tipo *unsafe* que pudieran provocar un bloqueo. En otras palabras, en el caso del toro 2D, todos los puertos de entrada permitirán a los mensajes de tipo *safe* avanzar por la red.

A continuación un ejemplo en la Figura 3.8. En este caso es un toro 1D en el que todos los conmutadores envían mensajes a los conmutadores localizados dos saltos a

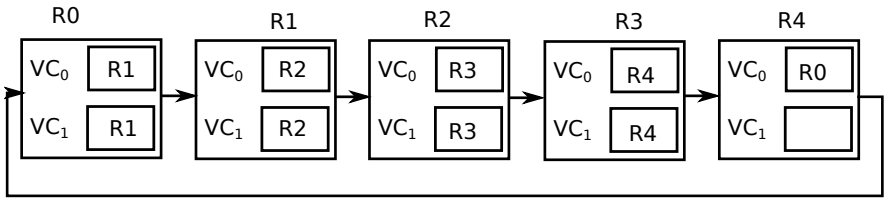


Figura 3.5: Ejemplo de *deadlock-freedomness* en una red tipo toro 1D

la derecha. R0 tiene almacenados en el puerto de entrada dos mensajes que provienen de R4 con destino en R1. Ambos mensajes son etiquetados como tipo *safe* porque han llegado desde R4 atravesando el *wraparound link* con la dimensión más baja que debe atravesar el mensaje. R1 contiene dos mensajes etiquetados *safe*. Estos mensajes tienen destino en R2 y provienen de R0. Como estos mensajes no tienen que atravesar ningún *wraparound link* y siguen el algoritmo de encaminamiento base son etiquetados como *safe*. R2 y R3 están en la misma situación que R1. Finalmente, R4 tiene un mensaje con destino en R0. Este paquete esta etiquetado como *unsafe* porque proviene de un conmutador que no esta conectado al *wraparound link* y todavía debe atravesar el *wraparound link* para alcanzar su destino. Por lo tanto, R3 no puede enviar otro mensaje de tipo *unsafe* a R4. Entonces, los mensajes almacenados en el puerto de entrada de R3 pueden avanzar y ser consumidos en R4, haciendo así que no se bloquee la red.

3.2.2 Ejemplo de TBFC+SUR.

	TIME	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
P_1 Safe	H_1	IB	RT	$\frac{VA}{SA}^{\vee}$	X	LN	IB	RT	$\frac{VA}{SA}^{\vee}$	X	LN						
	B_1		IB	-	$SA^{\vee}$	X	LN	IB	-	$SA^{\vee}$	X	LN					
	T_1			IB	-	$SA^{\vee}$	X	LN	IB	-	$SA^{\vee}$	X	LN					
P_2 Unsafe	H_2		IB	RT	$\frac{VA}{SA}^{\times}$	$\frac{VA}{SA}^{\times}$	$\frac{VA}{SA}^{\vee}$	X	LN	IB	RT	$\frac{VA}{SA}^{\vee}$	X	LN			
	B_2			IB	-	-	-	$SA^{\vee}$	X	LN	IB	-	$SA^{\vee}$	X	LN			
	T_2				IB	-	-	-	$SA^{\vee}$	X	LN	IB	-	$SA^{\vee}$	X	LN		
P_3 Safe	H_3					IB	RT	$\frac{VA}{SA}^{\times}$	$\frac{VA}{SA}^{\times}$	$\frac{VA}{SA}^{\vee}$	X	LN	IB	RT	$\frac{VA}{SA}^{\vee}$	X	LN
	B_3						IB	-	-	-	$SA^{\vee}$	X	LN	IB	-	$SA^{\vee}$	X
	T_3							IB	-	-	-	$SA^{\vee}$	X	LN	IB	-	$SA^{\vee}$
P_4 Unsafe	H_4						IB	RT	$\frac{VA}{SA}^{\times}$	$\frac{VA}{SA}^{\times}$	$\frac{VA}{SA}^{\times}$	$\frac{VA}{SA}^{\times}$	$\frac{VA}{SA}^{\vee}$	X	LN	IB	RT
	B_4							IB	-	-	-	-	-	$SA^{\vee}$	X	LN	IB
	T_4								IB	-	-	-	-	-	$SA^{\vee}$	X	LN

FREE	2	2	1	1	1	0	0	0	0	0	0	0	0	0	1	1
SAFE	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0

Free rec	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0
safe rec	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0

Figura 3.6: Ejemplo TBFC+SUR.

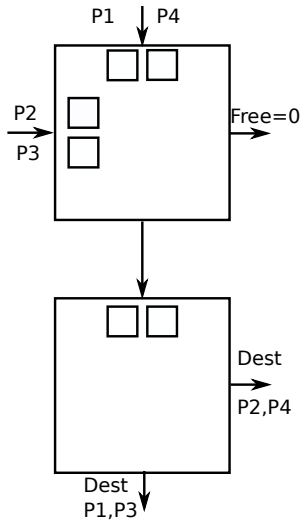


Figura 3.7: Dos conmutadores conectados para el ejemplo de la Figura 3.6

La Figura 3.6 muestra un ejemplo del avance de los flits en dos conmutadores a través de todas sus etapas al utilizar la implementación TBFC+SUR. Asumimos dos conmutadores, llamados X e Y, conectados a través de un enlace con un ciclo de retardo por etapa, situados uno debajo del otro como si se tratase de una parte de una malla 2D, véase Figura 3.7. Asumimos también que se utiliza conmutación de paquete. Las colas del puerto de entrada norte del conmutador Y están vacías, por lo cual los valores almacenados en los campos *FREE* y *SAFE* del puerto de salida sur del conmutador X son 2 y 0, respectivamente.

En el instante t_3 el *flit* de cabecera del mensaje p_1 con destino al sur del conmutador Y compite en las etapas de VA y SA. p_1 gana SA, y el mensaje será enviado en el siguiente ciclo a través del puerto de salida sur. Como p_1 sigue el encaminamiento base (DOR) se considera como un paquete de tipo *SAFE*. De acuerdo con esto, el conmutador X decrementa en 1 el campo *FREE* e incrementa en 1 el campo *SAFE*. En el instante t_4, el conmutador X envía el tipo de paquete (*SAFE*) y el *flit* al conmutador Y. Además, en ese instante p_2 llega a la etapa de VA y SA en el conmutador X. Como hemos asumido que se ha implementado conmutación de paquete, el mensaje p_2 no puede ser transmitido hasta que p_1 no se haya transmitido completamente. En t_6, p_1 liberará el *crossbar* y el *flit* de cabecera del mensaje alcanza el conmutador Y. En ese instante p_2 trata de ganar la etapa VA. En este caso el destino del mensaje esta situado al sur-este del conmutador X, esto provocará un giro ilegal en el algoritmo de encaminamiento XY. La etapa VA llama a la función *check-port()*. Utilizando el puerto de salida sur en la función, p_2 será transmitido y etiquetado como tipo *unsafe* porque no está siguiendo el algoritmo de encaminamiento base. Como SAFE es 1 (el puerto

de entrada tiene almacenado un paquete tipo *safe*), el árbitro SA puede seleccionar p_2 para ser transmitido a través del puerto de salida sur. La información de control del puerto de salida se actualiza: *FREE* se decrementa en 1 y *SAFE* no se modifica. El *flit* de cabecera del mensaje sale del conmutador en t_7.

En t_7, p_3 alcanza la etapa de VA en el conmutador X y éste es bloqueado porque el puerto de salida está siendo utilizado. Por la misma razón p_4 también se bloquea en t_8. En ese instante, p_1 gana la etapa VA/SA en el conmutador Y, por lo que envía al conmutador X la señal de *FREE* y *TYPE*. En t_9, el conmutador X recibe las señales de control de flujo e incrementa *FREE* en uno y como el tipo de la señal *TYPE* es *safe*, decrementa en uno el campo *TYPE*. En este instante, los valores de la información de control de flujo son *FREE* igual a 1 y *TYPE* igual a 0. Entonces, solo los mensajes que sean etiquetados como tipo *safe* pueden ser transmitidos por el puerto de salida sur, como se ha explicado anteriormente. Entonces, ambos mensajes p_3 y p_4 tratan de ganar VA. La función *check-port()* permite al mensaje p_3 ganar los recursos para el puerto de salida sur, porque este mensaje sigue el camino que indica el algoritmo de encaminamiento base. Sin embargo, la función no permite a p_4 acceder a los recursos. Esto sucede porque el destino del mensaje p_4 esta situado al sur-este y por tanto no sigue el algoritmo de encaminamiento base, por ello el mensaje sería transmitido como tipo *unsafe*. El árbitro de VA selecciona p_3 para ser transmitido a través del puerto de salida sur. Entonces, p_3 gana la etapa VA/SA y será transmitido en el siguiente ciclo.

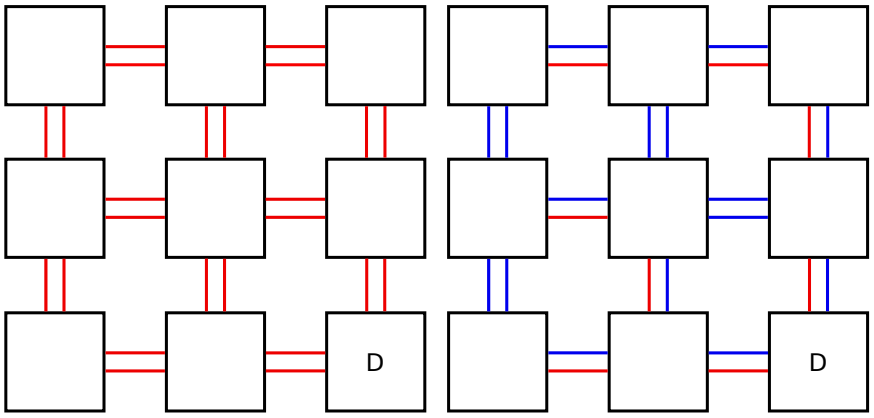
En t_{11}, el *flit* de cabecera del mensaje p_2 gana la etapa VA/SA en el conmutador Y, entonces el conmutador envía hacia el conmutador X las señales de control de flujo. En el siguiente ciclo, t_{12}, las señales de control de flujo alcanzan el conmutador X y *FREE* se incrementa en 1, y como el tipo es *unsafe*, el campo *SAFE* no se modifica. Entonces, p_4 accede a la etapa VA/SA y como el campo *SAFE* es igual a 1, el mensaje puede ser transmitido a través del puerto de salida sur como mensaje de tipo *unsafe*, decrementando la variable de control de flujo *FREE* en 1.

3.3 Filtro End-Point-Congestion (EPC).

En esta sección describimos el funcionamiento del mecanismo End-Point-Congestion Filter(EPC filter), el cual nos permite abordar el segundo de los problemas planteados en este trabajo, es decir, la ramificación de la congestión por la red.

El algoritmo de encaminamiento adaptativo tiende a evitar la congestión permitiendo que los mensajes eviten enlaces congestionados, tomando rutas alternativas. Ahora bien, si la congestión se produce en el nodo destino, esa adaptatividad puede

llevar a que la zona de congestión sea más grande, afectando de una manera más notable a las prestaciones de la red. El objetivo del filtro EPC es el de evitar la ramificación de la congestión dentro de la red de interconexión, evitando así una caída en sus prestaciones. Cuando un conmutador desea transmitir un mensaje, el conmutador comprueba si el mensaje puede contribuir potencialmente a una situación de congestión. Para comprobar esto, si el conmutador ha enviado recientemente un mensaje con el mismo destino y aun está siendo transmitido, entonces el conmutador bloquea el mensaje y no permite que este avance por la red. En otro caso, el mensaje es transmitido utilizando la adaptatividad permitida por el algoritmo de encaminamiento. Los mensajes son bloqueados hasta que el mensaje que está siendo transmitido se haya enviado por completo, esto es un claro indicador de que los mensajes con el mismo destino no construirán una situación de congestión. Por lo tanto, el filtro EPC impide temporalmente la transmisión de algunos mensajes.



(a) Totalmente adaptativo (b) Totalmente adaptativo con EPC

Figura 3.8: Ejemplo de la ramificación de la congestión.

La Figura 3.8 muestra un ejemplo del efecto de nuestro mecanismo en una malla 2D con dos VCs por puerto. Supongamos que todos los nodos quieren enviar mensajes al nodo *hotspot* (D). Las líneas rojas (representaciones de los VCs) significan que ese canal virtual ha sido asignado a un mensaje con destino al *hotspot*. Por otra parte, las líneas azules representan que el mensaje asignado tiene un destino distinto. La Figura 3.8(a) muestra como podría llegar a ser la situación de la red con un algoritmo de encaminamiento totalmente adaptativo. En este caso todos los canales virtuales han sido utilizados por mensajes con el destino al *hotspot*, construyendo así una situación de congestión en la red. La Figura 3.8(b) muestra nuestra propuesta, donde los conmutadores solo permiten la utilización de uno de los canales virtuales de salida para los mensajes con el mism destino, haciendo así que los mensajes destinados al *hotspot*

únicamente puedan utilizar un canal virtual por conmutador. Esto permite que la red no se congestione y que el tráfico no congestionado, o que no tiene como destino el *hotspot*, pueda avanzar por la red.

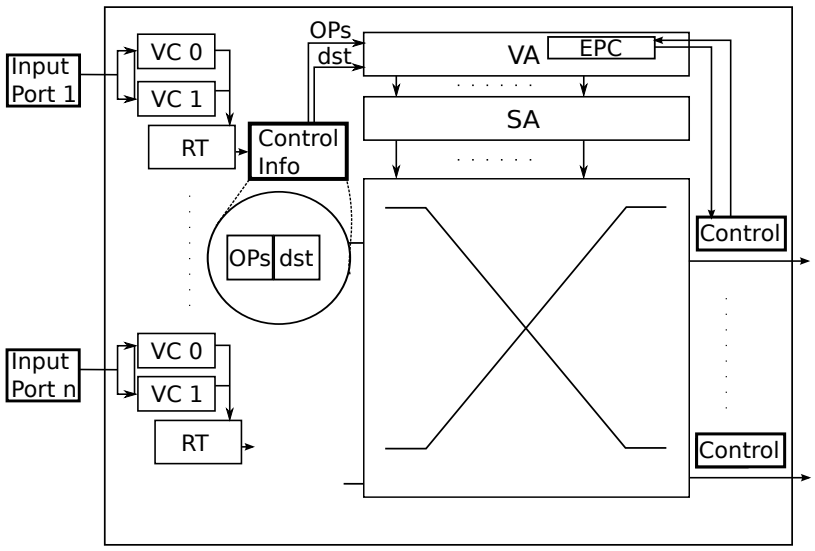
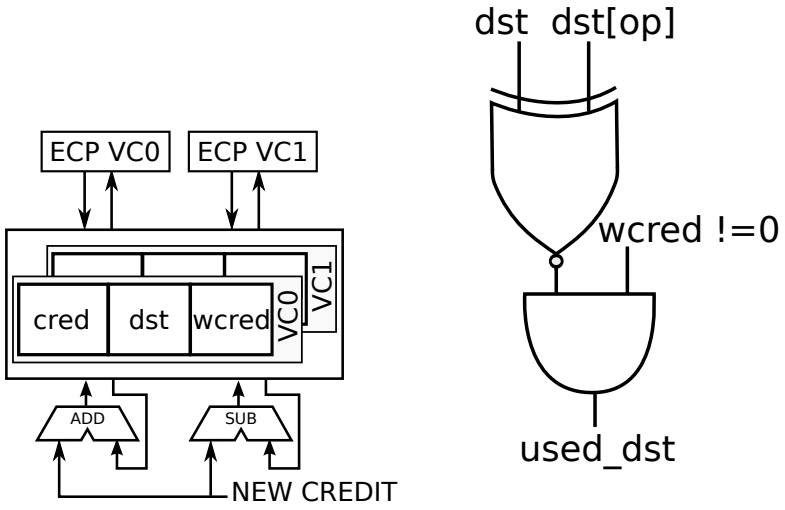


Figura 3.9: Arquitectura de conmutador base incluyendo EPC.

La Figura 3.9 muestra la arquitectura del conmutador asumida para la implementación de este mecanismo junto con las modificaciones necesarias para el correcto funcionamiento del mismo. Esta organización del conmutador es representativa tanto para interconexiones dentro como para fuera del chip. El conmutador está segmentado en 4 etapas con 5 componentes (dos componentes funcionan en paralelo, cada etapa ocupa un ciclo de reloj): *Input buffer* (IB) con una cola por VC, *Routing* (RT), *Virtual Channel Allocation* (VA), *Switch Allocation* (SA) y *Switch Traversal* (X). En RT, el conmutador almacena los puertos de salida seleccionados por el algoritmo de encaminamiento totalmente adaptativo [Dua12]. Estos puertos de salida (OPs) y el destino del mensaje (dst) son almacenados en la información de control (Control Info). En VA, el conmutador asigna los recursos (VC) a los *flits* que los han solicitado. En SA, el arbitro selecciona cual es el puerto de entrada seleccionado en cada ciclo para transmitir un *flit* a través del puerto de salida. Ambas etapas utilizan arbitros en modo Round Robin (RR). Además, ambos árbitros están implementados en 2 etapas y las etapas VA y SA se ejecutan en paralelo. El *crossbar* está multiplexado (solo un flit por puerto de entrada puede acceder al *crossbar* cada ciclo). El conmutador implementa *flit multiplexing* (también conocido como *wormhole flow control* [DT03]). El conmutador utiliza control de flujo basado en créditos y *virtual cut-through* (VCT).

La Figura 3.10(a) muestra la información de control requerida para cada puerto



(a) Output port control (b) EPC

Figura 3.10: Información de control del puerto de salida y EPC.

de salida por cada VC. Como base en la información de control cada VC necesita el número de créditos disponibles (Cred). Para la implementación del filtro EPC se han añadido dos nuevos campos: el destino del mensaje (dst) y el número de créditos a esperar para poder liberar el destino (wcred).

El filtro EPC está implementado en la etapa VA (Figura 3.10(b)). Cuando un *flit* de cabecera alcanza la etapa de VA, el conmutador proporciona al árbitro de VA los puertos de salida seleccionados por el *flit* (OPs) y el destino (dst) del *flit* desde la información de control. Entonces, el filtro EPC compara dst con los destinos almacenados en la información de control del puerto de salida con el valor *wcred* distinto de 0. Si *dst* no coincide, el *flit* puede acceder a la etapa VA y competir por el VC. En otro caso, el *flit* deberá esperar al siguiente ciclo y volver a solicitar el acceso.

Cuando un *flit* de cabecera gana un VC, entonces el conmutador configura los valores de los campos *dst* y *wcred* registrando en ellos la información de control del puerto de salida. En el campo *dst* se almacena el destino del mensaje. El campo *wcred* se actualiza de la siguiente forma: *wcred=queue_size-cred+1*, donde *queue_size* es el tamaño de la cola en *flits* y *cred* es el número de créditos disponibles en el VC. Este valor garantiza que siempre que la cabecera salga del siguiente conmutador el valor *wcred* alcanzará el valor 0, permitiendo de nuevo transmitir mensajes con ese destino. Cuando se recibe un *flit*, el conmutador actualiza los campos, añadiendo 1 a *cred* y reduciendo *wcred* en 1.

3.3.1 Estimación del Coste de EPC.

En esta sección comparamos los recursos necesarios que necesita el filtro EPC contra otras dos técnicas de control de congestión, *BAHIA* [EFG13] y *Regional Congestion Awareness* (RCA) [GGK08]. Ambas técnicas requieren más recursos que el filtro EPC. BAHIA necesita una *Virtual Network* (VN) extra para el tráfico congestionado y además necesita una *dedicated signaling network* (*Burst Notification Network*; BNN) por nodo para poder notificar la congestión. BAHIA en el interfaz de la red necesita dos vectores con una longitud igual al número de nodos y cierta lógica para manejar las señales de congestión. RCA también necesita una red de monitorización de baja frecuencia para propagar la información de congestión. Los conmutadores de las redes RCA necesitan dos módulos extra por puerto, uno para agregar la información y otro para propagar dicha información. RCA requiere también de un *Congestion Value Registers*(CVR). Como se puede observar estos dos mecanismos requieren de una red adicional mientras que el filtro EPC no lo necesita, por lo cual el filtro EPC exhibe un menor *overhead* en área. Además, nótese que el filtro EPC no elimina la congestión, solo previene la ramificación de la congestión debido al algoritmo de encaminamiento adaptativo, por lo cual BAHIA y RCA son ortogonales al filtro EPC.

Chapter 4

Evaluación de los Mecanismos.

En este capítulo realizamos la evaluación y análisis de las propuestas. En particular, para cada mecanismo describimos primeramente los parámetros utilizados en la simulación y posteriormente analizamos los resultados obtenidos.

4.1 TBFC+SUR.

Seguidamente, mostramos los parámetros de configuración y los resultados obtenidos con TBFC y SUR. Como escenario analizamos una malla 2D con 64 conmutadores y un toro 2D con 64 conmutadores.

4.1.1 Parámetros del análisis.

Hemos modelado un conmutador con cuatro etapas, con dos VCs y *flit-level crossbar switching*. La tabla 4.1 muestra los parámetros de simulación utilizados para la malla 2D con 64 conmutadores. Los mensajes transitorios y permanentes representan el número de mensajes procesados hasta que el simulador entra en estado permanente [1] y finaliza la simulación, respectivamente. En este escenario, se analizan tres algoritmos de encaminamiento: determinista (XY), encaminamiento totalmente adaptativo (FA) utilizando el control de flujo típico basado en créditos y SUR con TBFC.

Para el escenario del toro 2D con 64 conmutadores se utilizan los mismos parámetros excepto el número de VCs y el tamaño de la cola. Estos parámetros varían dependiendo del algoritmo de encaminamiento y el control de flujo utilizados. La tabla 4.2 muestra los valores escogidos para estos parámetros para cada algoritmo de encaminamiento. En este escenario analizamos cinco algoritmos de encaminamiento: encaminamiento

[1] El simulador descarta las estadísticas hasta ese punto.

Parámetros	FA, SUR_2VC, XY
Topología de red	malla 8x8
VCs por puerto de entrada	2
Tamaño del mensaje	80 bytes
Tamaño del flit	4 bytes
Tamaño de la cola	20 flits
Tiempo de vuelo	1 ciclo
Mensajes transitorios	10000
Mensajes permanentes	10000

Table 4.1: Parámetros y valores usados para los experimentos en la malla 2D.

determinista (DOR), encaminamiento totalmente adaptativo con un canal adaptativo y dos canales de escape (FA), encaminamiento totalmente adaptativo con control de flujo de tipo burbuja [Pue01] (FA bubble) usando un canal adaptativo y un canal de escape con el doble de tamaño (para implementar la burbuja). También analizamos TBFC+SUR con dos y tres canales virtuales (SUR 2VC y SUR 3VC). Todas estas configuraciones (excepto SUR 3VC) utilizan los mínimos recursos de memoria para ser libres de bloqueo y garantizar VCT.

Encaminamiento	VCs	Tamaño de cola
FA_Bubble	2	20 flits adap, 40 flits esc
FA, SUR_3VC	3	20 flits
XY, SUR_2VC	2	20 flits

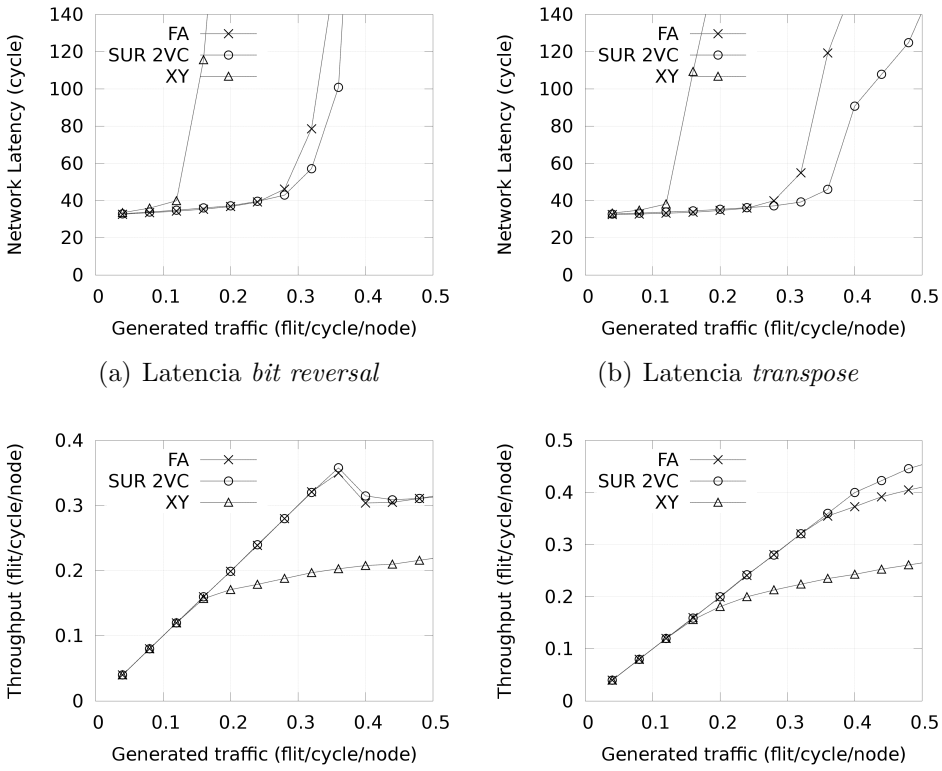
Table 4.2: Parámetros y valores usados para los experimentos en el toro 2D.

Evaluamos seis distribuciones de tráfico: *bit-complement, bit-reversal, transpose, perfect shuffle, uniform y hotspot*. Por cuestiones de espacio solo se muestran cuatro de ellos:*bit-reversal, transpose, uniform y hotspot*. [2] En tráfico *bit-reversal*, el nodo con valor binario a_{n-1}, a_{n-2},..., a_1, a_0 se comunica con el nodo a_0, a_1,..., a_{n-2}, a_{n-1}. Para tráfico *transpose* el nodo con valor binario a_{n-1}, a_{n-2},..., a_1, a_0 envía los mensajes al nodo $a_{n/2-1}$,... a_0, a_{n-1}, ...$a_{n/2}$. Finalmente, en tráfico *hotspot*, algunos nodos envían un 20% del tráfico a un nodo y el resto del tráfico a cualquier otro nodo con la misma probabilidad. El resto de nodos siguen inyectando usando una distribución de tráfico uniforme.

4.1.2 Resultados de rendimiento.

En este apartado discutimos los resultados obtenidos para ambos escenarios.

[2]Hemos obtenido conclusiones similares para los tráficos no mostrados.



(a) Latencia *bit reversal* (b) Latencia *transpose*

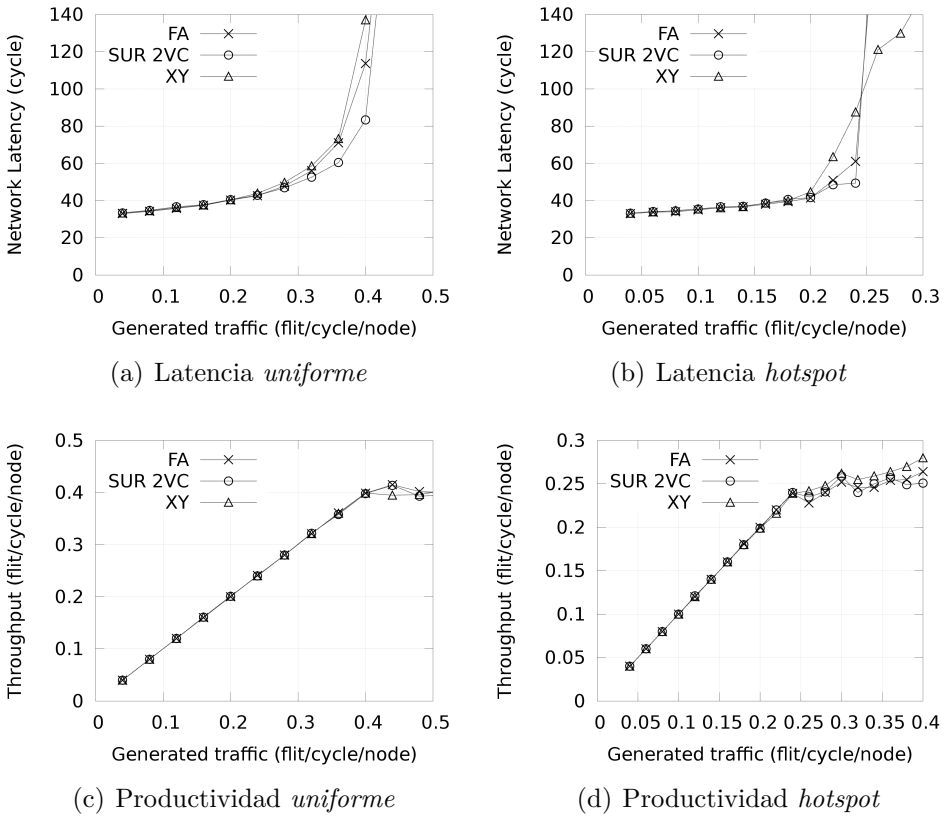
(c) Productividad *bit reversal* (d) Productividad *transpose*

Figura 4.1: Resultados de rendimiento en mallas 8×8, tráfico bit reversal y transpose para TBFC+SUR.

Las Figuras 4.1 y 4.2 presentan los resultados para el escenario propuesto. Las Figuras 4.1(a) y 4.1(c) muestran los resultados para tráfico *bit-reversal*. En este escenario, nuestra propuesta obtiene resultados de productividad similares a los obtenidos por FA. Sin embargo, SUR mejora la latencia cerca del punto de saturación, cuando se compara con el algoritmo FA. En cualquier caso, ambos algoritmos adaptativos obtienen mejores resultados que XY.

Con tráfico *transpose*, Figuras 4.1(b) y 4.1(d), SUR mejora los resultados de FA sobre un 10% en cuanto a productividad de la red. En latencia SUR también mejora FA cerca del punto de saturación. Para el resto de distribuciones de tráfico (uniforme y hotspot; resto de las figuras 4.2(a) 4.2(b) 4.2(c) 4.2(d)) se observan resultados similares para los tres algoritmos de encaminamiento. SUR, FA y XY obtienen resultados similares en cuanto a productividad, y la latencia de SUR es levemente más baja cerca del punto saturación.

En el caso del Toro 2D las diferencias en los resultados son más significativas. Las Figuras 4.3(a) y 4.3(c) muestran los resultados con tráfico *bit-reversal*. SUR 2VC mejora los resultados de latencia obtenidos tanto por FA como por FA bubble. Además,



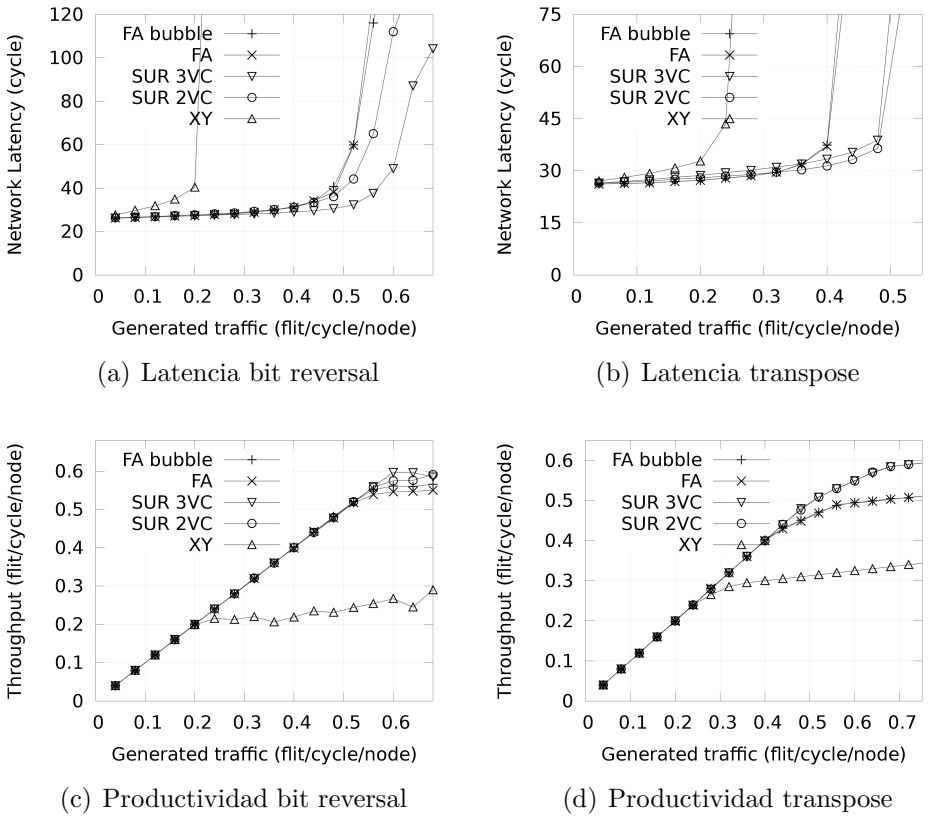
(a) Latencia *uniforme* (b) Latencia *hotspot*

(c) Productividad *uniforme* (d) Productividad *hotspot*

Figura 4.2: Resultados de rendimiento en mallas 8×8, tráfico uniform y hotspot para TBFC+SUR.

SUR 2VC mejora estos resultados utilizando menos recursos en las colas (2VCs con 20 slots cada uno, mientras que FA requiere 3 VCs con 20 slots cada uno o FA bubble requiere 1 VC con 20 slots y otro con 40 slots. Es más, con el mismo número de recursos, SUR 3VC trabaja mucho mejor que FA y FA bubble, puesto que los mejora tanto en latencia como en productividad (9% mejor). En tráfico *transpose*, Figuras 4.3(b) y 4.3(d) nuestra propuesta obtiene resultados mucho mejores que FA y FA bubble. En este caso, las dos versiones de SUR obtienen una mejora del 20% en productividad con respecto a FA. También, las dos versiones de SUR obtienen mejores resultados en latencia del *flit* en la red.

Finalmente, estas mejoras también se han obtenido para los patrones de tráfico uniforme y *hotspot*. Las Figuras4.4(a) y 4.4(c) presentan la comparación de los resultados obtenidos en tráfico uniforme. SUR 2VC y FA obtienen resultados similares en latencia y productividad (nótese que SUR requiere menos recursos). SUR 3VC obtiene los mejores resultados tanto en latencia del *flit* en la red como de productividad, estos resultados son un 14% mejores que los obtenidos por FA. Con tráfico *hotspot*, Figuras 4.4(b) and 4.4(d), los resultados muestran como SUR 3VC mejora las prestaciones de



(a) Latencia bit reversal

(b) Latencia transpose

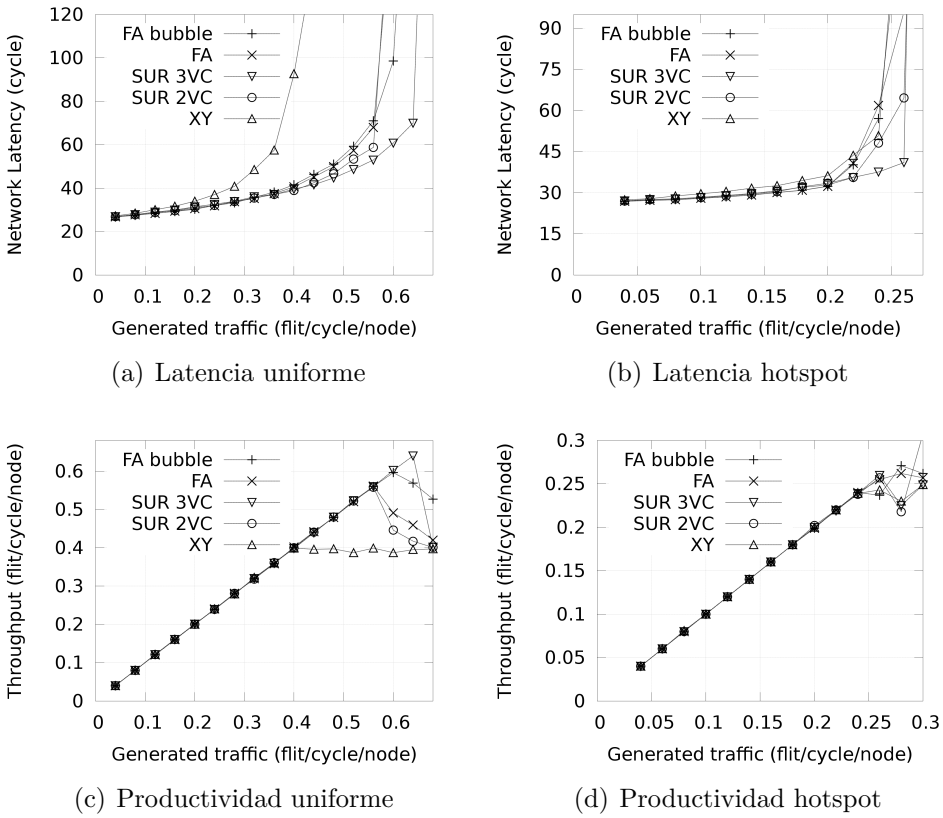
(c) Productividad bit reversal

(d) Productividad transpose

Figura 4.3: Resultados de rendimiento en toros 8×8, tráfico bit-reversal y transpose para TBFC+SUR.

FA y FA bubble en latencia del *flit* en la red, mientras que se aprecia que todos los algoritmos de encaminamiento obtienen resultados similares en productividad.

Como se observa en los resultados, SUR junto con TBFC mejoran la productividad y la latencia en la red comparado con FA. En la figura 4.5 se muestra un ejemplo que destaca porqué se consigue esta mejora sobre FA. La figura muestra una malla 2x2. Asúmase que R0 quiere enviar a R3. En FA, R0 puede enviar el mensaje a R1 o R2. En el caso de que se enviara hacia R1 el mensaje puede ser enviado por el canal adaptativo y el canal de escape (conforme el algoritmo XY), si, por el contrario, se envía el mensaje hacia R2, solo puede ser enviado por el canal adaptativo. Por lo tanto, el mensaje puede ser asignado a los dos VCs de R1 y a un VC en R2. Con el algoritmo totalmente adaptativo (que promueve el uso de canal adaptativo sobre el de escape) puede usar solo dos VC (uno en cada conmutador). En el caso de usar FA optimizado (donde se da la misma prioridad a canales adaptativos y de escape) se pueden usar tres VC. Sin embargo en SUR, los mensajes *safe* pueden asignarse a cualquiera de los cuatro canales. Incluso los mensajes *unsafe* pueden usar cualquiera de los cuatro VCs (teniendo en cuenta que hay un VC vacío en el puerto de entrada). Por tanto, SUR



(a) Latencia uniforme

(b) Latencia hotspot

(c) Productividad uniforme

(d) Productividad hotspot

Figura 4.4: Resultados de rendimiento en toros 8×8, tráfico uniforme y hotspot para TBFC+SUR.

tiene más opciones para asignar el mensaje, obteniendo así mejores resultados que FA.

4.1.3 Equilibrado de la utilización VCs.

La Figura 4.5 muestra la utilización de los VCs en el escenario descrito anteriormente con un tráfico uniforme. Como se puede observar, en FA, a niveles bajos de carga (antes de la saturación) la utilización de canal VC0, que corresponde al canal adaptativo, es muy superior a la carga soportada por VC1. Sin embargo, en SUR, para todas las cargas de tráfico, se obtiene una utilización equilibrada de los canales virtuales.

4.2 EPC.

Seguidamente, pasamos a evaluar el mecanismo EPC.

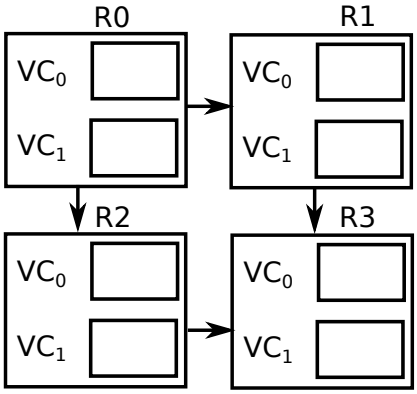


Figura 4.5: Ejemplo malla 2D.

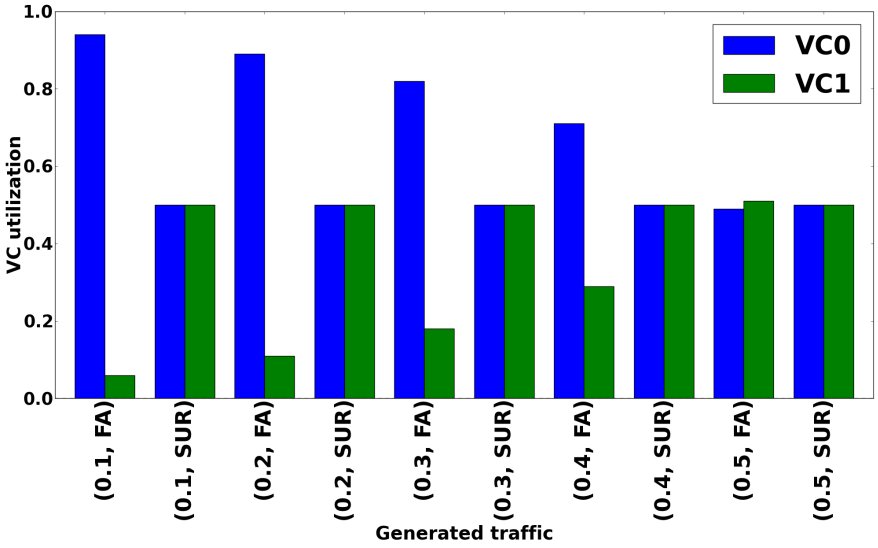


Figura 4.6: Utilización de canales.

4.2.1 Parámetros del análisis.

La Tabla 4.3 muestra los parámetros utilizados para la evaluación de este mecanismo. En este caso la topología analizada es una malla 2D con 16 conmutadores ordenados de forma 4x4.

Para este mecanismo analizamos tres posibles escenarios. El primero (UNIF) hace referencia a cuando se utiliza una distribución de tráfico de tipo uniforme, por lo tanto no hay congestión en la red. El segundo escenario (C_LOW) hace referencia a que existe una pequeña congestión en la red. En concreto, ocho nodos (seleccionados aleatoriamente) envían mensajes al nodo 11 con un 30% de probabilidad, mientras que el resto del tráfico se distribuye uniformemente. En este escenario también se presentan los resultados para una malla 8x8. El tercer escenario (C_HIGH) hace referencia a un

Parámetros	Valor
Topología de red	Malla 4x4, Malla 8x8
Encaminamientos	XY, FA y FA-EPC
Flow control	Virtual cut-through
VCs por puerto de entrada	2
Tamaño del mensaje	16 bytes
Tamaño del flit	4 bytes
Tamaño de la cola	4 flits
Tiempo de vuelo	1 ciclo
Mensajes transitorios	10000
Mensajes permanentes	10000

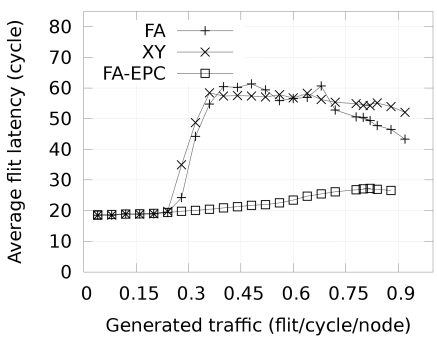
Table 4.3: Parámetros y valores usados para los experimentos de EPC.

escenario igual al segundo pero con un aumento de la probabilidad de enviar mensajes al nodo 11, en este caso se envían el 70% de los mensajes.

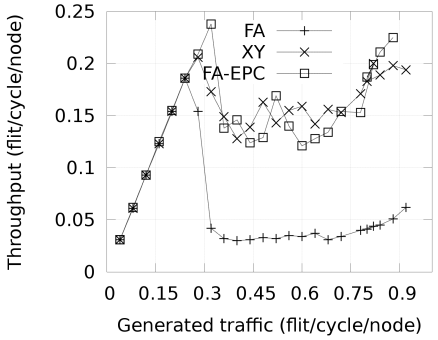
A diferencia de los resultados mostrados para TBFC+SUR que mostraban medias generales del tráfico, para este mecanismo hacemos una diferenciación entre tráfico dirigido al nodo *hotspot* y el resto de tráfico de la red, tráfico *background*. Para hacer esto, se etiquetan los primeros 10.000 mensajes generados después de que los mensajes transitorios hayan alcanzado su destino y la red se encuentre en un estado estable. Para calcular las estadísticas de la red solo se tienen en cuenta los mensajes marcados. Esta modificación se realiza para asegurar que los mensajes utilizados para calcular los resultados mantienen la distribución constante desde la inyección hasta la recepción, asegurándonos entonces de que la distribución de tráfico no se modifica por las posibles situaciones de congestión en la red.

4.2.2 Resultados de Rendimiento.

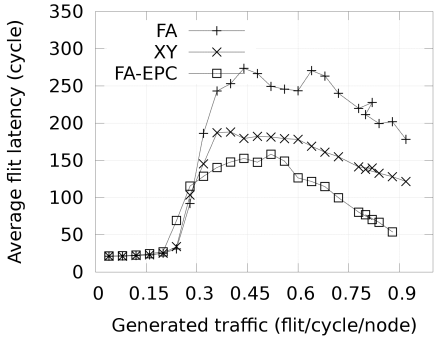
Las Figuras 4.7(a) y 4.7(b) muestran los resultados obtenidos para el tráfico *background* (tráfico uniforme que no tiene como destino el nodo *hotspot*) para el escenario C_LOW. Como se puede observar la latencia del mensaje en la red en FA-EPC es tres veces menor que la obtenida por FA y XY. Como se observa, el incremento de la latencia en FA y XY tiene un incremento muy pronunciado alrededor de una tasa de inyección de 0.3 flit/ciclo/nodo. Este es el momento en el que la congestión producida por el *hotspot* afecta al tráfico *background*. Sin embargo, en FA-EPC la latencia incrementa linealmente y esto se debe al incremento de tráfico *background* en la red. Esta diferencia en la latencia del mensaje en la red es producida porque sin el filtro EPC la congestión en la red se ramifica. Sin embargo, el filtro EPC previene esta ramificación



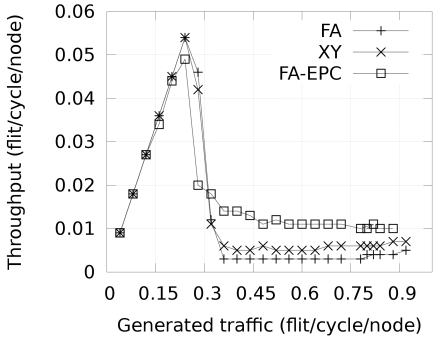
(a) Latencia de red tráfico background



(b) Productividad tráfico background



(c) Latencia de red tráfico hotspot



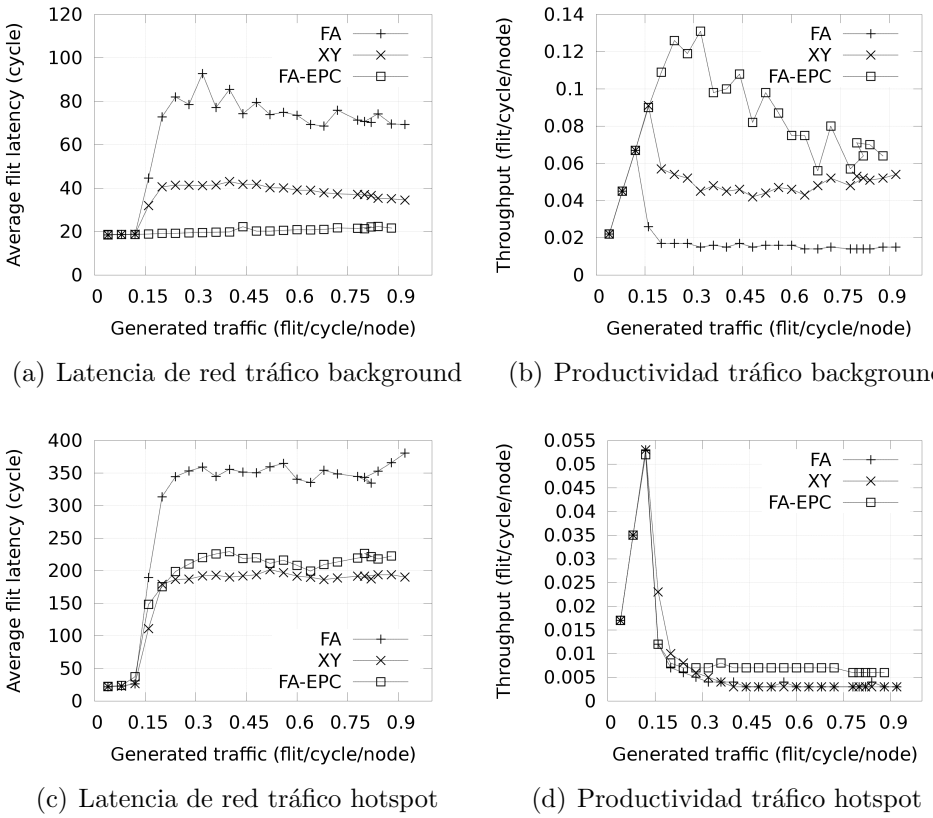
(d) Productividad tráfico hotspot

Figura 4.7: Resultados de rendimiento para distribución de tráfico *hotspot* en una malla 4×4 para EPC. El porcentaje de *hotspot* es 30%.

y causa que el tráfico de tipo *background* no se vea alterado por el *hotspot*. En cuanto a la productividad del tráfico *background* (Figura 4.7(b)), FA alcanza la saturación en 0.25 flits/ciclo/nodo, mientras que el filtro EPC en 0.32 flits/ciclo/nodo. Nótese también que el algoritmo de encaminamiento determinista XY se comporta mejor que FA. Esto es debido a que el algoritmo totalmente adaptativo ramifica la congestión por toda la red, como ya hemos comentado.

Las Figuras 4.7(c) y 4.7(d) muestran los resultados para el tráfico *hotspot* en este escenario. Como ocurre con el tráfico *background*, FA-EPC mantiene la latencia más baja, alrededor de un 50 %, cuando ya se ha congestionado la red (después de alcanzar el punto de saturación). Para el tráfico *hotspot* los tres algoritmos de encaminamiento alcanzan el punto de saturación al mismo tiempo. Esto se produce porque naturalmente el tráfico satura la red, debido a que se demanda más ancho de banda que el ofrecido por el destino congestionado.

Seguidamente analizamos los resultados obtenidos para el escenario C_HIGH. En este caso, se hace más destacable el impacto del efecto del filtro EPC en el tráfico *background*. La Figura 4.8(a) muestra como FA para tráfico *background* obtiene una



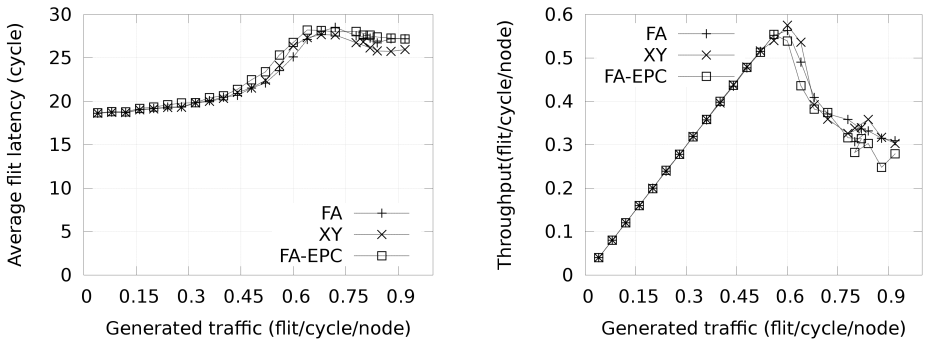
(a) Latencia de red tráfico background (b) Productividad tráfico background

(c) Latencia de red tráfico hotspot (d) Productividad tráfico hotspot

Figura 4.8: Resultados de rendimiento para distribución de tráfico *hotspot* en una malla 4×4 para EPC. El porcentaje de hotspot es 70%.

latencia del mensaje en la red cuatro veces más grande que la conseguida por FA-ECP. En cuanto a la productividad obtenida para el tráfico *background*, como muestra la Figura 4.8(b), FA alcanza el punto de saturación en 0.07 flits/ciclo/nodo. Sin embargo, el filtro EPC permite a la red alcanzar 0.13 flits/ciclo/nodo. Esto significa alcanzar prácticamente el doble de productividad de la red. Después de alcanzar el punto de saturación el filtro EPC obtiene al menos tres veces más productividad que FA. El tráfico *hotspot* obtiene resultados muy similares como muestran las Figuras 4.8(c) y 4.8(d).

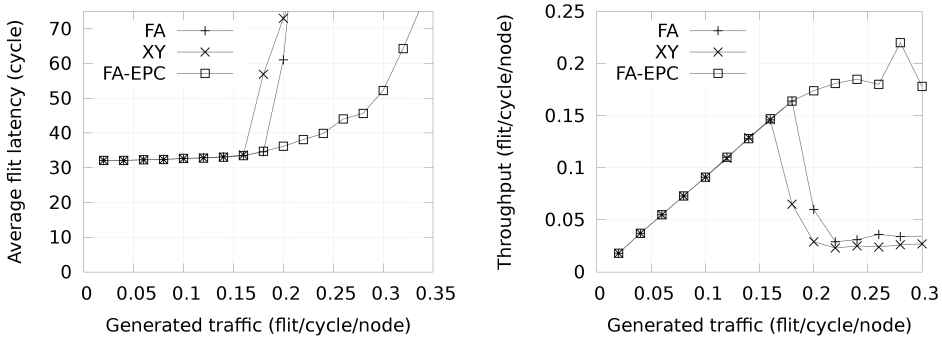
En el escenario UNIF, solo se muestran las gráficas para el tráfico *background*. La Figura 4.9(a) muestra como el filtro EPC tiene alrededor de un 8% de *overhead* en la latencia del mensaje. Este pequeño *overhead* se produce porque el filtro, aunque no exista la situación de congestión, debe bloquear temporalmente los mensajes no congestionados. Sin embargo, la productividad obtenida en la red es prácticamente la misma para todos los algoritmos de encaminamiento. Este pequeño *overhead* en tráfico uniforme nos sugiere que el filtro EPC debe ser utilizado en escenarios específicos, y que debe existir la posibilidad de activar y desactivar el mecanismo dinámicamente.



(a) Latencia de red (b) Productividad

Figura 4.9: Resultados de rendimiento para la distribución de tráfico uniforme en una malla 4×4 para EPC.

Un análisis interesante se deriva de comparar las figuras 4.8(a) and 4.9(a). Si se compara la latencia del tráfico *background* para el escenario C_HIGH y el escenario UNIF, se puede observar como la latencia es muy similar en ambos casos. Esto significa que el filtro EPC desacopla completamente el tráfico *background* del efecto de la congestión en la red.



(a) Latencia de red (b) Productividad

Figura 4.10: Resultados de rendimiento para distribución de tráfico *hotspot* en una malla 8×8 para EPC. El porcentaje de hotspot es 30%.

Finalmente, la Figura 4.10 muestra los resultados obtenidos por el filtro EPC en una malla 8×8 para tráfico *background*. Como muestra la Figura 4.10(a), el filtro EPC obtiene casi el doble de productividad para este tráfico. Como se observa, la latencia tampoco se ve afectada para esta configuración de la red como sucedía en la malla 4×4, Figura 4.7(a). Estos datos reflejan que el filtro EPC escala correctamente.

Chapter 5

Conclusiones y Trabajo Futuro

5.1 Conclusiones.

En esta tesina se han abordado dos problemas actuales de las redes en el chip, el uso desbalanceado de las colas en los puertos de entrada y la ramificación de la congestión en la red. Pasa solventar estos problemas se han propuesto dos soluciones.

Primeramente hemos presentado un mecanismo de control de flujo, *Type-Based Flow-Control* (TBFC), con un nuevo algoritmo de encaminamiento, *Safe/Unsafe routing* (SUR), el cual permite la optimización del balanceado del uso de las colas. Esto se consigue al no distinguir entre canal adaptativo y escape. La combinación de TBFC y SUR permite reducir el número de canales virtuales requeridos para implementar adaptatividad total ya sea en mallas o en toros. Los resultados de simulación muestran como nuestra propuesta mejora los métodos anteriores bajo diferentes patrones de tráfico.

Finalmente, para el segundo de los problemas planteados, la ramificación de la congestión, hemos propuesto un novedoso mecanismo llamado filtro End-Point-Congestion (filtro EPC), este filtro actuá previniendo la ramificación de la congestión bloqueando los paquetes que se dirigen a un mismo destino, haciendo que solo puedan ocupar un canal en un puerto de salida. Los resultados obtenidos han demostrado que el efecto del filtro mejora los resultados obtenidos por el algoritmo de encaminamiento totalmente adaptativo, puesto que no permite la ramificación. Además, haciendo una comparación del tráfico *background*, los resultados muestran que el filtro con tráfico intenso al *hotspot* y con tráfico uniforme obtienen resultados similares, lo que significa que el filtro desacopla totalmente el tráfico *background* del tráfico congestionado en la red.

5.2 Trabajo Futuro.

La linea de investigación seguida puede extenderse como se ha nombrado ya en el texto haciendo que el filtro pueda activarse y desactivarse adaptativamente cuando se detecte la congestión. Además de realizar una implementación real en VHDL para estimar los costes de los mecanismos propuestos en esta tesina. Por otra parte, otra posible continuación de estos estudios sería la utilización de estas técnicas en las redes en chip para el control de tráfico de coherencia de memoria.

5.3 Artículos Publicados y Relación con la Tesina.

- Miguel Gorgues, José Flich, Dong Xiang, José Duato *Codiseño de un Mecanismo de Control de Flujo y un Algoritmo de Encaminamiento.* Este artículo ha sido aceptado en el congreso "Jornadas SARTECO 2014". En él se presentan TBFC y SUR para mallas.

- Miguel Gorgues, Dong Xiang, José Flich, Zhigang Yu, José Duato *Achieving Balanced Buffer Utilization with a Proper Co-Design of Flow Control and Routing Algorithm.* Este artículo ha sido aceptado en el congreso "8th International Symposium on Networks-on-Chip. NOCs. Ferrara, Sept. 2014". En él se presentan TBFC y SUR tanto para mallas como para Toros.

- Miguel Gorgues, José Flich *End-Point Congestion Filter for Adaptive Routing with Congestion-Insensitive Performance.* Este artículo está en proceso de revisión (major revision) en la revista "*IEEE Computer Architecture Letters*". En él se presenta el filtro EPC.

En este trabajo también se ha colaborado con el profesor Dong Xiang de la School of Software, Tsinghua University, China. En concreto se ha trabajado en la elaboración y definición del algoritmo SUR y del mecanismo de control de flujo TBFC.

Bibliography

[ANM+05] P. Avasare, V. Nollet, J-Y. Mignolet, D. Verkest, and H. Corporaal. Centralized end-to-end flow control in a best-effort network-on-chip. In *Procedigns of the 5th ACM International Conference on Embedded Software*, pages 17–20, 2005.

[BM06] Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of network-on-chip. In *Journal ACM Computing Surveys*, 2006.

[Chi00] G. M. Chiu. The odd-even turn model for adaptive routing. In *EEE Transactions on Parallel and Distributed Systems, vol. 11*, pages 729–738, 2000.

[DJF+05] J. Duato, I. Johnson, J. Flich, F. Naven, and T. Nachiondo P. García. A new scalable and cost-effective congestion management strategy for lossless multistage interconnection networks. In *IEEE International Symposium On High Performance Computer Architecture 2005*, 2005.

[DP01] J. Duato and T. M. Pinkston. Ieee trans. parallel distributed systems, vol. 12, no. 12. In *Computer Architecture Letters, 2008*, pages 1219–1235, 2001.

[DS12] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. In *SIGARCH Computer Architecture News, vol. 31, no. 2*, pages 547–553, 2012.

[DT01] W.J. Dally and B. Towles. Route packets, not wires: on-chip interconnection networks. In *Design Automation Conference, 2001. Proceedings*, pages 684 – 689, 2001.

[DT03] W. Dally and B. Towles. Principles and practices of interconnection networks. In *M. Kaufmann Publishers Inc., San Francisco, USA*, 2003.

[Dua95] J. Duato. A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks. In *IEEE Transactions Parallel Distributed Systems, vol. 6, no. 10,*, pages 1055–1067, 1995.

[Dua12] J. Duato. A new theory of deadlock-free adaptive routing in wormhole networks. In *IEEE Transactions on Parallel and Distributed Systems, 4(12)*, pages 1320–1331, 2012.

[EFG13] Jose V. Escamilla, Jose Flich, and Pedro Javier Garcia. Head-of-line blocking avoidance in networks-on-chip. In *IEEE International Parallel and Distributed Processing Symposium Workshops 2013*, 2013.

[EFG14] Jose V. Escamilla, Jose Flich, and Pedro Javier Garcia. Icaro: Congestion isolation in networks-on-chip. In *International Symposium on Networks-on-Chip, NOCS 2014*, 2014.

[FBR+09] J. Ferrer, E. Baydal, A. Robles, P. López, and J. Duato. Progressive congestion management based on packet marking and validation techniques. In *IEEE Transactions on Computers, vol 61*, 2009.

[GGK08] Paul Gratz, Boris Grot, and Stephen W. Keckler. Regional congestion awareness for load balance in networks-on-chip. In *IEEE International Symposium On High Performance Computer Architecture 2008*, 2008.

[GL06] G.DeMicheli and L.Benini. Networks on chips: Technology and tools. In *Networks on Chips: Technology And Tools*, 2006.

[GN92] C. J. Glass and L. Ni. The turn model for adaptive routing. In *SIGARCH Computer Architecture News vol. 20, no. 2,*, pages 278–287, 1992.

[MAW99] N.W. McKeown, A. Anantharam, and J. Walrand. Achieving 100% in an input-queued switch. In *IEEE Transactions on Communications, vol. 47, no. 8*, 1999.

[NA06] I. Nousias and T. Aslan. Wormhole routing with virtual channels using adaptive rate control for network-on-chip (noc). In *Procedigns of the First International Conference on Adaptive Hardware and Systems*, pages 420–423, 2006.

[PD00] L. S. Peh and W. J. Dally. Flit-reservation flow control. In *Procedigns of the 6th International Symposium on HPCA*, pages 73–84, 2000.

[Pue01] V. Puente. The adaptive bubble router. In *Journal of Parallel and Distributed Computing, vol. 61,no. 9*, pages 1180–1208, 2001.

[SDGT03] A. Singh, W. J. Dally, A. K. Gupta, and B. Towles. Goal: a load-balanced adaptive routing algorithm for torus networks. In *SIGARCH Computer Architecture News, vol. 31, no. 2*, pages 194–205, 2003.

[TL09] M. Tang and X. Lin. A new adaptive flow control for mesh-based network-on-chip. In *Procedigns of 8th International Computer and Information Science. Science*, pages 255–260, 2009.

[Wu03] J. Wu. A fault-tolerant and deadlock-free routing protocol in 2d meshes based on odd-even turn model. In *IEEE Transactions in Computers, vol. 52, no. 9*, pages 1154–1169, 2003.

[XL08] D. Xiang and W. Luo. An efficient adaptive deadlock-free routing algorithm for torus networks. In *IEEE Transactions in Parallel and Distributed Systems, vol. 23, no. 5*, pages 800–808, 2008.

[XZZY10] Yi Xu, Bo Zhao, Youtao Zhang, and Jun Yang. Simple virtual channel allocation for high throughput and high frequency on-chip routers. In *IEEE International Symposium On High Performance Computer Architecture 2010*, pages 1–11, 2010.

[YTL87] P. Yew, N. Tzeng, and D.H. Lawrie. Distributing hot-spot addressing in large-scale multiprocessors. In *IEEE Transactions on Computers, vol. 36*, 1987.