Capítulo 3

Multicomputadores

Los multiprocesadores de memoria compartida presentan algunas desventajas como por ejemplo:

1. Son necesarias técnicas de sincronización para controlar el acceso a las variables compartidas
2. La contención en la memoria puede reducir significativamente la velocidad del sistema.
3. No son fácilmente ampliables para acomodar un gran número de procesadores.

Un sistema multiprocesador alternativo al sistema de memoria compartida que elimina los problemas arriba indicados es tener únicamente una memoria local por procesador eliminando toda la memoria compartida del sistema. El código para cada procesador se carga en la memoria local al igual que cualquier dato que sea necesario. Los programas todavía están divididos en diferentes partes, como en el sistema de memoria compartida, y dichas partes todavía se ejecutan concurrentemente por procesadores individuales. Cuando los procesadores necesitan acceder a la información de otro procesador, o enviar información a otro procesador, se comunican enviando mensajes. Los datos no están almacenados globalmente en el sistema; si más de un proceso necesita un dato, éste debe duplicarse y ser enviado a todos los procesadores peticionarios. A estos sistemas se les suele denominar multicomputadores.

La arquitectura básica de un sistema multiprocesador de paso de mensajes se muestra en la figura 3.1. Un multiprocesador de paso de mensajes consta de nodos, que normalmente están conectados mediante enlaces directos a otros pocos nodos. Cada nodo está compuesto por un procesador junto con una memoria local y canales de comunicación de entrada/salida. No existen localizaciones de memoria global. La memoria local de cada nodo sólo puede ser accedida por el procesador de dicho nodo. Cada memoria local puede usar las mismas direcciones. Dado que cada nodo es un ordenador autocontenido, a los multiprocesadores de paso de mensajes se les suelen denominar multicomputadores.

El número de nodos puede ser tan pequeño como 16 (o menos), o tan grande como varios millares (o más). Sin embargo, la arquitectura de paso de mensajes muestra sus ventajas sobre los sistemas de memoria compartida cuando el número de procesadores es grande. Para sistemas multiprocesadores pequeños, los sistemas de memoria compartida presentarán probablemente un mejor rendimiento y mayor flexibilidad. El número de canales físicos entre nodos suele oscilar entre cuatro y ocho. La principal ventaja de

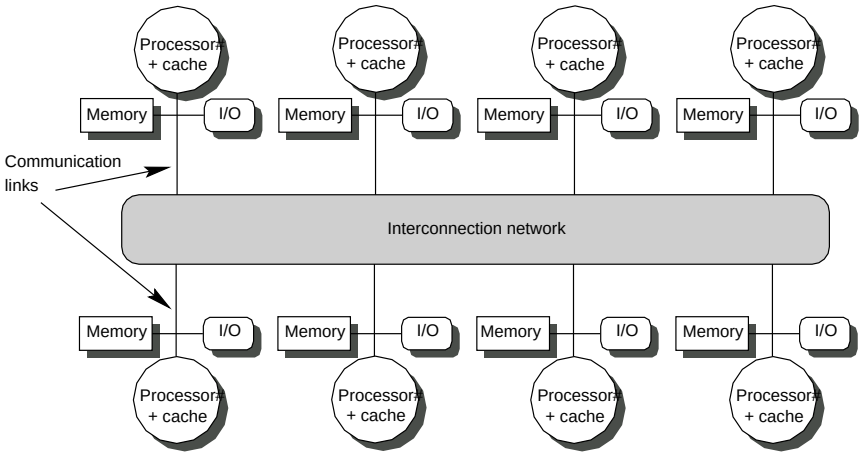


Figura 3.1: Arquitectura básica de un multicomputador.

esta arquitectura es que es directamente escalable y presenta un bajo coste para sistemas grandes. Encaja en un diseño VLSI, con uno o más nodos fabricados en un chip, o en pocos chips, dependiendo de la cantidad de memoria local que se proporcione.

Cada nodo ejecuta uno o más *procesos*. Un proceso consiste a menudo en un código secuencial, como el que se encontraría en un ordenador von Neumann. Si existe más de un proceso en un procesador, éste puede eliminarse del planificador cuando está a la espera de enviar o recibir un mensaje, permitiendo el inicio de otro proceso. Se permite el paso de mensajes entre los distintos procesos de un procesador mediante el uso de canales internos. Los mensajes entre procesos pertenecientes a diferentes procesadores se pasan a través de canales externos usando los canales físicos de comunicación existentes entre los procesadores.

Idealmente, los procesos y los procesadores que ejecutan dichos procesos pueden ser vistos como entidades completamente separadas. Un problema se describe como un conjunto de procesos que se comunican entre sí y que se hacen encajar sobre una estructura física de procesadores. El conocimiento de la estructura física y de la composición de los nodos es necesaria únicamente a la hora de planificar una ejecución eficiente.

El tamaño de un proceso viene determinado por el programador y puede describirse por su *granularidad*, que puede expresarse de manera informal como la razón:

$$Granularidad = \frac{Tiempo\ de\ cálculo}{Tiempo\ de\ comunicación}$$

o mediante los términos:

1. Granularidad gruesa.
2. Granularidad media.
3. Granularidad fina.

En la granularidad gruesa, cada proceso contiene un gran número de instrucciones secuenciales que tardan un tiempo sustancial en ejecutarse. En la granularidad fina, un proceso puede tener unas pocas instrucciones, incluso una instrucción; la granularidad media describe el terreno intermedio entre ambos extremos. Al reducirse la granulari-

dad, la sobrecarga de comunicación de los procesos suele aumentar. Es particularmente deseable reducir la sobrecarga de comunicación debido a que el coste de la misma suele ser bastante alto. Por ello, la granularidad empleada en este tipo de máquinas suele ser media o gruesa.

Cada nodo del sistema suele tener una copia del núcleo de un sistema operativo. Este sistema operativo se encarga de la planificación de procesos y de realizar las operaciones de paso de mensajes en tiempo de ejecución. Las operaciones de encaminamiento de los mensajes suelen estar soportadas por hardware, lo que reduce la latencia de las comunicaciones. Todo el sistema suele estar controlado por un ordenador anfitrión.

Existen desventajas en los sistemas multicomputador. El código y los datos deben de transferirse físicamente a la memoria local de cada nodo antes de su ejecución, y esta acción puede suponer una sobrecarga considerable. De forma similar, los resultados deben de transferirse de los nodos al sistema anfitrión. Claramente los cálculos a realizar deben ser lo suficientemente largos para compensar este inconveniente. Además, el programa a ejecutar debe ser intensivo en cálculo, no intensivo en operaciones de entrada/salida o de paso de mensajes. El código no puede compartirse. Si los procesos van a ejecutar el mismo código, lo que sucede frecuentemente, el código debe duplicarse en cada nodo. Los datos deben pasarse también a todos los nodos que los necesiten, lo que puede dar lugar a problemas de incoherencia. Estas arquitecturas son generalmente menos flexibles que las arquitecturas de memoria compartida. Por ejemplo, los multiprocesadores de memoria compartida pueden emular el paso de mensajes utilizando zonas compartidas para almacenar los mensajes, mientras que los multicomputadores son muy ineficientes a la hora de emular operaciones de memoria compartida.

3.1 Redes de interconexión para multicomputadores

Antes de ver las redes más comunes en sistemas multicomputadores, conviene repasar los distintos tipos de redes utilizadas en sistemas paralelos. La figura 3.2 muestra una clasificación que integra la mayoría de las redes comúnmente utilizadas en sistemas de altas prestaciones.

Las redes directas, también llamadas estáticas, son las más extendidas para la comunicación entre los elementos de un multicomputador. Las redes estáticas utilizan enlaces directos que son fijos una vez construida la red. Este tipo de red viene mejor para construir ordenadores donde los patrones de comunicación son predecibles o realizables mediante conexiones estáticas. Se describen a continuación las principales topologías estáticas en términos de los parámetros de red comentando sus ventajas en relación a las comunicaciones y la escalabilidad.

La mayoría de las redes directas implementadas en la práctica tienen una topología ortogonal. Una topología se dice *ortogonal* si y sólo si los nodos pueden colocarse en un espacio ortogonal n-dimensional, y cada enlace puede ponerse de tal manera que produce un desplazamiento en una única dimensión. Las topologías ortogonales pueden clasificarse a su vez en estrictamente ortogonales y débilmente ortogonales. En una topología *estrictamente ortogonal*, cada nodo tiene al menos un enlace cruzando cada dimensión. En una topología *débilmente ortogonal*, algunos de los nodos pueden no tener enlaces en alguna dimensión.

- Redes de medio compartido
  - Redes de área local
    - Bus de contención (Ethernet)
    - Bus de tokens (Arenet)
    - Anillo de tokens (FDDI Ring, IBM Token Ring)
  - Bus de sistema (Sun Gigaplane, DEC AlphaServer8X00, SGI PoweerPath-2)
- Redes directas (Redes estáticas basadas en encaminador)
  - Topologías estrictamente ortogonales
    - Malla
      - Malla 2-D (Intel Paragon)
      - Malla 3-D (MIT J-Machine)
    - Toros (n-cubo k-arios)
      - Toro 1-D unidireccional o anillo (KSR forst-level ring)
      - Toro 2-D bidireccional (Intel/CMU iWarp)
      - Toro 2-D bidireccional (Cray T3D, Cray T3E)
    - Hipercubo (Intel iPSC, nCUBE)
  - Otras topologías directas: Árboles, Ciclos cubo-conectados, Red de Bruijn, Grafos en Estrella, etc.
- Redes Indirectas (Redes dinámicas basadas en conmutadores)
  - Topologías Regulares
    - Barra cruzada (Cray X/Y-MP, DEC GIGAswitch, Myrinet)
    - Redes de Interconexión Multietapa (MIN)
      - Redes con bloqueos
        MIN Unidireccionales (NEC Cenju-3, IBM RP3)
        MIN Bidireccional (IBM SP, TMC CM-5, Meiko CS-2)
      - Redes sin bloqueos: Red de Clos
  - Topologías Irregulares (DEC Autonet, Myrinet, ServerNet)
- Redes Híbridas
  - Buses de sistema múltiples (Sun XDBus)
  - Redes jerárquicas (Bridged LANs, KSR)
    - Redes basadas en Agrupaciones (Stanford DASH, HP/Convex Exemplar)
  - Otras Topologías Hipergrafo: Hiperbuses, Hipermallas, etc.

Figura 3.2: Clasificación de las redes de interconexión

3.1.1 Topologías estrictamente ortogonales

La característica más importante de las topologías estrictamente ortogonales es que el encaminamiento es muy simple. En efecto, en una topología estrictamente ortogonal los nodos pueden enumerarse usando sus coordenadas en el espacio n-dimensional. Dado que cada enlace atraviesa una única dimensión y que cada nodo tiene al menos un enlace atravesando cada dimensión, la distancia entre dos nodos puede calcularse como la suma de la diferencia en las dimensiones. Además, el desplazamiento a lo largo de un enlace dado sólo modifica la diferencia dentro de la dimensión correspondiente. Teniendo en cuenta que es posible cruzar cualquier dimensión desde cualquier nodo en la red, el encaminamiento se puede implementar fácilmente seleccionando un enlace que decremente el valor absoluto de la diferencia de alguna de las dimensiones. El conjunto de dichas diferencias pueden almacenarse en la cabecera del paquete, y ser actualizada (añadiendo o substrayendo una unidad) cada vez que el paquete se encamina en un nodo intermedio. Si la topología no es estrictamente ortogonal, el encaminamiento se complica.

Las redes directas más populares son las *mallas n-dimensionales*, los *n-cubos k-arios* o *toros*, y los *hipercubos*. Todas ellas son estrictamente ortogonales en su definición aunque hay ciertas variantes que no son estrictamente ortogonales pero se las engloba en el mismo grupo.

Mallas

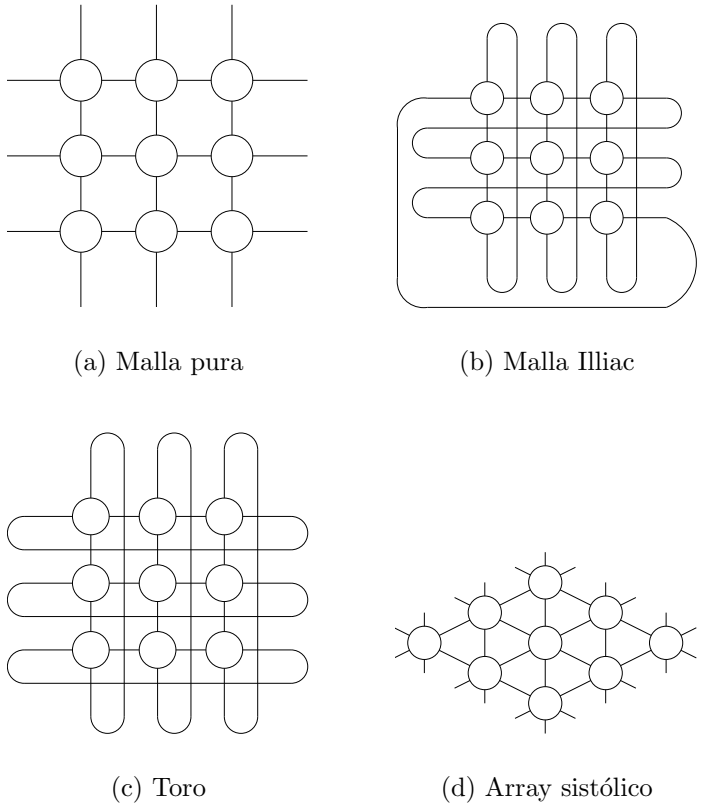
Formalmente, un malla n-dimensional tiene $k_0 \times k_1 \times \cdots \times k_{n-2} \times k_{n-1}$ nodos, k_i nodos en cada dimensión i, donde $k_i \geq 2$ y $0 \leq i \leq n-1$. Cada nodo X está definido por sus n coordenadas, $(x_{n-1}, x_{n-2}, \ldots, x_1, x_o)$, donde $0 \leq x_i \leq k_i - 1$ para $0 \leq i \leq n-1$. Dos nodos X e Y son vecinos si y sólo si $y_i = x_i$ para todo i, $0 \leq i \leq n-1$, excepto uno, j, donde $y_j = x_j \pm 1$. Así, los nodos pueden tener de n a $2n$ vecinos, dependiendo de su localización en la red. Por tanto, esta topología no es regular.

La figura 3.3(a) muestra una malla de 3×3. La estructura de malla, con algunas variaciones, se ha venido empleando en varios computadores comerciales, especialmente para la interconexión de procesadores en SIMD o masivamente paralelos.

En general, una malla k-dimensional, con $N = n^k$ nodos, tiene un grado de nodo interior de $2k$ y el diámetro de la red es $d = k(n-1)$. Hay que hacer notar que la malla pura, mostrada en la figura 3.3(a) no es simétrica y que los nodos en los bordes pueden ser 2 y 3 mientras que en el interior son siempre 4.

La figura 3.3(b) muestra una variación de una malla o toro en la que se permiten conexiones largas entre los nodos en los bordes. El Illiac IV tiene una malla con este principio ya que se trata de una malla de 8×8 con un grado nodal constante de 4 y un diámetro de 7. La malla Illiac es topológicamente equivalente a un anillo acorde de grado 4 como se muestra en la figura 3.7(c) para una configuración de $N = 9 = 3 \times 3$.

En general, una malla Illiac $n \times n$ debería tener un diámetro $d = n - 1$, que es sólo la mitad del diámetro de una malla pura. El *toro* de la figura 3.3(c) se puede ver como otra variante de la malla aunque se trata en realidad de un 2-cubo ternario, es decir, pertenece al grupo de las redes n-cubo k-arias que se muestran a continuación.



(a) Malla pura (b) Malla Illiac

(c) Toro (d) Array sistólico

Figura 3.3: Variaciones de mallas y toros.

Redes n-cubo k-arias

En un n-cubo k-ario, todos los nodos tienen el mismo número de vecinos. La definición del n-cubo k-ario difiere de la de malla n-dimensional en que todos los k_i son iguales a k y dos nodos X e Y son vecinos si y sólo si $y_i = x_i$ para todo i, $0 \leq i \leq n-1$, excepto uno, j, donde $y_j = (x_j \pm 1)\ mod\ k$. El cambio a aritmética modular en la dirección añade el canal entre el nodo $k-1$ y el 0 en cada dimensión en los n-cubos k-arios, dándole regularidad y simetría. Cada nodo tiene n vecinos si $k = 2$ y $2n$ vecinos si $k > 2$. Cuando $n = 1$ el n-cubo k-ario se colapsa en un anillo bidireccional con k nodos.

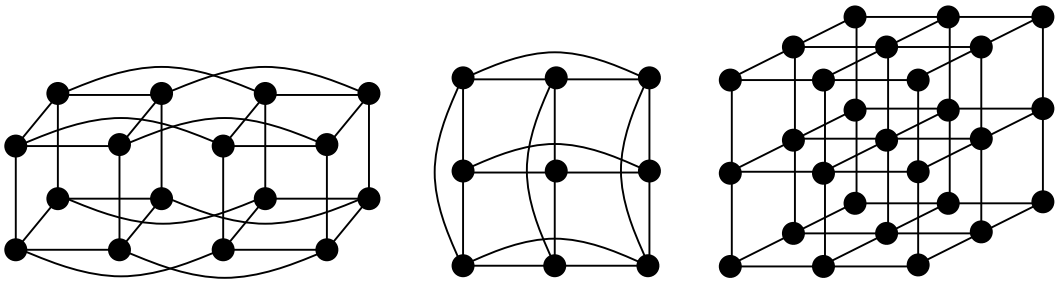
A las redes n-cubo k-aria también se les suele llamar toros. En general, un toro $n \times n$ binario tiene un grado nodal de 4 y un diámetro $d = 2\lfloor n/2 \rfloor$. El toro es una topología simétrica. Todas las conexiones de contorno añadidas ayudan a reducir el diámetro de la malla pura a la mitad.

Hipercubos

Otra topología con regularidad y simetría es el hipercubo, que es un caso particular de mallas n-dimensionales o n-cubo k-ario. Un hipercubo es una malla n-dimensional con $k_i = 2$ para $0 \leq i \leq n-1$, o un n-cubo binario.

La figura 3.4(a) muestra un hipercubo con 16 nodos. La parte (b) de dicha figura ilustra un 2-cubo ternario o toro bidimensional (2-D). La figura 3.4(c) muestra una

malla tridimensional.



(a) 2−ary 4−cube (hypercube) (b) 3−ary 2−cube (c) 3−ary 3−D mesh

Figura 3.4: Topologías estrictamente ortogonales en una red directa.

En la figura 3.5 se muestran diversas topologías del tipo n-cubo k-aria, incluyendo hipercubos de 3 y 4 dimensiones. También en esta figura se ha incluido el ciclo cubo-conectado que es un tipo especial de red directa no estrictamente ortogonal pero basada en el hipercubo de tres dimensiones.

3.1.2 Otras topologías directas

Aparte de las topologías definidas arriba, se han propuesto muchas otras topologías en la literatura. La mayoría de ellas fueron propuestas con la meta de minimizar el diámetro de la red para un número dado de nodos y grado del nodo. Como veremos en secciones posteriores, para estrategias de conmutación encauzadas, la latencia es casi insensible al diámetro de la red, especialmente cuando los mensajes son largos. Así que es poco probable que esas topologías lleguen a ser implementadas. En los siguientes párrafos, presentaremos una descripción informal de algunas topologías de red directas relevantes.

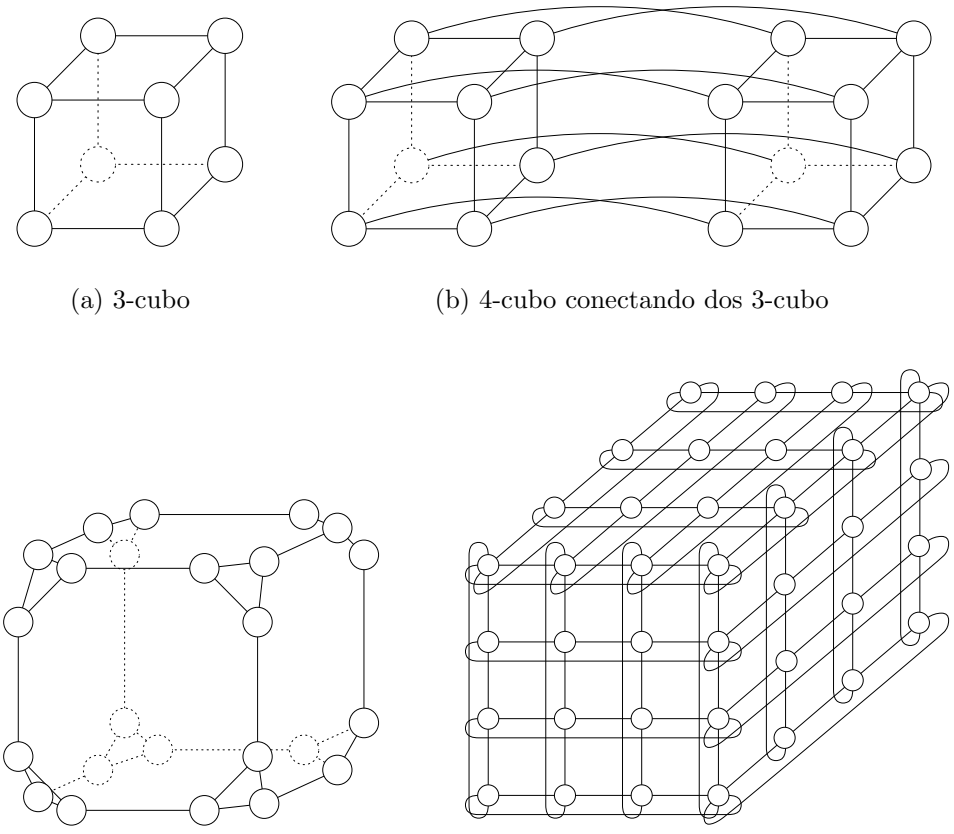
Las topologías que se presentan a continuación tienen diferentes objetivos: algunas intentan simplificar la red a costa de un mayor diámetro, otras pretenden disminuir el diámetro y aumentando los nodos manteniendo un compromiso de complejidad no muy elevada.

La figura 3.6 muestra otras topologías propuestas para reducir el grado del nodo a la vez que se mantiene un diámetro bajo.

Matriz lineal

Esta es una red monodimensional en la cual N nodos están conectados mediante $N-1$ enlaces tal y como se muestra en la figura 3.7(a). Los nodos internos son de grado 2 mientras que los terminales son de grado 1. El diámetro es $N-1$ que es excesivo si N es grande. La anchura biseccional b es 1. Estas matrices lineales son los más simples de implementar. La desventaja es que no es simétrica y que la comunicación es bastante ineficiente a poco que N sea algo grande, con $N=2$ el sistema está bien, pero para N más grande hay otras topologías mejores.

No hay que confundir la matriz lineal con el *bus* que es un sistema de tiempo compartido entre todos los nodos pegados a el. Una matriz lineal permite la utilización



(a) 3-cubo

(b) 4-cubo conectando dos 3-cubo

(c) 3 ciclo cubo conectado

(d) n-cubo k-ario con $k = 4$ y $n = 3$

Figura 3.5: Hipercubos, ciclo cubos y n-cubos k-arios.

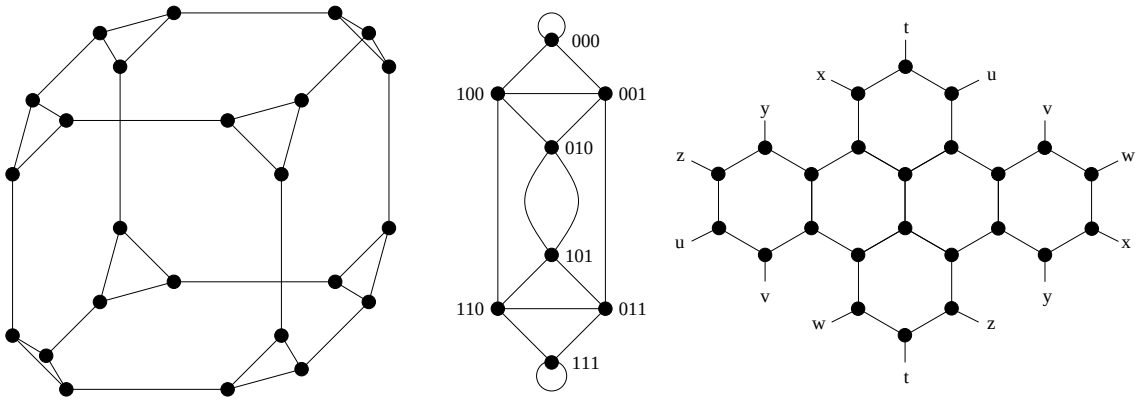
simultánea de diferentes secciones (canales) de la matriz con diferentes orígenes y destinos.

Anillo

Un *anillo* se obtiene conectando los dos nodos terminales de una matriz lineal mediante un enlace más como se muestra en la figura 3.7(b). El anillo puede ser unidireccional (diámetro $N - 1$) o bidireccional (diámetro $N/2$). El anillo es simétrico con todos los nodos de grado 2 constante.

Este tipo de topologías se han utilizado en el pasado como en el IBM *token ring* donde los mensajes circulaban hasta encontrar un nodo con un *token* que coincida con el mensaje. También en otros sistemas se ha utilizado para la comunicación entre procesadores por el paso de paquetes.

Si añadimos más enlaces a cada nodo (aumentamos el grado del nodo), entonces obtenemos *anillos acordes* como se muestra en la figura 3.7(c). Siguiendo esto se pueden ir añadiendo enlaces a los nodos aumentando la conectividad y reduciendo el diámetro de la red. En el extremo podemos llegar a la *red completamente conectada* en la cual los nodos son de grado $N - 1$ y el diámetro es 1.



(a) Cube-Connected Cycles (b) De Bruijn Network (c) Star Graph

Figura 3.6: Otras topologías directas.

Desplazador de barril

El desplazador de barril *barrel shifter* se obtiene a partir de la configuración en anillo, añadiendo a cada nodo enlaces con los nodos que tengan una distancia potencia de 2 como se muestra en la figura 3.7(d). Esto implica que cada nodo i está conectado con un nodo j si se cumple que $|j - i| = 2^r$ para algún $r = 0, 1, \ldots, n-1$ siendo el tamaño de la red $N = 2^n$. El grado de cada nodo es $d = 2n - 1$ y un diámetro $D = n/2$.

Naturalmente, la conectividad del desplazador de barril es mayor que la de cualquier anillo acorde con un grado de nodo menor. Suponiendo $N = 16$, el desplazador de barril tiene un grado de nodo igual a 7 con un diámetro de 2. A pesar de todo la complejidad del desplazador de barril es menor que la del anillo completamente conectado.

Ciclo Cubo Conectado

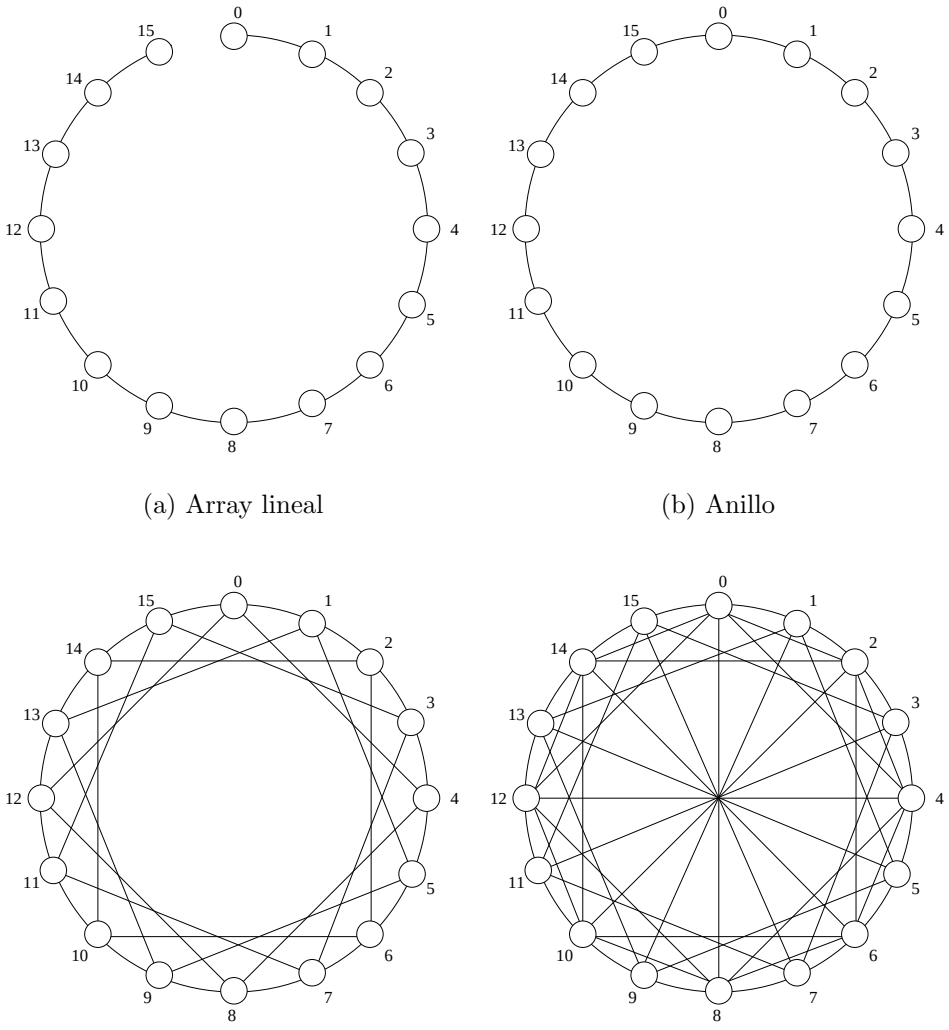
Esta arquitectura se realiza a partir del hipercubo y consiste en sustituir cada vértice del cubo por un anillo (ciclo) normalmente con un número igual de nodos que de dimensiones del cubo. Las figuras 3.5(c) y 3.6(a) muestran un 3-ciclo cubo conectado (un 3-CCC).

En general uno puede construir un k-ciclo cubo conectado a partir de un k-cubo con $n = 2^k$ ciclos. La idea es reemplazar cada vértice del hipercubo k-dimensional por un anillo de k nodos. Un k-cubo puede ser transformado entonces en un k-CCC con $k \times 2^k$ nodos.

El diámetro de un k-CCC es $2k$, es decir, el doble que el del hipercubo. La mejora de esta arquitectura está en que el grado de nodo es siempre 3 independientemente de la dimensión del hipercubo, por lo que resulta una arquitectura escalable.

Supongamos un hipercubo con $N = 2^n$ nodos. Un CCC con el mismo número N de nodos se tendría que construir a partir de un hipercubo de menor dimensión tal que $2^n = k \cdot 2^k$ para algún $k < n$.

Por ejemplo, un CCC de 64 nodos se puede realizar con un 4-cubo reemplazando los vértices por ciclos de 4 nodos, que corresponde al caso $n = 6$ y $k = 4$. El CCC tendría un diámetro de $2k = 8$ mayor que 6 en el 6-cubo. Pero el CCC tendría un grado



(a) Array lineal

(b) Anillo

(c) Anillo acorde de grado 4 (coincide con la malla del Illiac)

(d) Desplazador de barril

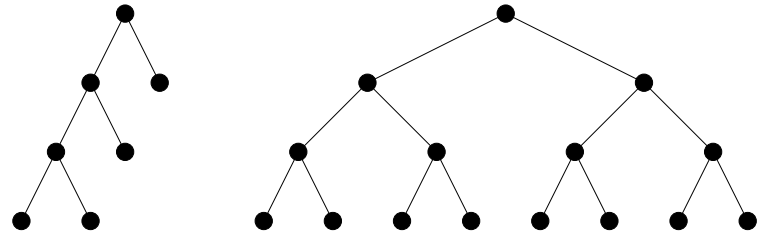
Figura 3.7: Matriz lineal, anillos, y barril desplazado.

nodal de 3, menor que 6 en el 6-cubo. En este sentido, el CCC es una arquitectura más conveniente para fabricar sistemas escalables siempre que la mayor latencia pueda ser tolerada de alguna manera.

Árbol, árbol grueso y estrella

Una topología popular es el *árbol.* Esta topología tiene un nodo *raíz* conectado a un cierto número de nodos descendientes. Cada uno de estos nodos se conecta a la vez a un conjunto disjunto (posiblemente vacío) de descendientes. Un nodo sin descendientes es un nodo *hoja.* Una propiedad característica de los árboles es que cada nodo tiene un único padre. Por tanto, los árboles no tienen ciclos. Un árbol en el cual cada nodo menos las hojas tiene un número k fijo de descendientes es un árbol k-ario. Cuando la distancia entre cada nodo hoja y la raíz es la misma, es decir, todas las ramas del árbol

tienen la misma longitud, el árbol está *balanceado*. Las figura 3.8(a) y 3.8(b) muestran un árbol binario no balanceado y balanceado, respectivamente.



(a) Binary Tree (b) Balanced Binary Tree

Figura 3.8: Algunas topologías árbol.

La desventaja más importante de los árboles como redes de interconexión en general es que el nodo raíz y los nodos cerca de él se convierten en cuellos de botella. Además, no existen caminos alternativos entre cualquier par de nodos. El cuello de botella puede eliminarse utilizando canales con mayor ancho de banda en los canales cercanos al nodo raíz. Sin embargo, el uso de canales de diferente ancho de banda no es práctico, especialmente para la transmisión de mensajes encauzada. Una forma práctica de implementar árboles con canales de mayor ancho de banda en la vecindad del nodo raíz son los árboles gruesos (*fat trees*).

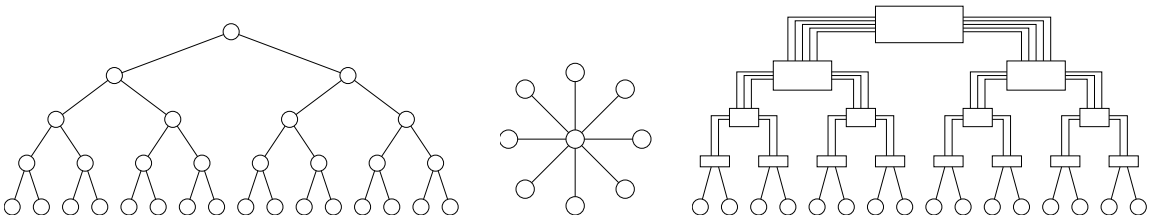
La estructura convencional de árbol utilizada en ciencias de la computación, puede ser modificada para conseguir un *árbol grueso*. Un árbol grueso binario se muestra en la figura 3.9(c). La anchura de canal de un árbol grueso se incrementa conforme se sube de las hojas a la raíz. El árbol grueso es más parecido a un árbol real donde las ramas son más gruesas conforme nos acercamos a la raíz.

El árbol grueso se introdujo para aliviar un problema grave de los árboles binarios convencionales, que consiste en el cuello de botella que se produce cerca de la raíz ya que el tráfico en esta zona es más intenso. El árbol grueso se ha utilizado, y se utiliza, en algunos supercomputadores. La idea del árbol grueso binario puede extenderse a árboles gruesos multivía.

Una de las propiedades más interesantes de los árboles es que, para cualquier grafo conectado, es posible definir un árbol dentro del grafo. Como consecuencia, para cualquier red conectada, es posible construir una red acíclica conectando todos los nodos eliminando algunos enlaces. Esta propiedad puede usarse para definir un algoritmo de encaminamiento para redes irregulares. Sin embargo, este algoritmo de encaminamiento puede ser ineficiente debido a la concentración del tráfico alrededor del nodo raíz.

Un *árbol binario* con 31 nodos y 5 niveles se muestra en la figura 3.9(a). En general, un árbol completamente balanceado de k niveles tiene $N = 2^k - 1$ nodos. El grado máximo de nodo en un árbol binario es 3 y el diámetro es $2(k-1)$. Si el grado de los nodos es constante, entonces el árbol es fácilmente escalable. Un defecto del árbol es que el diámetro puede resultar demasiado grande si hay muchos nodos.

La *estrella* es un árbol de dos niveles con un alto grado de nodo que es igual a $d = N - 1$ como se muestra en la figura 3.9(b). El diámetro resultante es pequeño, contante e igual a 2. La estructura en estrella se suele utilizar en sistemas con un supervisor que hace de nodo central.



(a) Árbol binario (b) Estrella (c) Árbol grueso binario

Figura 3.9: Árbol, estrella y árbol grueso.

Matrices sistólicas

Este es un tipo de arquitectura segmentada multidimensional diseñada para la realización de algoritmos específicos fijos. Por ejemplo, la red de la figura 3.3(d) corresponde a una matriz sistólica especialmente diseñada para la multiplicación de dos matrices. En este caso el grado de los nodos interiores es 6.

Con la interconexión fija y el funcionamiento síncrono de los nodos, la matriz sistólica encaja con la estructura de comunicación del algoritmo a realizar. Para aplicaciones específicas como el tratamiento de imágenes o señales, las matrices sistólicas pueden ofrecer una mejor relación entre rendimiento y coste. Sin embargo, la estructura puede tiene una aplicabilidad limitada y puede resultar muy difícil de programar.

3.1.3 Conclusiones sobre las redes directas

En la tabla 3.1 se resumen las características más importantes de las redes estáticas vistas hasta ahora. El grado nodal de la mayoría de redes es menor que 4, que está bastante bien. Por ejemplo, el Transputer chip de INMOS en un microprocesador que lleva incorporado la lógica de intercomunicación con un total de 4 puertos. Con un grado nodal constante igual a 4, un Transputer se puede utilizar como un bloque de construcción.

Los grados nodales de la estrella y de la red completamente conectada son ambos malos. El grado del nodo del hipercubo crece con $\log_2 N$ siendo también poco recomendable cuando N se hace grande.

Los diámetros de la red varían en un margen amplio. Con la invención del encaminamiento hardware (encaminado de agujero de gusano o *wormhole routing*), el diámetro ya no es un parámetro tan crítico ya que el retraso en la comunicación entre dos nodos cualquiera se ha convertido en algo casi constante con un alto nivel de segmentación. El número de enlaces afecta el coste de la red. La anchura biseccional afecta al ancho de banda.

La propiedad de simetría afecta a la escalabilidad y a la eficiencia en el rutado. Es justo decir que el coste total de la red crece con d (diámetro) y l (número de enlaces), por tanto, un diámetro pequeño es aun una virtud, pero la distancia media entre nodos puede ser una mejor medida. La anchura de banda biseccional puede mejorarse con un canal más ancho. Tomando el análisis anterior, el anillo, la malla, el toro, k-aria n-cubo, y el CCC todos tienen características propicias para construir los futuros sistemas MPP

Tipo de red	Grado nodal (d)	Diámetro de la red (D)	Número Enlaces (l)	Anchura bisección (B)	Simetría	Notas sobre el tamaño
Array lineal	2	$N-1$	$N-1$	1	No	N nodos
Anillo	2	$\lfloor N/2 \rfloor$	N	2	Sí	N nodos
Conectado completo	$N-1$	1	$N(N-1)/2$	$(N/2)^2$	Sí	N nodos
Árbol binario	3	$2(h-1)$	$N-1$	1	No	Altura árbol $h = \lceil \log_2 N \rceil$
Estrella	$N-1$	2	$N-1$	$\lfloor N/2 \rfloor$	No	N nodos
Malla 2D	4	$2(r-1)$	$2N-2r$	r	No	Malla $r \times r$ con $r = \sqrt{N}$
Malla Illiac	4	$r-1$	$2N$	$2r$	No	Equivalente al acorde con $r = \sqrt{N}$
Toro 2D	4	$2\lfloor r/2 \rfloor$	$2N$	$2r$	Sí	Toro $r \times r$ con $r = \sqrt{N}$
Hipercubo	n	n	$nN/2$	$N/2$	Sí	N nodos, $n = \log_2 N$ (dimensión)
CCC	3	$2k-1+\lfloor k/2 \rfloor$	$3N/2$	$N/(2k)$	Sí	$N = k \times 2^k$ nodos con longitud de ciclo $k \geq 3$
k-aria n-cubo	$2n$	$n\lfloor k/2 \rfloor$	nN	$2k^{n-1}$	Sí	$N = k^n$ nodos

Tabla 3.1: Resumen de las características de las redes estáticas.

(Procesadores Masivamente Paralelos).

En definitiva, con la utilización de técnicas segmentadas en el encaminamiento, la reducción del diámetro de la red ya no es un objetivo primordial. Cuestiones como la facilidad de encaminamiento, la escalabilidad y la facilidad para ampliar el sistema pueden ser actualmente temas más importantes, por lo que las redes estrictamente ortogonales se imponen en los multicomputadores actuales.

3.2 La capa de conmutación o control de flujo (*switching*)

En esta sección nos centraremos en las técnicas que se implementan dentro de los encaminadores (*routers*) para realizar el mecanismo por el cual los mensajes pasan a través de la red. Estas técnicas difieren en varios aspectos. Las *técnicas de conmutación* determinan cuándo y cómo se conectan los conmutadores internos del encaminador para conectar las entradas y salidas del mismo, así como cuándo los componentes del mensaje pueden transferirse a través de esos caminos. Estás técnicas están ligadas a los mecanismos de *control de flujo* que sincronizan la transferencia de unidades de información entre encaminadores y a través de los mismos durante el proceso de envío de los mensajes a través de la red. El control de flujo está a su vez fuertemente acoplado a los algoritmos de *manejo de buffers* que determinan cómo se asignan y liberan los buffers, determinando como resultado cómo se manejan los mensajes cuando se bloquean en la red.

Las implementaciones del nivel de conmutación difieren en las decisiones que se realizan en cada una de estas áreas, y su temporización relativa, es decir, cuándo una operación puede comenzar en relación con la ocurrencia de otra. Las elecciones específicas interactúan con la arquitectura de los encaminadores y los patrones de tráfico impuestos por los programas paralelos para determinar las características de latencia y el rendimiento de la red de interconexión.

3.2.1 Elementos básicos de la conmutación

El control de flujo es un protocolo asíncrono para transmitir y recibir una unidad de información. La *unidad de control de flujo* (flit) se refiere a aquella porción del mensaje cuya transmisión debe sincronizarse. Esta unidad se define como la menor unidad de información cuya transferencia es solicitada por el emisor y notificada por el receptor. Esta señalización *request/acknowledgment* se usa para asegurar una transferencia exitosa y la disponibilidad de espacio de buffer en el receptor. Obsérvese que estas transferencias son atómicas en el sentido de que debe asegurarse un espacio suficiente para asegurar que el paquete se transfiere en su totalidad.

El control de flujo ocurre a dos niveles. El *control de flujo del mensaje* ocurre a nivel de paquete. Sin embargo, la transferencia del paquete por el canal físico que une dos encaminadores se realiza en varios pasos, por ejemplo, la transferencia de un paquete de 128-bytes a través de una canal de 16-bits. La transferencia resultante multiciclo usa un *control del flujo del canal* para enviar un flit a través de la conexión física.

Las técnicas de conmutación suelen diferenciarse en la relación entre el tamaño de la unidad de control física y del paquete. En general, cada mensaje puede dividirse en *paquetes* de tamaño fijo. Estos paquetes son a su vez divididos en unidades de control de flujo o flits. Debido a las restricciones de anchura del canal, puede ser necesario varios ciclos de canal para transferir un único flit. Un *phit* es la unidad de información que puede transferirse a través de un canal físico en un único paso o ciclo. Los flits representan unidades lógicas de información en contraposición con los phits que se corresponden a cantidades físicas, es decir, número de bits que pueden transferirse en paralelo en un único ciclo. La figura 3.10 muestra N paquetes, 6 flits/paquete y 2 phits/flit.

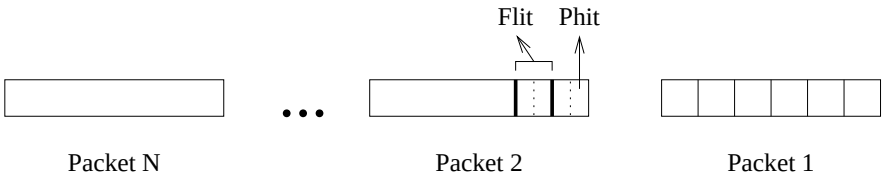


Figura 3.10: Distintas unidades de control de flujo en un mensaje.

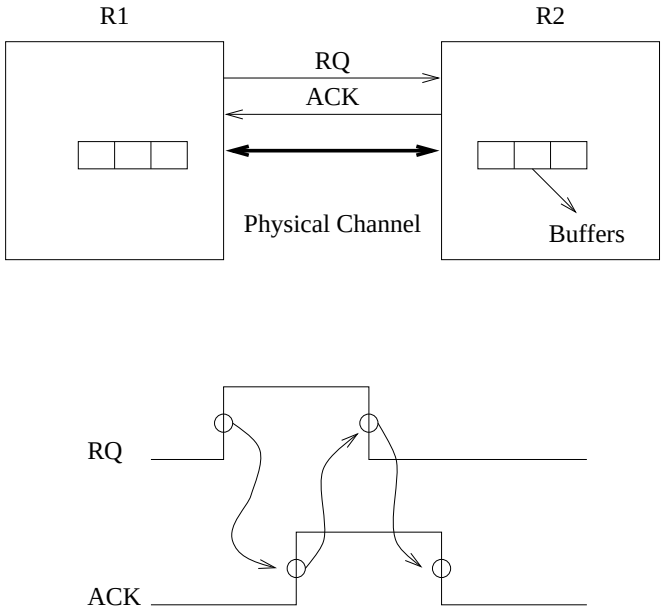


Figura 3.11: Un ejemplo de control de flujo asíncrono de un canal físico.

Existen muchos candidatos a protocolo de sincronización para coordinar la transferencia de bits a través de un canal. La figura 3.11 ilustra un ejemplo de protocolo asíncrono de cuatro fases. El encaminador $R1$ pone a uno la señal RQ antes de comenzar la transferencia de información. El encaminador $R2$ responde leyendo los datos y activando la señal ACK. Esto da lugar a la desactivación de RQ por parte de $R1$ que causa la desactivación de ACK por $R2$. La señal ACK se utiliza tanto para confirmar la recepción (flanco ascendente) como para indicar la existencia de espacio de buffer (flanco descendente) para la siguiente transferencia.

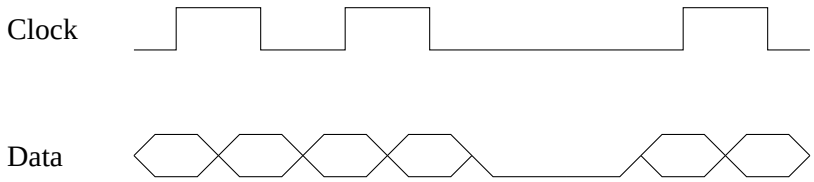


Figura 3.12: Un ejemplo de control de flujo síncrono de un canal físico.

El control de flujo del canal también puede ser síncrono, como se muestra en la figura 3.12. La señal de reloj se transmite por el canal y ambos flancos de la señal de reloj se utilizan para validar las líneas de datos en el receptor. La figura 3.12 no muestra las señales ACK utilizadas para indicar la existencia de espacio en el nodo receptor.

Mientras que las transferencias inter-encaminador deben realizarse necesariamente en términos de phits, las técnicas de conmutación manejan flits (que pueden definirse hasta llegar a tener el tamaño del paquete completo). Las técnicas de conmutación manejan el conmutador interno para conectar buffers de entrada con buffers de salida, y enviar los flits a través de este camino. Estas técnicas se distinguen por el instante en que ocurren en relación con la operación de control de flujo y la operación de encaminamiento. Por ejemplo, la conmutación puede tener lugar después de que un flit haya sido recibido completamente. Alternativamente, la transferencia de un flit a través del conmutador puede empezar en cuanto finaliza la operación de encaminamiento, pero antes de recibir el resto del flit desde el encaminador anterior. En este caso la conmutación se solapa con el control de flujo a nivel de mensaje. En una última técnica de conmutación propuesta, la conmutación comienza después de recibir el primer phit, antes incluso de que haya finalizado la operación de encaminamiento.

Modelo de encaminador

A la hora de comparar y contrastar diferentes alternativas de conmutación, estamos interesados en cómo las mismas influyen en el funcionamiento del encaminador y, por tanto, la latencia y ancho de banda resultante. La arquitectura de un encaminador genérico se muestra en la figura 3.13 y está compuesto de los siguientes elementos principales:

- *Buffers*. Son buffers FIFO que permiten almacenar mensajes en tránsito. En el modelo de la figura 3.13, un buffer está asociado con cada canal físico de entrada y de salida. En diseños alternativos, los buffers pueden asociarse únicamente a las entradas o a las salidas. El tamaño del buffer es un número entero de unidades de control de flujo (*flits*).
- *Conmutador* . Este componente es el responsable de conectar los buffers de entrada del encaminador (*router*) con los buffers de salida. Los encaminadores de alta

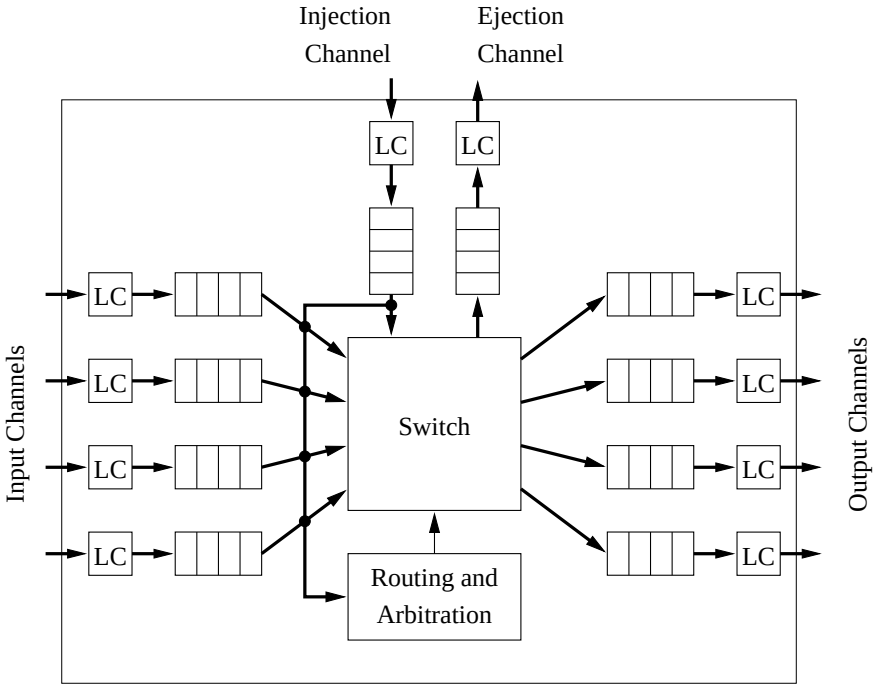


Figura 3.13: Modelo de encaminador (*router*) genérico. (LC = Link controller.)

velocidad utilizan redes de barra cruzada (*crossbar*) con conectividad total, mientras que implementaciones de más baja velocidad utilizan redes que no proporcionan una conectividad total entre los buffers de entrada y los de salida.

- *Unidad de encaminamiento y arbitraje.* Este componente implementa los algoritmos de encaminamiento, selecciona el enlace de salida para un mensaje entrante, programando el conmutador en función de la elección. Si varios mensajes piden de forma simultánea el mismo enlace de salida este componente debe proporcionar un arbitraje entre ellos. Si el enlace pedido está ocupado, el mensaje debe permanecer en el buffer de entrada hasta que éste quede libre.
- *Controladores de enlace (LC).* El flujo de mensajes a través de los canales físicos entre encaminadores adyacentes se implementa mediante el LC. Los controladores de enlace de cada lado del canal deben coordinarse para transferir flits.
- *Interfaz del procesador.* Este componente simplemente implementa un canal físico con el procesador en lugar de con un encaminador adyacente. Consiste en uno o más canales de inyección desde el procesador y uno o más canales de eyección hacia el procesador. A los canales de eyección también se les denominan canales de reparto o canales de consumición.

Desde el punto de vista del rendimiento del encaminador (*router*) estamos interesados en dos parámetros. Cuando un mensaje llega a un encaminador, éste debe ser examinado para determinar el canal de salida por el cual se debe enviar el mensaje. A esto se le denomina *retraso de encaminamiento* (*routing delay*), y suele incluir el tiempo para configurar el conmutador. Una vez que se ha establecido un camino a través del encaminador, estamos interesados en la velocidad a la cual podemos enviar mensajes a través del conmutador. Esta velocidad viene determinado por el retraso de propagación a través de conmutador (retraso intra-encaminadores), y el retraso que permite la sincronización de la transferencia de datos entre los buffers de entrada y de

salida. A este retraso se le denomina latencia de *control de flujo interno*. De manera similar, al retraso a través de los enlaces físicos (retraso inter-encaminadores) se le denomina latencia del *control de flujo externo*. El retraso debido al encaminamiento y los retrasos de control de flujo determinan la latencia disponible a través del conmutador y, junto con la contención de los mensajes en los enlaces, determina el rendimiento o productividad de la red (*throughput*).

Técnicas de conmutación

Antes de comenzar a estudiar las distintas técnicas de conmutación, es necesario tener en cuenta algunas definiciones. Para cada técnica de conmutación se considerará el cálculo de la latencia base de un mensaje de L bits en ausencia de tráfico. El tamaño del *phit* y del *flit* se supondrán equivalentes e iguales al ancho de un canal físico (W bits). La longitud de la cabecera se supondrá que es de 1 flit, así el tamaño el mensaje será de $L+W$. Un encaminador (*router*) puede realizar una decisión de encaminamiento en t_r segundos. El canal físico entre 2 encaminadores opera a B Hz, es decir, el ancho de banda del canal físico es de BW bits por segundo. Al retraso de propagación a través de este canal se denota por $t_w = \frac{1}{B}$. Una vez que sea establecido un camino a través de un encaminador, el retraso intra-encaminador o retraso de conmutación se denota por t_s. El camino establecido dentro del encaminador se supone que coincide con el ancho del canal de W bits. Así, en t_s segundos puede transferirse un flit de W bits desde la entrada a la salida del encaminador. Los procesadores origen y destino constan de D enlaces. La relación entre estos componentes y su uso en el cálculo de la latencia de un mensaje bajo condiciones de no carga se muestra en la figura 3.14.

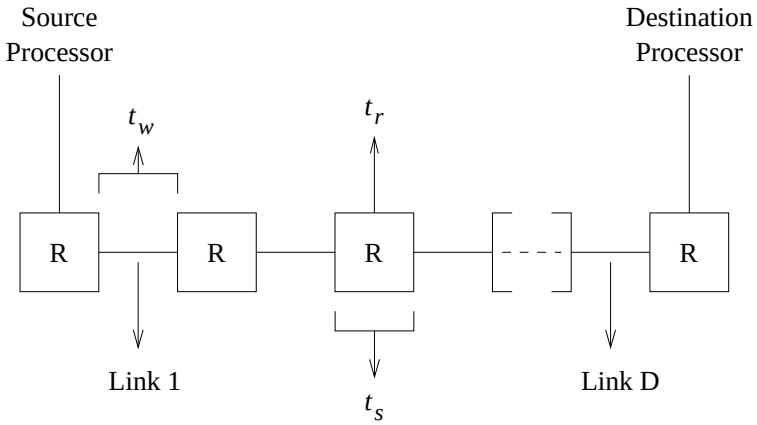


Figura 3.14: Cálculo de la latencia en una red para el caso de ausencia de carga (R = Encaminador o *Router*).

3.2.2 Conmutación de circuitos

En la conmutación de circuitos, se reserva un camino físico desde el origen hasta el destino antes de producirse la transmisión de los datos. Esto se consigue mediante la inyección de un flit cabecera en la red. Esta trama de sondeo del encaminamiento contiene la dirección del destino e información de control adicional. La misma progresa hacia el destino reservando los enlaces físicos conforme se van transmitiendo a través

de los encaminadores intermedios. Cuando alcanza el destino, se habrá establecido un camino, enviándose una señal de asentimiento de vuelta al origen. El contenido del mensaje puede ahora ser emitido a través del camino hardware. El circuito puede liberarse por el destino o por los últimos bits del mensaje. En la figura 3.15 se muestra un cronograma de la transmisión de un mensaje a través de tres enlaces. La cabecera se envía a través de los tres enlaces seguida de la señal de asentimiento. En la figura 3.16 se muestra un ejemplo del formato de la trama de creación de un circuito.

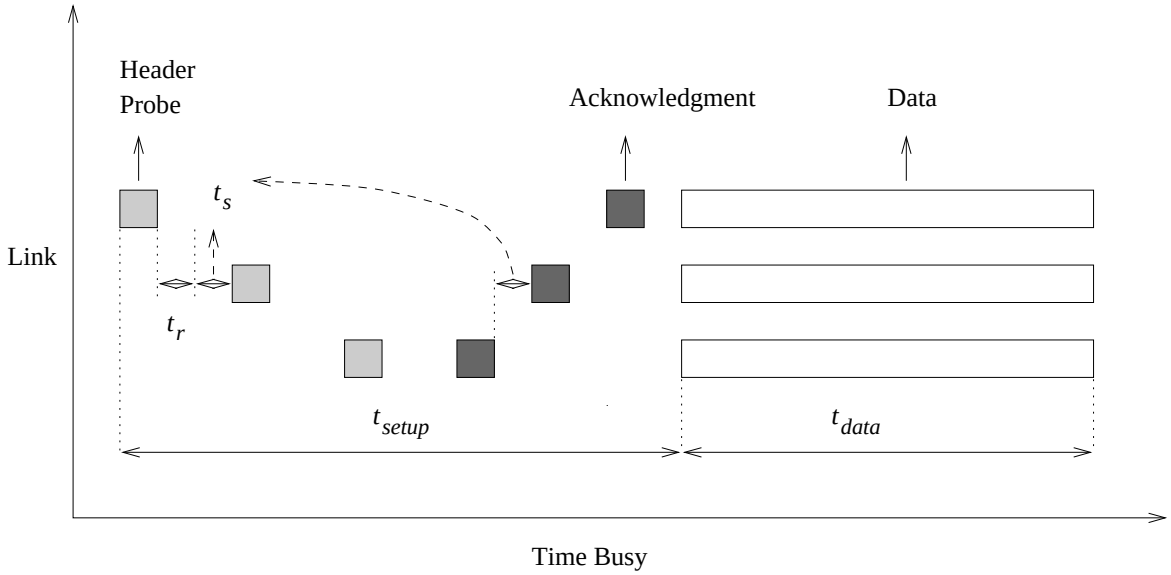


Figura 3.15: Cronograma de un mensaje por conmutación de circuitos.

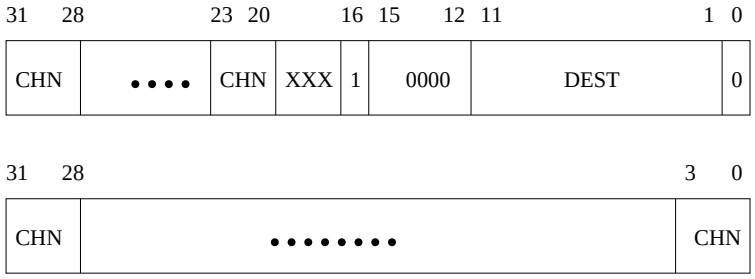


Figura 3.16: Un ejemplo del formato en una trama de creación de un circuito. (CHN = Número de canal; DEST = Dirección destino; XXX = No definido.)

El mejor comportamiento de esta técnica de conmutación ocurre cuando los mensajes son infrecuentes y largos, es decir, cuando el tiempo de transmisión del mensaje es largo en comparación con el tiempo de establecimiento del circuito. La desventaja es que el camino físico se reserva para toda la duración del mensaje y puede bloquear a otros mensajes. Por ejemplo, consideremos el caso en donde la trama de sondeo está esperando a que un enlace físico esté libre. Todos los enlaces reservados por la trama hasta ese punto permanecen reservados, no pueden ser utilizados por otros mensajes, y pueden bloquear el establecimiento de otros circuitos. Así, si el tamaño del mensaje no es mucho más grande que el tamaño de la trama de sondeo, sería ventajoso transmitir el mensaje junto con la cabecera y almacenar el mensaje dentro de los encaminadores

mientras que se espera a que se libere un enlace. A esta técnica alternativa se le denomina conmutación de paquetes.

La latencia base un mensaje en esta técnica de conmutación está determinado por el tiempo de establecimiento de un camino, y el tiempo posterior en el que el camino está ocupado transmitiendo los datos. El modo de funcionamiento de encaminador difiere un poco el mostrado en la figura 3.13. Mientras que la trama de sondeo se almacena en cada encaminador, los datos no son almacenados. No hay buffers y el circuito se comporta como si se tratase de un único enlace entre el origen y destino.

La expresión de la latencia base para un mensaje es la siguiente:

$$\begin{aligned} t_{circuit} &= t_{setup} + t_{data} \\ t_{setup} &= D[t_r + 2(t_s + t_w)] \\ t_{data} &= \frac{1}{B}\left\lceil \frac{L}{W} \right\rceil \end{aligned} \tag{3.1}$$

3.2.3 Conmutación de paquetes

En la conmutación de circuitos, la totalidad del mensaje se transmite después de que se haya restablecido el circuito. Una alternativa sería que el mensaje pudiese dividirse y transmitirse en paquetes de longitud fija, por ejemplo, 128 bytes. Los primeros bytes del paquete contendrían la información de encaminamiento y constituirían lo que se denomina cabecera paquete. Cada paquete se encamina de forma individual desde el origen al destino. Un paquete se almacena completamente en cada nodo intermedio antes de ser enviado al siguiente nodo. Esta es la razón por la que a este método de conmutación también se le denomina conmutación de *almacenamiento y reenvío* (SAF). La información de la cabecera se extrae en los encaminadores intermedios y se usa para determinar el enlace de salida por el que se enviará el paquete. En la figura 3.17 se muestra un cronograma del progreso de un paquete a lo largo de tres enlaces. En esta figura podemos observar que la latencia experimentada por un paquete es proporcional a la distancia entre el nodo origen y destino. Observar que en la figura se ha omitido la latencia del paquete a través del encaminador.

Esta técnica es ventajosa cuando los mensajes son cortos y frecuentes. Al contrario que en la conmutación de circuitos, donde un segmento de un camino reservado puede estar sin ser usado durante un período de tiempo significativo, un enlace de comunicación se usa completamente cuando existen datos que transmitir. Varios paquetes pertenecientes a un mensaje pueden estar en la red simultáneamente incluso si el primer paquete no ha llegado todavía al destino. Sin embargo, dividir un mensaje en paquetes produce una cierta sobrecarga. Además del tiempo necesario en los nodos origen y destino, cada paquete debe ser encaminado en cada nodo intermedio. En la figura 3.18 se muestra un ejemplo del formato de la cabecera del paquete.

La latencia base de un mensaje en conmutación de paquetes se puede calcular como:

$$t_{packet} = D\left\{ t_r + (t_s + t_w)\left\lceil \frac{L+W}{W} \right\rceil \right\} \tag{3.2}$$

Esta expresión sigue el modelo de encaminador (*router*) mostrado en la figura 3.13,

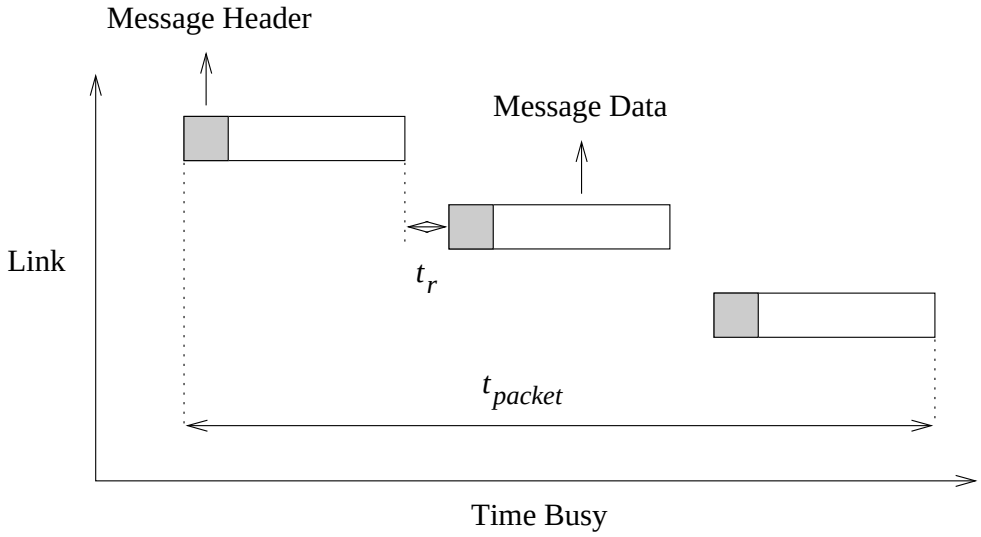


Figura 3.17: Cronograma de un mensaje por conmutación de paquetes.

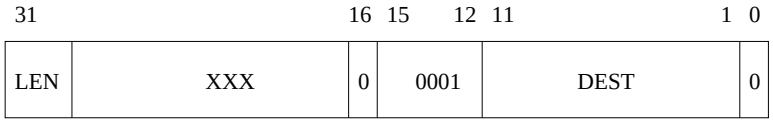


Figura 3.18: Un ejemplo de formato de la cabecera del paquete. (DEST = Dirección destino; LEN = Longitud del paquete en unidades de 192 bytes; XXX = No definido.)

y es el resultado de incluir los factores que representan el tiempo de transferencia del paquete de longitud $L + W$ a través del canal (t_w) así como del almacenamiento en la entrada y la transferencia desde los buffers de entrada a los de salida del encaminador.

3.2.4 Conmutación de paso a través virtual, *Virtual Cut-Through* (VCT)

La conmutación de paquete se basa en suponer que un paquete debe recibirse en su globalidad antes de poder realizar cualquier decisión de encaminamiento y antes de continuar el envío de paquete hacia su destino. En general, esto no es cierto. Consideremos un paquete de 128 bytes y el modelo de encaminador mostrado en la figura 3.13. En ausencia de canales físicos de ciento veintiocho bytes de ancho, la transferencia del paquete a través del canal físico llevará varios ciclos. Sin embargo, los primeros bytes contienen la información de encaminamiento que estará disponible después de los primeros ciclos. En lugar de esperar a recibir el paquete en su totalidad, la cabecera del paquete puede ser examinada tan pronto como llega. El encaminador puede comenzar a enviar la cabecera y los datos que le siguen tan pronto como se realice una decisión de encaminamiento y el buffer de salida esté libre. De hecho, el mensaje no tiene ni siquiera que ser almacenado en la salida y puede ir directamente a la entrada del siguiente encaminador antes de que el paquete completo se haya recibido en el encaminador actual. A esta técnica de conmutación se le denomina conmutación *virtual cut-through* (VCT) o paso a través virtual. En ausencia de bloqueo, la latencia experimentada por la cabecera en cada nodo es la latencia de encaminamiento y el retraso de propagación a

través del encaminador y a lo largo de los canales físicos. El mensaje puede segmentarse a través de comunicaciones sucesivas. Si la cabecera se bloquea en un canal de salida ocupado, la totalidad del mensaje se almacena en el nodo. Así, para altas cargas de la red, la conmutación VCT se comporta como la conmutación de paquetes.

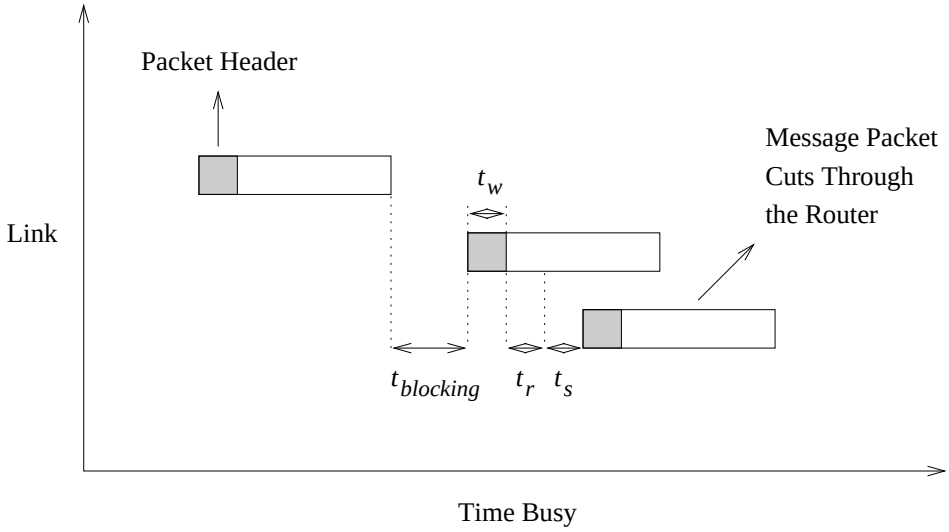


Figura 3.19: Cronograma para un mensaje conmutado por VCT. ($t_{blocking}$= Tiempo de espera en un enlace de salida.)

La figura 3.19 ilustra un cronograma de la transferencia de un mensaje usando conmutación VCT. En esta figura el mensaje se bloquea después de pasar el primer enlace a la espera de que se libere un canal de salida. En este caso observamos cómo la totalidad del paquete tiene que ser transferida al primer encaminador mientras éste permanece bloqueado esperando a la liberación de un puerto de salida. Sin embargo, en la misma figura podemos observar que el mensaje atraviesa el segundo encaminador sin detenerse cruzando al tercer enlace.

La latencia base para un mensaje que no se bloquea en alguno de los encaminadores puede calcularse como sigue:

$$t_{vct} = D(t_r + t_s + t_w) + \max(t_s, t_w) \left\lceil \frac{L}{W} \right\rceil \tag{3.3}$$

En el cálculo de dicha latencia se ha supuesto que el encaminamiento ocurre a nivel de flit con la información del encaminamiento contenida en un flit. Este modelo supone que no existe penalización de tiempo por atravesar un encaminador si el buffer y el canal de salida están libres. Dependiendo de la velocidad de operación de los encaminadores este hecho puede no ser realista. Otro hecho a destacar es que únicamente la cabecera experimenta el retraso del encaminamiento, al igual que el retraso en la conmutación y en los enlaces en cada encaminador. Esto es debido a que la transmisión está segmentada y a la existencia de buffers a la entrada y salida del conmutador. Una vez que el flit cabecera alcanza al destino, el resto del tiempo viene determinado por el máximo entre el retraso del conmutador y el retraso en el enlace entre los encaminadores. Si el conmutador sólo tuviera buffers en la entrada, entonces en un ciclo, un flit atravesaría el conmutador y el canal entre los encaminadores, en este caso el coeficiente del segundo

término sería $t_s + t_w$. Obsérvese que la unidad de control de flujo del mensaje es un paquete. Por lo tanto, incluso en el caso de que el mensaje pueda atravesar el encaminador, debe existir suficiente espacio buffer para permitir el almacenamiento de un paquete completo en el caso de que la cabecera se bloquee.

3.2.5 Conmutación de lombriz (*Wormhole*)

La necesidad de almacenar completamente los paquetes dentro del encaminador (*router*) puede complicar el diseño de encaminadores compactos, rápidos y de pequeño tamaño. En la conmutación segmentada, los paquetes del mensaje también son segmentados a través de la red. Sin embargo, las necesidades de almacenamiento dentro de los encaminadores pueden reducirse sustancialmente en comparación con la conmutación VCT. Un paquete se divide en flits. El flit es la unidad de control de flujo de los mensajes, siendo los buffers de entrada y salida de un encaminador lo suficientemente grandes para almacenar algunos flits. El mensaje se segmenta dentro de la red a nivel de flits y normalmente es demasiado grande para que pueda ser totalmente almacenado dentro de un buffer. Así, en un instante dado, un mensaje bloqueado ocupa buffers en varios encaminadores. En la figura 3.20 se muestra el diagrama temporal de la conmutación segmentada. Los rectángulos en blanco y ilustran la propagación de los flits al largo del canal físico. Los rectángulos sombreados muestran la propagación de los flits cabecera a lo largo de los canales físicos. También se muestran en la figura los retrasos debidos al encaminamiento y a la propagación dentro del encaminador de los flits cabecera. La principal diferencia entre la conmutación segmentada y la conmutación VCT es que la unidad de control del mensaje es el flit y, como consecuencia, el uso de buffers más pequeños. Un mensaje completo no puede ser almacenado en un buffer.

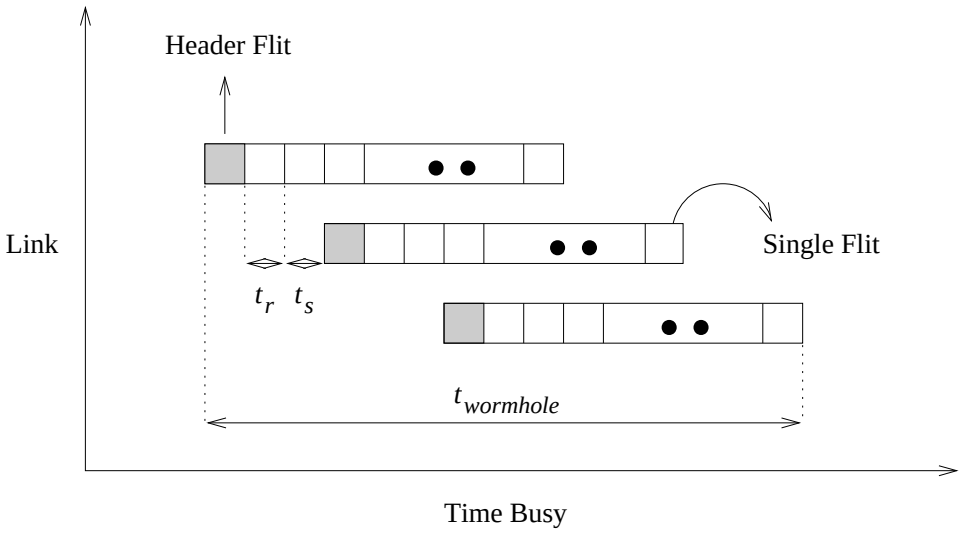


Figura 3.20: Cronograma de un mensaje conmutado mediante wormhole.

En ausencia de bloqueos, los paquetes de mensaje se segmenta a largo de la red. Sin embargo, las características de bloqueo son muy diferentes de las del VCT. Si el canal de salida demandado está ocupado, el mensaje se bloquea in situ. Por ejemplo, la figura 3.21 ilustra una instantánea de un mensaje siendo transmitido a través de

los encaminadores $R1$, $R2$ y $R3$. Los buffers de entrada y salida son de dos flits y las cabeceras tienen esa misma longitud. En el encaminador $R3$, el mensaje A necesita un canal de salida que está siendo utilizado por el mensaje B. Por lo tanto, el mensaje A queda bloqueado. El menor tamaño de los buffers en cada nodo hace que el mensaje ocupe buffers en varios encaminadores, pudiendo dar lugar al bloqueo de otros mensajes. De hecho las dependencias entre los buffers abarcan varios encaminadores. esta propiedad complicará el diseño de algoritmos libres de bloqueos para esta técnica de conmutación, como veremos en la siguiente sección. Sin embargo, ya no es necesario el uso de la memoria del procesador local para almacenar mensaje, reduciendo significativamente la latencia media del mensaje. Las bajas necesidades de almacenamiento y la segmentación de los mensajes permite la construcción de encaminadores que son pequeños, compactos, y rápidos. La figura 3.22 muestra el formato de los paquetes en el Cray T3D.

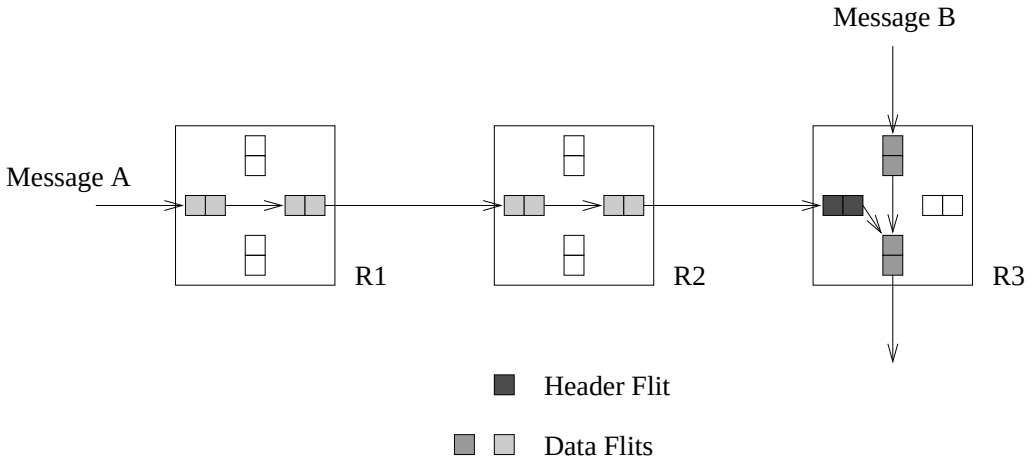


Figura 3.21: Un ejemplo de mensaje bloqueado con la técnica wormhole.

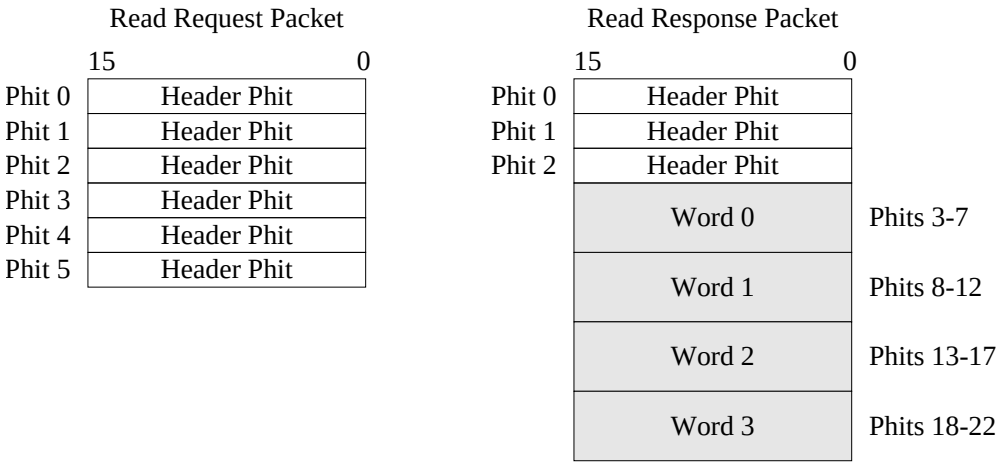


Figura 3.22: Formato de los paquetes conmutados mediante wormhole en el Cray T3D.

La latencia base para un mensaje conmutado mediante wormhole puede calcularse como sigue:

$$t_{wormhole} = D(t_r + t_s + t_w) + \max(t_s, t_w) \left\lceil \frac{L}{W} \right\rceil \tag{3.4}$$

Esta expresión supone que la capacidad de los buffers se mide en flits tanto en las entradas como en las salidas del encaminador. Obsérvese que, en ausencia de contención, VCT y la conmutación segmentada tienen la misma latencia. Una vez que el flit cabecera llega al destino, el tiempo de llegada del resto del mensaje está determinado por el máximo entre el retraso enn el conmutador y el retraso en el enlace.

3.2.6 Conmutación cartero loco

La conmutación VCT mejora el rendimiento de la conmutación de paquetes permitiendo la segmentación de los mensajes a la vez que retiene la posibilidad de almacenar completamente los paquetes de un mensaje. La conmutación segmentada proporciona una mayor reducción de la latencia al permitir menor espacio de almacenamiento que el VCT de tal manera que el encaminamiento puede realizarse utilizando encaminadores implementados en un único chip, proporcionando la menor latencia necesaria para el procesamiento paralelo fuertemente acoplado. Esta tendencia a aumentar la segmentación del mensaje continua con el desarrollo del mecanismo de conmutación del cartero loco en un intento de obtener la menor latencia de encaminamiento posible por nodo.

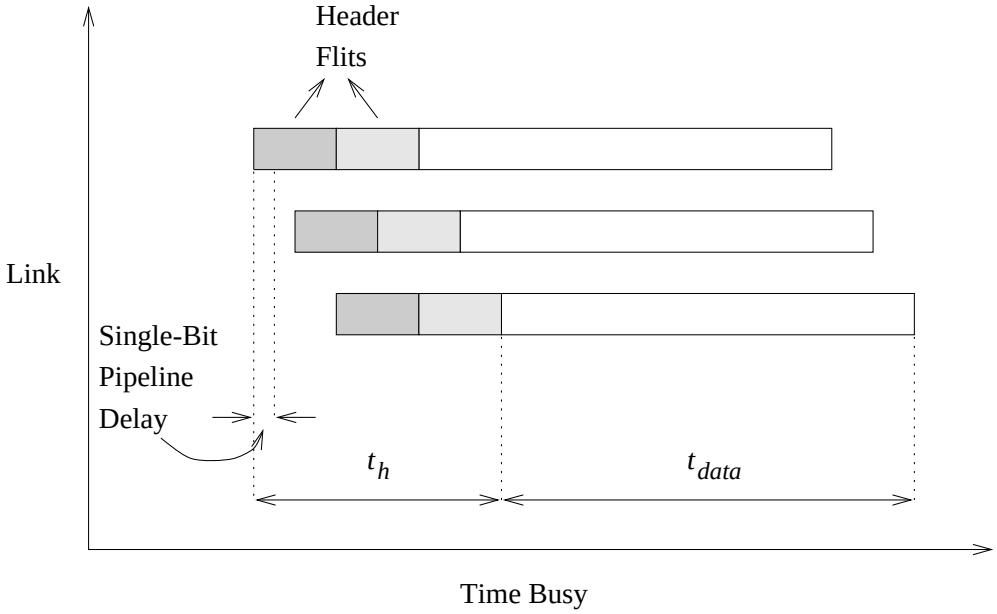


Figura 3.23: Cronograma de la transmisión de un mensaje usando la conmutación del cartero loco.

Esta técnica puede entenderse más fácilmente en el contexto de canales físicos serie. Consideremos una malla 2-D con paquetes cuyas cabeceras tienen dos flits. El encaminamiento se realiza por dimensiones: los mensajes se encaminan en primer lugar a lo largo de la dimensión 0 y después a lo largo de la dimensión 1. El primer flit cabecera contiene la dirección destino de un nodo en la dimensión 0. Cuando el mensaje llega a ese nodo, se envía a través de la dimensión 1. El segundo flit cabecera

contiene el destino del mensaje en esta dimensión. En VCT y *wormhole* los flits no pueden continuar hasta que los flits cabecera han sido recibidos completamente en el encaminador. Si tenemos un flit de 8 bits, la transmisión de los flits cabecera a través de un canal serie de un bit tardará 16 ciclos. Suponiendo un retraso de 1 ciclo para seleccionar el canal de salida en cada encaminador intermedio, la latencia mínima para que la cabecera alcance el encaminador destino a una distancia de tres enlaces es de 51 ciclos. El cartero loco intenta reducir aún más la latencia por nodo mediante una segmentación a nivel de bit. Cuando el flit cabecera empieza a llegar a un encaminador, se supone que el mensaje continuará a lo largo de la misma dimensión. Por lo tanto los bits cabecera se envían hacia el enlace de salida de la misma dimensión tan pronto como se reciben (suponiendo que el canal de salida está libre). Cada bit de la cabecera también se almacena localmente. Una vez que se recibe el último bit del primer flit de la cabecera, el encaminador puede examinar este flit y determinar si el mensaje debe continuar a lo largo de esta dimensión. Si el mensaje debe ser enviado por la segunda dimensión, el resto del mensaje que empieza con el segundo flit de la cabecera se transmite por la salida asociada a la segunda dimensión. Si el mensaje ha llegado a su destino, se envía al procesador local. En esencia, el mensaje primero se envía a un canal de salida y posteriormente se comprueba la dirección, de ahí el nombre de la técnica de conmutación. Esta estrategia puede funcionar muy bien en redes 2-D ya que un mensaje realizará a lo más un giro de una dimensión a otra y podemos codificar la diferencia en cada dimensión un 1 flit cabecera. El caso común de los mensajes que continúan en la misma dimensión se realiza muy rápidamente. La figura 3.23 muestra un cronograma de la transmisión de un mensaje que se transmite sobre tres enlaces usando esta técnica de conmutación.

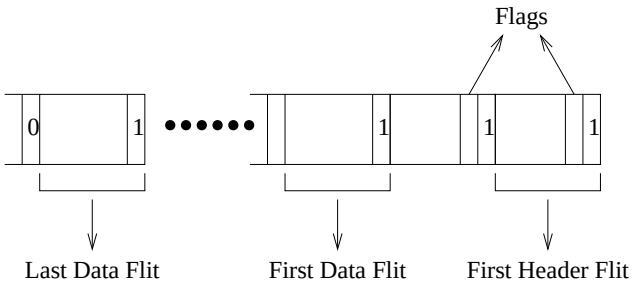


Figura 3.24: Un ejemplo del formato de un mensaje en la técnica de conmutación del cartero loco.

Es necesario considerar algunas restricciones en la organización de la cabecera. la figura 3.24 muestra un ejemplo, donde la diferencia en cada dimensión se codifica en un flit cabecera, y estos flits se ordenan de acuerdo con el orden en que se atraviesa cada dimensión. Por ejemplo, cuando el paquete ha atravesado completamente la primera dimensión el encaminador puede empezar a transmitir en la segunda dimensión con el comienzo del primer bit del segundo flit cabecera. El primer flit se elimina del mensaje, pero continúa atravesando la primera dimensión. A este flit se le denomina *flit de dirección muerto*. En una red multidimensional, cada vez que un mensaje cambia a una nueva dirección, se genera un flit muerto y el mensaje se hace más pequeño. En cualquier punto si se almacena un flit muerto, por ejemplo, al bloquearse por otro paquete, el encaminador local lo detecta y es eliminado.

Consideremos un ejemplo de encaminamiento en una malla 2-D 4×4. En este ejemplo la cabecera de encaminamiento se encuentra comprimida en 2 flits. Cada flit

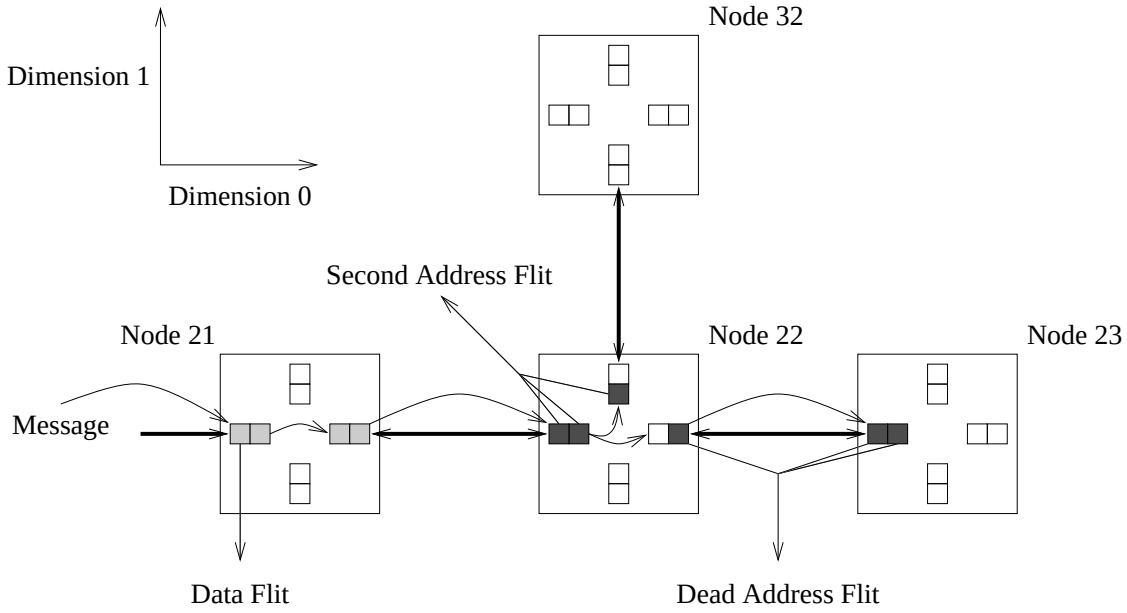


Figura 3.25: Ejemplo de encaminamiento con la conmutación del cartero loco y la generación de flits de dirección muertos.

consta de 3 bits: un bit especial de inicio y dos bits para identificar el destino en cada dimensión. El mensaje se segmenta a través de la red a nivel de bit. El buffer de entrada y salida es de 2 bits. Consideremos el caso en donde el nodo se transmite desde el nodo 20 al nodo 32. la figura 3.25 ilustra el progreso y la localización de los flits cabecera. El mensaje se transmite a lo largo de la dimensión 0 al nodo 22 en donde se transmite al nodo 32 a lo largo de la dimensión 1. En el nodo 22, el primer flit se envía a través del la salida cuando se recibe. Después de recibir el tercer bit, se determina que el mensaje debe continuar a lo largo de la dimensión 1. El primer bit del segundo flit cabecera se envía a través de la salida de la dimensión 1 tal como se muestra la figura. Observar que los flits cabecera se eliminan del mensaje cuando éste cambia de dimensión. Los flits muertos continúan a lo largo de la dimensión 0 hasta que se detectan y son eliminados.

Para un número dado de procesadores el tamaño de los flits de dirección muertos está determinado por el número de procesadores en una dimensión. Por lo tanto, para un número dado de procesadores, las redes de baja dimensión producirán un menor número de flits muertos de mayor tamaño mientras que las redes de mayor dimensión introducirán una mayor cantidad de flits muertos de menor tamaño. Inicialmente podría parecer que los flits de dirección muertos podrían afectar negativamente al rendimiento hasta que son eliminados de la red ya que están consumiendo ancho de banda de los canales físicos. Dado que los paquetes de un mensaje son generalmente más grandes que los flits muertos, la probabilidad de que un paquete se bloquee a causa de un flit de dirección muerto es muy pequeña. Es más probable que el flit muerto se bloquee por un paquete. En este caso el encaminador local tiene la oportunidad de detectar el flit muerto y eliminarlo de la red. A altas cargas, podríamos estar preocupados por los flits de dirección muertos que consumen un ancho de banda escaso. En este caso, es interesante destacar que un incremento del bloqueo en la red proporcionará más oportunidades para que los encaminadores eliminen los flits muertos. A mayor congestión, menor será la probabilidad de que un paquete encuentre un flit de dirección muerto.

Mediante el encaminamiento optimista del flujo de bits que componen un mensaje hacia un canal de salida, la latencia de encaminamiento en un nodo se minimiza consiguiéndose una segmentación a nivel de bit. Consideremos de nuevo una malla 2-D con canales físicos con una anchura de 1 bit y con paquetes con dos flits cabecera de 8-bits, atravesando tres enlaces, la latencia mínima para que la cabecera alcance el destino es de 18 en lugar de 51 ciclos. En general, la estrategia del cartero loco es útil cuando son necesarios varios ciclos para que la cabecera atraviese un canal físico. En este caso la latencia puede reducirse mediante el envío optimista de porciones de la cabecera antes de que pueda determinarse el enlace de salida real. Sin embargo, los modernos encaminadores permiten la transmisión de flits más anchos a lo largo de un canal en un único ciclo. Si la cabecera puede transmitirse en un ciclo, las ventajas que se pueden obtener son pequeñas.

La latencia base de un mensaje encaminado usando la técnica de conmutación del cartero loco puede calcularse como sigue:

$$\begin{aligned} t_{madpostman} &= t_h + t_{data} \\ t_h &= (t_s + t_w)D + \max(t_s, t_w)W \\ t_{data} &= \max(t_s, t_w)L \end{aligned} \tag{3.5}$$

La expresión anterior realiza varias suposiciones. La primera es que se usan canales serie de 1 bit que son los más favorables a la estrategia del cartero loco. El tiempo de encaminamiento t_r se supone que es equivalente al retraso en la conmutación y ocurre de forma concurrente con la transmisión del bit, y por lo tanto no aparece en la expresión. El término t_h se corresponde con el tiempo necesario para enviar la cabecera.

Consideremos el caso general en donde no tenemos canales serie, sino canales de C bits, donde $1 < C < W$. En este caso se necesitan varios ciclos para transferir la cabecera. En este caso la estrategia de conmutación del cartero loco tendría una latencia base de

$$t_{madpostman} = D(t_s + t_w) + \max(t_s, t_w) \left\lceil \frac{W}{C} \right\rceil + \max(t_s, t_w) \left\lceil \frac{L}{C} \right\rceil \tag{3.6}$$

Por razones de comparación, en este caso la expresión de la conmutación segmentada habría sido

$$t_{wormhole} = D \left\{ t_r + (t_s + t_w) \left\lceil \frac{W}{C} \right\rceil \right\} + \max(t_s, t_w) \left\lceil \frac{L}{C} \right\rceil \tag{3.7}$$

Supongamos que los canales internos y externos son de C bits, y un flit cabecera (cuya anchura es de W bits) requiere $\lceil \frac{W}{C} \rceil$ ciclos para cruzar el canal y el encaminador. Este coste se realiza en cada encaminador intermedio. Cuando $C = W$, la expresión anterior se reduce a la expresión de la conmutación segmentada con cabeceras de un único flit.

3.2.7 Canales virtuales

Las técnicas de comunicación anteriores fueron descritas suponiendo que los mensajes o parte de los mensajes se almacenan a la entrada y salida de cada canal físico. Por

lo tanto, una vez que un mensaje ocupa el buffer asociado a un canal, ningún otro mensaje pueda acceder al canal físico, incluso en el caso de que el mensaje este bloqueado. Sin embargo, un canal físico puede soportar varios canales *virtuales* o *lógicos* multiplexados sobre el mismo canal físico. Cada canal virtual unidireccional se obtiene mediante el manejo independiente de un par de buffers de mensajes como se muestra en la figura 3.26. Esta figura muestra dos canales virtuales unidireccionales en cada dirección sobre un canal físico. Consideremos la conmutación segmentada con mensaje en cada canal virtual. Cada mensaje puede comprobar si el canal físico a nivel de flit. El protocolo del canal físico debe ser capaz de distinguir entre los canales virtuales que usan el canal físico. Lógicamente, cada canal virtual funciona como si estuviera utilizando un canal físico distinto operando a mitad de velocidad. Los canales virtuales fueron originariamente introducidos para resolver el problema de bloqueo en las redes con conmutación segmentada. El bloqueo en una red ocurre cuando los mensajes no pueden avanzar debido a que cada mensaje necesita un canal ocupado por otro mensaje. Discutiremos este problema en detalle en la siguiente sección.

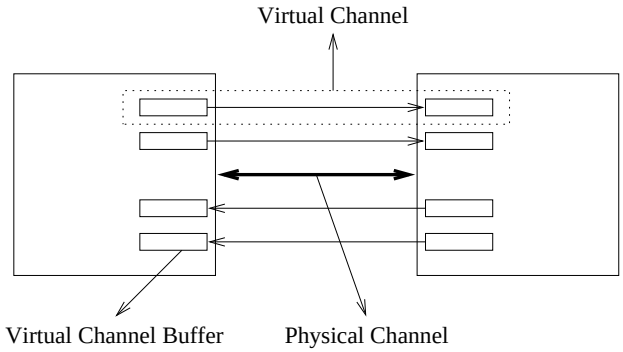


Figura 3.26: Canales virtuales.

Los canales virtuales también se pueden utilizar para mejorar la latencia de los mensajes y el rendimiento de la red. Al permitir que los mensajes compartan un canal físico, los mensajes pueden progresar en lugar de quedarse bloqueados. Por ejemplo, la figura 3.27 muestra a los mensajes cruzando el canal físico entre los encaminadores $R1$ y $R2$. Sin canales virtuales, el mensaje A impide al mensaje B avanzar hasta que la transmisión del mensaje A si hubiera completado. Sin embargo, en la figura existen dos canales virtuales multiplexados sobre cada canal físico. De esta forma, ambos mensajes pueden continuar avanzando. La velocidad a la cual cada mensaje avanza es nominalmente la mitad de la velocidad conseguida cuando el canal no está compartido. De hecho, el uso de canales virtuales desacopla los canales físicos de los buffers de mensajes permitiendo que varios mensajes compartan un canal físico de la misma manera que varios programas pueden compartir un procesador. El tiempo total que un mensaje permanece bloqueado en un encaminador a la espera de un canal libre se reduce, dando lugar a una reducción total en la latencia de los mensajes. Existen dos casos específicos donde la compartición del ancho de banda de un canal físico es particularmente beneficiosa. Consideremos el caso donde el mensaje A está temporalmente bloqueado en el nodo actual. Con un protocolo de control de flujo de los canales físicos apropiado, el mensaje B puede hacer uso de la totalidad del ancho de banda del canal físico entre los encaminadores. Sin canales virtuales, ambos mensajes estarían bloqueados. Consideremos ahora el caso en donde mensaje A es mucho más grande, en comparación, que el mensaje B. El mensaje B puede continuar a mitad de la velocidad del enlace, y a continuación el mensaje A puede continuar la transmisión utilizando todo el ancho de

banda del enlace.

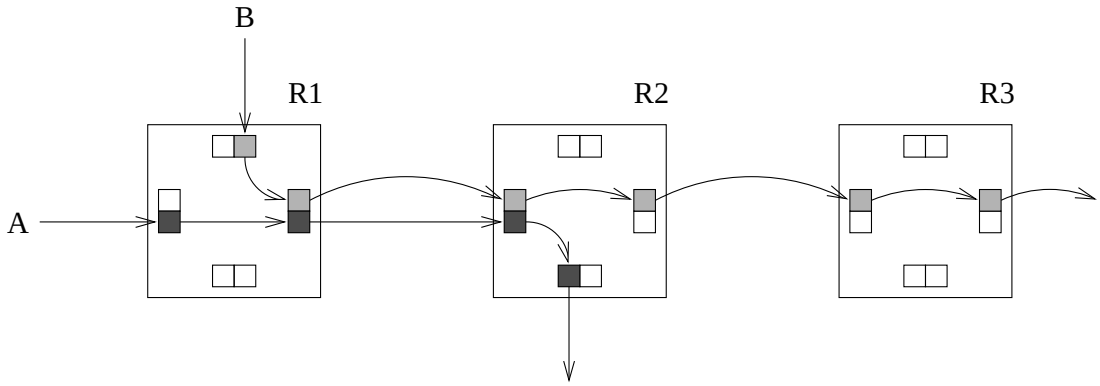


Figura 3.27: Un ejemplo de la reducción del retraso de la cabecera usando dos canales virtuales por canal físico.

Podríamos imaginarnos añadiendo más canales virtuales para conseguir una mayor reducción al bloqueo experimentado por cada mensaje. El resultado sería un incremento del rendimiento de la red medido en flits/s, debido al incremento de la utilización de los canales físicos. Sin embargo, cada canal virtual adicional mejora el rendimiento en una cantidad menor, y el incremento de la multiplexación de canales reduce la velocidad de transferencia de los mensajes individuales, incrementando la latencia del mensaje. Este incremento en la latencia debido a la multiplexación sobrepasará eventualmente a la reducción de la latencia debido al bloqueo dando lugar a un incremento global de la latencia media de los mensajes.

El incremento en el número de canales virtuales tiene un impacto directo en el rendimiento del encaminador debido a su impacto en el ciclo de reloj hardware. El controlador de los enlaces se hace cada vez más complejo dado que debe soportar el arbitraje entre varios canales virtuales. El número de entradas y salidas que deben de conmutarse en cada nodo aumenta, incrementando sustancialmente la complejidad del conmutador. Para una cantidad fija de espacio buffer en nodo, ¿cómo se asigna dicho espacio entre los canales?. Además, el flujo de mensajes a través del encaminador debe coordinarse con la asignación del ancho de banda del canal físico. El incremento de la complejidad de estas funciones puede dar lugar a incremento en las latencias de control de flujo interno y externo. Este incremento afecta a todos los mensajes que pasan a través de los encaminadores.

3.2.8 Mecanismos híbridos de conmutación

Conmutación encauzada de circuitos

Conmutación de exploración

3.2.9 Comparación de los mecanismos de conmutación

En esta sección, comparamos el rendimiento de varias técnicas de conmutación. En particular, analizamos el rendimiento de redes que utilizan conmutación de paquetes, VCT,

y conmutación segmentada (*wormhole switching*). VCT y la conmutación segmentada tienen un comportamiento similar a baja carga.

Para conmutación de paquetes, consideraremos buffers laterales con capacidad para cuatro paquetes. Para VCT, consideraremos buffers con capacidades para uno, dos y cuatro paquetes. Para la conmutación segmentada, se mostrará el efecto de añadir canales virtuales. El número de canales virtuales varía de uno a cuatro. En comparación, la capacidad de los buffers asociados a cada canal virtual se mantiene constante (4 flits) sin importar el número de canales virtuales. Por lo tanto, al aumentar los canales virtuales se aumenta también la capacidad total de almacenamiento asociada con cada canal físico. El efecto de añadir canales virtuales a la vez que se mantiene la capacidad total de almacenamiento constante se estudiará posteriormente.

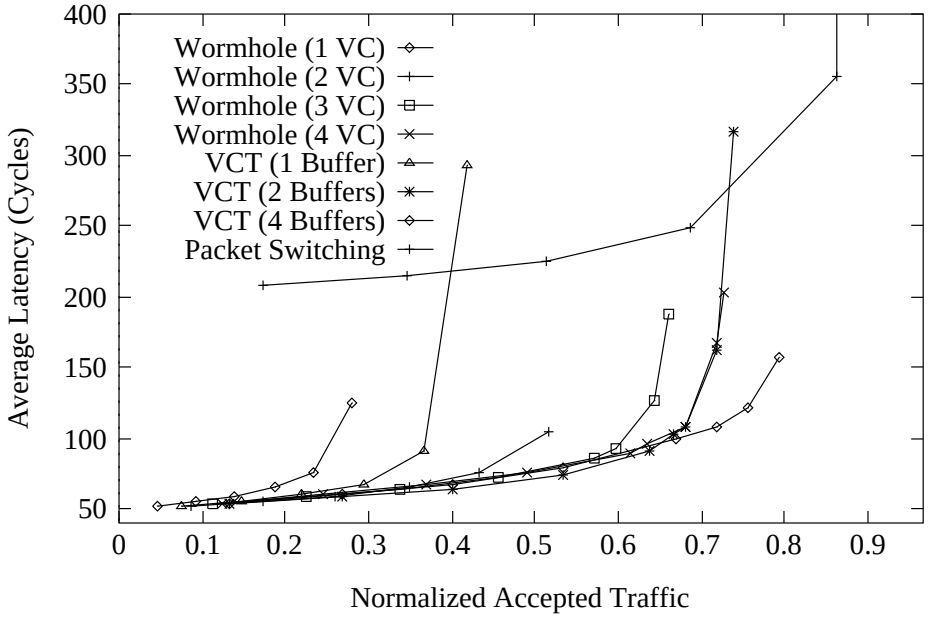


Figura 3.28: Latencia media del paquete vs. tráfico aceptado normalizado en una malla 16×16 para diferentes técnicas de conmutación y capacidades de buffer. (VC = Virtual channel; VCT = Virtual cut-through.)

La figura muestra la latencia media de un paquete en función del tráfico aceptado normalizado para diferentes técnicas de conmutación en un malla de 16×16 usando un encaminamiento por dimensiones, paquetes de 16 flits y una distribución uniforme del destino de los mensajes. Como era de esperar de la expresión de la latencia que obtuvimos para las diferentes técnicas de conmutación, VCT y la conmutación segmentada presentan la misma latencia cuando el tráfico es bajo. Esta latencia es mucho menor que la obtenida mediante la conmutación de paquetes. Sin embargo, al incrementarse el tráfico la conmutación segmentada sin canales virtuales satura rápidamente la red, dando lugar a una baja utilización de los canales.

Esta baja utilización de canales puede mejorarse añadiendo canales virtuales. Al añadir canales virtuales, el rendimiento de la red (*throughput*) también aumenta. Como puede observarse, el añadir nuevos canales virtuales da lugar a una mejora en el rendimiento cada vez menor. De forma similar, al incrementar el tamaño de la cola en la técnica de VCT da lugar a un aumento considerable del rendimiento de la red. Una observación interesante es que la latencia media para VCT y la conmutación segmentada

con canales virtuales es casi idéntica para todo el rango de carga aplicada hasta que las curvas alcanza el punto de saturación. Además, cuando la capacidad de almacenamiento por canal físico es la misma que en el VCT, la conmutación segmentada con canales virtuales consigue un mayor rendimiento de la red. Este es el caso del rendimiento de la conmutación segmentada con cuatro canales virtuales frente a la técnica de VCT con capacidad de un único paquete por canal físico.

Cuando la conmutación VCT se implementa utilizando colas laterales con capacidad para varios paquetes, los canales se liberan después de transmitir cada paquete. Por lo tanto, un paquete bloqueado no impide el uso del canal por parte de otros paquetes. Como una consecuencia de este hecho, el rendimiento de la red es mayor que en el caso de la conmutación segmentada con cuatro canales virtuales. Obsérvese, sin embargo, que la mejora es pequeña, a pesar del hecho de que la capacidad de almacenamiento para la conmutación VCT es dos o cuatro veces la capacidad existente en la conmutación segmentada. En particular, obtenemos el mismo resultado para la conmutación segmentada con cuatro canales virtuales y la conmutación VCT con dos buffers por canal. En el caso de paquetes largos, y manteniendo constante el tamaño de los buffers para la conmutación segmentada, los resultados son más favorables para la conmutación VCT pero las necesidades de almacenamiento se incrementan de forma proporcional.

Finalmente, cuando la red alcanza el punto de saturación, la conmutación VCT tiene que almacenar los paquetes muy frecuentemente, con lo que desaparece la segmentación. Como consecuencia, las técnicas de VCT y conmutación de paquetes con el mismo número de buffers consiguen un rendimiento similar cuando la red alcanza el punto de saturación.

La conclusión más importante es que la conmutación segmentada (*wormhole routing*) es capaz de conseguir latencias y rendimiento comparables a los del VCT, en el caso de existir suficientes canales virtuales y teniendo una capacidad de almacenamiento similar. Si la capacidad de almacenamiento es mayor para la técnica de VCT entonces esta técnica de conmutación consigue un mejor rendimiento pero la diferencia es pequeña si se utilizan suficientes canales virtuales en la conmutación segmentada. Una ventaja adicional de la conmutación segmentada es que es capaz de manejar mensajes de cualquier tamaño sin dividirlos en paquetes lo que no ocurre en el caso de VCT, especialmente cuando los buffers se implementan en hardware.

Aunque el añadir nuevos canales virtuales incrementa el tiempo de encaminamiento, haciendo disminuir la frecuencia del reloj, también se pueden realizar consideraciones similares al añadir espacio de almacenamiento en la conmutación VCT. A partir de ahora nos centraremos en redes que utilizan conmutación segmentada a no ser que se indique lo contrario.

3.3 La capa de encaminamiento (*routing*)

En esta sección estudiaremos los algoritmos de encaminamiento. Estos algoritmos establecen el camino que sigue cada mensaje o paquete. La lista de algoritmos propuestos en la literatura es casi interminable. Nosotros nos centraremos en aquellos que han sido usados o propuestos en los multiprocesadores actuales o futuros.

Muchas de las propiedades de la red de interconexión son una consecuencia directa del algoritmo de encaminamiento usado. Entre estas propiedades podemos citar las siguientes:

- *Conectividad*. Habilidad de encaminar paquetes desde cualquier nodo origen a cualquier nodo de destino.
- *Adaptabilidad*. Habilidad de encaminar los paquetes a través de caminos alternativos en presencia de contención o componentes defectuosos.
- *Libre de bloqueos (deadlock y livelock)*. Habilidad de garantizar que los paquetes no se bloquearán o se quedarán esperando en la red para siempre.
- *Tolerancia fallos*. Habilidad de encaminar paquetes en presencia de componentes defectuosos. Aunque podría parecer que la tolerancia fallos implica a la adaptabilidad, esto no es necesariamente cierto. La tolerancia a fallos puede conseguirse sin adaptabilidad mediante en encaminamiento de un paquete en dos o más fases, almacenándolo en algún nodo intermedio.

3.3.1 Clasificación de los algoritmos de encaminamiento

La figura 3.29 muestra una taxonomía de los algoritmos encaminamiento. Los algoritmos de encaminamiento se pueden clasificar de acuerdo a varios criterios. Estos criterios se indican en la columna izquierda en itálica. Cada fila contiene aproximaciones alternativas que pueden usarse para cada criterio. Las flechas indican las relaciones existentes entre las diferentes aproximaciones. Los algoritmos de encaminamiento pueden clasificarse en primer lugar en función del número de destinos. Los paquetes pueden tener un único destino (unicast routing) o varios destinos (multicast routing).

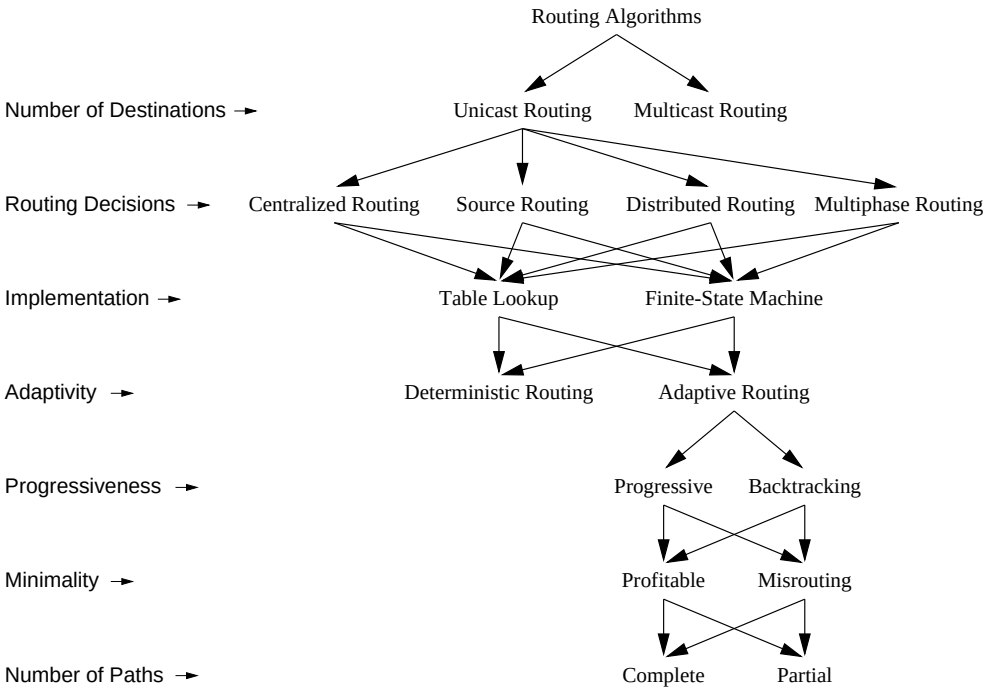


Figura 3.29: Una taxonomía de los algoritmos de encaminamiento

Los algoritmos de encaminamiento también pueden clasificarse de acuerdo con el lugar en donde se toman las decisiones de encaminamiento. Básicamente, el camino puede establecerse o bien por un controlador centralizado (encaminamiento centralizado), en el nodo origen antes de la inyección del paquete (encaminamiento de origen) o

ser determinado de una manera distribuida mientras el paquete atraviesa la red (encaminamiento distribuido). También son posibles esquemas híbridos. A estos esquemas híbridos se les denominan encaminamiento multifase. En el encaminamiento multifase, el nodo origen determina algunos de los nodos destinos. El camino entre estos se establece de una manera distribuida y el paquete puede ser enviado a todos los nodos destinos calculados (*multicast routing*) o únicamente al último de los nodos destino (*unicast routing*). En este caso, los nodos intermedios se usan para evitar la congestión o los fallos.

Los algoritmos de encaminamiento pueden implementarse de varias maneras. Entre ellas, las más interesantes son el uso de una tabla de encaminamiento (*table-lookup*) o la ejecución de un algoritmo de encaminamiento de acuerdo con una máquina de estados finita. En este último caso, los algoritmos puede ser deterministas o adaptativos. Los algoritmos de encaminamiento deterministas siempre suministran el mismo camino entre un nodo origen y un nodo destino. Los algoritmos adaptativos usan información sobre el tráfico de la red y/o el estado de los canales para evitar la congestión o las regiones con fallos de la red.

Los algoritmos de encaminamiento pueden clasificarse de acuerdo con su progresividad como *progresivos* o con *vuelta atrás*. En los algoritmos de encaminamiento progresivos la cabecera siempre sigue hacia delante reservando un nuevo canal en cada operación de encaminamiento. En los algoritmos con vuelta atrás la cabecera puede retroceder hacia atrás, liberando canales reservados previamente.

A un nivel más bajo, los algoritmos de encaminamiento pueden clasificarse dependiendo de su minimalidad como aprovechables (*profitables*) o con desencaminamientos (*misrouting*). Los algoritmos de encaminamiento aprovechables sólo proporcionan canales que acercan al paquete a su destino. También se les denominan *mínimos*. Los algoritmos con desencaminamientos pueden proporcionar además algunos canales que alejen al paquete su destino. A estos últimos también se les denominan *no mínimos*. Al nivel más bajo, los algoritmos de encaminamiento se pueden clasificar dependiendo del número de caminos alternativos como completamente adaptativos (también conocidos como *totalmente adaptativos*) o *parcialmente adaptativos*.

3.3.2 Bloqueos

En las redes directas, los paquetes deben atravesar varios nodos intermedios antes de alcanzar su destino. En las redes basadas en conmutadores, los paquetes atraviesan varios conmutadores antes de alcanzar su destino. Sin embargo, puede ocurrir que algunos paquetes no puedan alcanzar su destino, incluso existiendo un camino libre de fallos entre los nodos origen y destino para cada paquete. Suponiendo que existe un algoritmo de encaminamiento capaz de utilizar esos caminos, existen varias situaciones que pueden impedir la recepción del paquete. En este apartado estudiaremos este problema y las soluciones existentes.

Como se vio en la sección anterior, se necesita espacio buffer para permitir el almacenamiento de fragmentos de un paquete, o incluso la totalidad del mismo, en cada nodo intermedio o conmutador. Sin embargo, dado que existe un coste asociado a la existencia de dichos buffers, la capacidad de los mismos es finita. Al permitir que un paquete cuya cabecera no ha llegado a su destino solicite buffers adicionales al mismo tiempo que mantiene los buffers que almacenan en la actualidad el paquete, puede surgir una situación de bloqueo. Un *bloqueo mortal* (deadlock) ocurre cuando algu-

nos paquetes no pueden avanzar hacia su destino ya que los buffers que solicitan están ocupados. Todos los paquetes involucrados en una configuración de bloqueo mortal están bloqueados para siempre. Obsérvese que un paquete puede bloquearse en la red porque el nodo destino no lo consuma. Esta clase de bloqueo se produce a causa de la aplicación, estando fuera del ámbito que a nosotros nos interesa. En lo que queda de sección supondremos que los paquetes siempre se consumen cuando llegan a su nodo destino en un tiempo finito. La figura 3.30 muestra un ejemplo de esta situación.

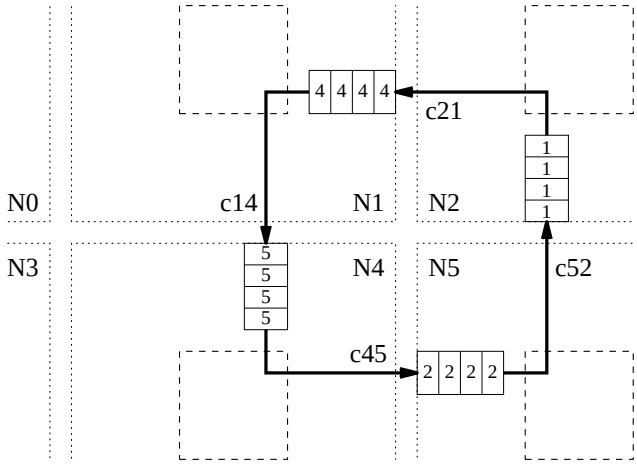


Figura 3.30: Configuración de bloqueo en una malla 2-D.

Una situación diferente surge cuando algunos paquetes no pueden llegar a su destino, incluso aunque no estén bloqueados permanentemente. Un paquete puede estar viajando alrededor del nodo destino sin llegar nunca a alcanzarlo porque los canales que necesitan están ocupados por otros paquetes. A esta situación se le conoce con el nombre de *bloqueo activo* (livelock) y sólo puede ocurrir en el caso de que los paquetes puedan seguir caminos no mínimos.

Finalmente, un paquete puede permanecer completamente parado si el tráfico es intenso y los recursos solicitados son asignados siempre a otros paquetes que lo solicitan. Esta situación, conocida como *muerte por* inanición (starvation), suele deberse a una asignación incorrecta de los recursos en el caso de conflicto entre dos o más paquetes.

Es muy importante eliminar los diferentes tipos de bloqueos (*deadlock*, *livelock* y *starvation*) al implementar un red de interconexión. En caso contrario, algunos paquetes nunca llegarían a su destino. Dado que estas situaciones surgen debido a la limitación en los recursos de almacenamiento, la probabilidad de llegar a una de estas situaciones aumenta al aumentar el tráfico de la red y decrece con la cantidad de espacio de almacenamiento. Por ejemplo, una red que utilice una conmutación segmentada es mucho más sensible a los bloqueos que la misma red utilizando el mecanismo de conmutación de almacenamiento y reenvío (SAF) en el caso de que el algoritmo de encaminamiento no sea libre de bloqueos.

En la figura 3.31 se muestra una clasificación de las situaciones que pueden impedir la llegada de un paquete y las técnicas existentes para resolver estas situaciones. La muerte por inanición (*starvation*) es relativamente sencilla de resolver. Solamente es necesario usar un esquema de asignación de recursos correcto. Por ejemplo, un esquema de asignación de recursos basado en una cola circular. Si permitimos la existencia de distintas prioridades, será necesario reservar parte del ancho de banda para paquetes

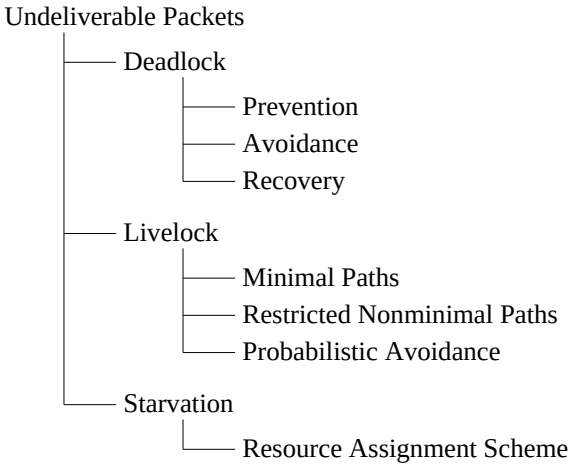


Figura 3.31: Una clasificación de las situaciones que pueden impedir el envío de paquetes.

de baja prioridad para de esta forma evitar la muerte por inanición. Esto se puede conseguir limitando el número de paquetes de mayor prioridad, o reservando algunos canales virtuales o buffers para los paquetes de baja prioridad.

El bloqueo activo (*livelock*) también es fácil de evitar. El modo más simple consiste en usar únicamente caminos mínimos. Esta restricción incrementa el rendimiento en las redes que usan conmutación segmentada ya que los paquetes no ocupan más canales que los estrictamente necesarios para alcanzar su destino. La principal motivación de usar caminos no mínimos es la tolerancia a fallos. Incluso en el caso de usar caminos no mínimos, el bloqueo se puede prevenir limitando el número de encaminamientos que no nos acercan al destino. Otro motivo para usar caminos no mínimos es la evitación del *deadlock* mediante el encaminamiento mediante desviación. En este caso, el encaminamiento es probabilísticamente libre de bloqueos.

El bloqueo mortal (*deadlock*) es de lejos el problema más difícil de resolver. Existen tres estrategias para tratar este problema: *prevención* del bloqueo, *evitación* del bloqueo y *recuperación* del bloqueo[1]. En la prevención del *deadlock*, los recursos (canales o buffers) son asignados a un paquete de tal manera que una petición nunca da lugar a una situación de bloqueo. Esto se puede conseguir reservando todos los recursos necesarios antes de empezar la transmisión del paquete. Este es el caso de todas las variantes de la técnica de conmutación de circuitos en el caso de permitir la vuelta hacia atrás (*backtracking*). En la evitación del *deadlock*, los recursos son asignados a un paquete al tiempo que este avanza a través de la red. Sin embargo, un recurso se le asigna a un paquete únicamente si el estado global resultante es seguro. Este estrategia evita el envío de nuevos paquetes para actualizar el estado global por dos razones: porque consumen ancho de banda y porque pueden contribuir a su vez a producir una situación de bloqueo. Conseguir que el estado global sea seguro de forma distribuida no es una tarea sencilla. La técnica comúnmente utilizada consiste en establecer un orden entre los recursos y garantizar que los recursos son asignados a los paquetes en orden decreciente. En las estrategias de recuperación, los recursos se asignan a paquetes sin ningún chequeo adicional. En este caso son posibles las situaciones de bloqueo, y se hace necesario un mecanismo de detección de las mismas. Si se detecta un *deadlock*, se

[1] En el artículo *Deadlock detection in distributed systems* de M. Singhal se utiliza el término detección en lugar de recuperación.

liberan algunos de los recursos que son reasignados a otros paquetes. Para la liberación de estos recursos, se suele proceder a la eliminación de los paquetes que tenían dichos recursos.

Las estrategias de prevención de bloqueos mortales son muy conservadoras. Sin embargo, la reserva de todos los recursos antes de empezar la transmisión del paquete puede dar lugar a una menor utilización de recursos. Las estrategias de evitación de bloqueos mortales son menos conservadoras, solicitando los recursos cuando son realmente necesarios para enviar el paquete. Finalmente, las estrategias de recuperación son optimistas. Únicamente pueden ser usadas si los bloqueos son raros y el resultado de los mismos puede ser tolerado.

3.3.3 Teoría para la evitación de bloqueos mortales (*deadlocks*)

El objetivo de este apartado es llegar a una condición necesaria y suficiente para conseguir un encaminamiento libre de bloqueos en redes directas utilizando SAF, VCT o conmutación segmentada. El resultado para la conmutación segmentada también puede aplicarse a la conmutación mediante la técnica del cartero loco (*mad postman*) realizando la suposición de que los flits muertos son eliminados de la red tan pronto como se bloquean. Las condiciones necesarias y suficientes para la conmutación segmentada se convierte en una condición suficiente en el caso de método del explorador (*scouting switching*).

Definiciones básicas

La red de interconexión I se modela usando un grafo conexo con múltiples arcos, $I = G(N, C)$. Los vértices del grafo, N, representan el conjunto de nodos de procesamiento. Los arcos del grafo, C, representa el conjunto de canales de comunicación. Se permite la existencia de más de un canal entre cada par de nodos. Los canales bidireccionales se consideran como dos canales unidireccionales. Nos referiremos a un canal y al buffer asociado al mismo de forma indistinta. Los nodos origen y destino de un canal c_i vienen denotados por s_i y d_i, respectivamente.

Un algoritmo de encaminamiento se modela mediante dos funciones: encaminamiento y selección. La *función de encaminamiento* devuelve un conjunto de canales de salida basándose en el nodo actual y el nodo destino. La selección del canal de salida, basándose en el estado de los canales de salida y del nodo actual, a partir de este conjunto se realiza mediante la *función de selección*. Si todos todos los canales de salida están ocupados, el paquete se encaminará de nuevo hasta que consiga reservar un canal. Como veremos, la función de encaminamiento determina si el algoritmo de encaminamiento es o no libre de bloqueos. La función de selección únicamente afecta al rendimiento. Obsérvese que en nuestro modelo el dominio de la función de selección es $N \times N$ ya que sólo tiene en consideración el nodo actual y el nodo destino. Por lo tanto, no se considera el camino seguido por el paquete a la hora de calcular el siguiente canal a utilizar.

Para conseguir unos resultados teóricos tan generales como sean posibles, no supondremos ninguna restricción acerca de la tasa de generación de paquetes, destino de los mismos, y longitud de los mismos. Tampoco supondremos ninguna restricción en

los caminos suministrados por el algoritmo de encaminamiento. Se permiten tanto los caminos mínimos como los no mínimos. Sin embargo, y por razones de rendimiento, un algoritmo de encaminamiento debe proporcional al menos un canal perteneciente a un camino mínimo en cada nodo intermedio. Además, nos centraremos en los bloqueos producidos por la red de interconexión. Por lo tanto, supondremos que los paquetes se consumen en los nodos destinos en un tiempo finito.

Consideraremos varias técnicas de conmutación. Cada una de ellas pueden verse como un caso particular de la teoría general. Sin embargo, serán necesario realizar una serie de suposiciones para algunas de estas técnicas. Para la conmutación segmentada supondremos que una cola no puede contener flits pertenecientes a paquetes diferentes. Después de aceptar el último flit, una cola debe vaciarse antes de aceptar otro flit cabecera. Cuando un canal virtual tiene colas a ambos lados, ambas colas deben vaciarse antes de aceptar otro flit cabecera. Así, si un paquete se bloquea, su flit cabecera siempre ocupará la cabeza de una cola. Además, para cada camino P que pueda establecerse mediante una función de encaminamiento R, todos los subcaminos de P deben de ser también caminos de R. A las funciones de encaminamiento que satisfacen esta última propiedad se les denominan *coherentes*. Para la técnica del cartero loco supondremos las mismas restricciones que en la conmutación segmentada. Además, los flits muertos se eliminan de la red tan pronto como se bloquean.

Una *configuración* es una asignación de un conjunto de paquetes o flits a cada cola. La configuración que se mostró en la figura 3.30 es un ejemplo de *configuración bloqueada*. Esta configuración también estaría bloqueada si existieran paquetes adicionales viajando por la red y que no estén bloqueados. Una configuración bloqueada en donde todos los paquetes están bloqueados se denomina *canónica*. Dada una configuración bloqueada, la correspondiente configuración canónica puede obtenerse deteniendo la inyección de paquetes en todos los nodos, y esperando a que lleguen todos los paquetes que no están bloqueados. Desde un punto de vista práctico, sólo es necesario considerar configuraciones bloqueadas canónicas.

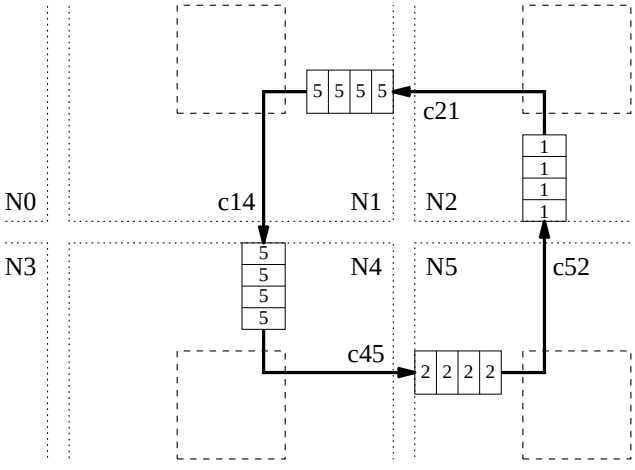


Figura 3.32: Configuración ilegal para R.

Observar que una configuración describe el estado de una red de interconexión en un instante dado. Así, no es necesario considerar configuraciones que describen situaciones imposibles. Una configuración es *legal* si describe una situación posible. En particular, no consideraremos configuraciones en donde se exceda la capacidad del buffer. Así, un paquete no puede ocupar un canal dado al menos que la función de encaminamiento lo

proporcione para el destino de ese paquete. La figura 3.32 muestra una configuración ilegal para una función de encaminamiento R en una malla bidireccional que envía los paquetes por cualquier camino mínimo.

En resumen, una *configuración de bloqueo mortal canónica* es una configuración legal en donde ningún paquete puede avanzar. Si se utilizan las técnicas de conmutación SAF ó VCT:

- Ningún paquete ha llegado a su nodo destino.
- Los paquetes no pueden avanzar debido a que las colas de todos los canales de salida alternativos suministrados por la función de encaminamiento están llenos.

Si se está utilizando el encaminamiento segmentado:

- No hay ningún paquete cuyo flit cabecera haya llegado a su destino.
- Los flits cabecera no pueden avanzar porque las colas de todos los canales de salida alternativos suministrados por la función de encaminamiento no están vacíos (recordar que hemos realizado la suposición de que una cola no puede contener flits pertenecientes a diferentes paquetes).
- Los flits de datos no pueden avanzar porque el siguiente canal reservado por sus paquetes cabecera tiene su cola llena. Observar que un flit de datos puede estar bloqueado en un nodo incluso si hay canales de salida libres para alcanzar su destino ya que los flits de datos deben seguir el camino reservado por su cabecera.

En algunos casos, una configuración no puede alcanzarse mediante el encaminamiento de paquetes a partir de una red vacía. Esta situación surge cuando dos o más paquetes precisan del uso del mismo canal al mismo tiempo para alcanzar dicha configuración. Una configuración que puede alcanzarse mediante el encaminamiento de paquetes a partir de una red vacía se dice que es *alcanzable* o *encaminable*. Hacer notar que de la definición del dominio de la función de encaminamiento como $N \times N$, toda configuración legal es también alcanzable. Efectivamente, dado que la función de encaminamiento no tiene memoria del camino seguido por cada paquete, podemos considerar que, para cualquier configuración legal, un paquete almacenado en la cola de un canal fue generado por el nodo origen de ese canal. En la conmutación segmentada, podemos considerar que el paquete fue generado por el nodo origen del canal que contiene el último flit del paquete. Esto es importante ya que cuando todas las configuraciones legales son alcanzables, no necesitamos considerar la evolución dinámica de la red que dio lugar a dicha configuración. Podemos considerar simplemente las configuraciones legales, sin tener en cuenta de la secuencia de inyección de paquetes necesaria para alcanzar dicha configuración. Cuando todas las configuraciones legales son alcanzables, una función de encaminamiento se dice libre de bloqueos mortales (*deadlock-free*) si, y sólo si, no existe una configuración de bloqueo mortal para dicha función de encaminamiento.

Condición necesaria y suficiente

El modelo teórico para la evitación de bloqueos que vamos a estudiar a continuación se basa en el concepto de *dependencia entre canales*. Cuando un paquete está ocupando un canal y a continuación solicita el uso de otro canal, existe una dependencia entre dichos canales. Ambos canales están en uno de los caminos que puede seguir el paquete. Si se usa conmutación segmentada (*wormhole switching*), dichos canales no tienen por qué ser adyacentes ya que un paquete puede estar ocupando varios ca-

nales simultáneamente. Además, en un nodo dado, un paquete puede solicitar el uso de varios canales para después seleccionar uno de ellos (encaminamiento adaptativo). Todos los canales solicitados son candidatos a la selección. Así, todos los canales solicitados producen dependencias, incluso si no son seleccionados en una operación de encaminamiento determinada. Veámoslo con un ejemplo:

Consideremos un anillo unidireccional con cuatro nodos denotados por n_i, $i = \{0, 1, 2, 3\}$ y un canal unidireccional conectando cada par de nodos adyacentes. Sea c_i, $i = \{0, 1, 2, 3\}$, el canal de salida del nodo n_i. En este caso, es sencillo definir una función de encaminamiento conexa. Se puede enunciar como sigue: Si el nodo actual n_i es igual al nodo destino n_j, almacenar el paquete. En caso contrario, usar c_i, $\forall j \neq i$. La figura 3.33(a) muestra la red. Existe una dependencia cíclica entre los canales c_i. En efecto, un paquete en el nodo n_0 destinado al nodo n_2 puede reservar c_0 y después solicitar c_1. Un paquete en el nodo n_1 destinado al nodo n_3 puede reservar c_1 y después solicitar c_2. Un paquete en el nodo n_2 destinado al nodo n_0 puede reservar c_2 y después solicitar c_3. Finalmente, un paquete en el nodo n_3 destinado al nodo n_1 puede reservar c_3 y después solicitar c_0. Es fácil de ver que una configuración conteniendo los paquetes arriba mencionados no es libre de bloqueos ya que cada paquete tiene reservado un canal y está esperando un canal ocupado por otro paquete.

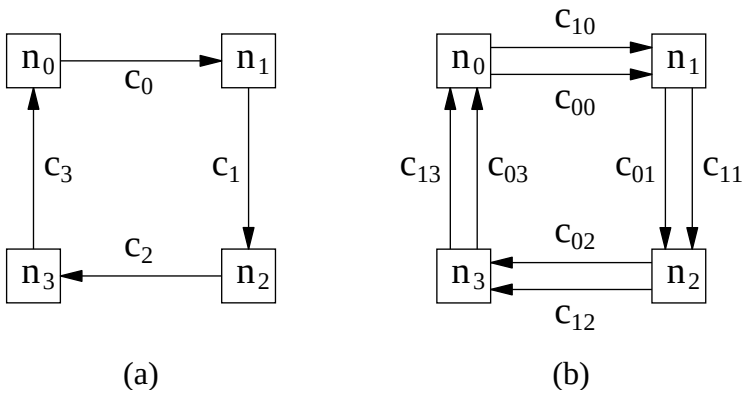


Figura 3.33: Redes del ejemplo anterior.

Consideremos ahora que cada canal físico c_i se divide en dos canales virtuales, c_{0i} y c_{1i}, como se muestra en la figura 3.33(b). La nueva función de encaminamiento puede enunciarse como sigue: Si el nodo actual n_i es igual al nodo destino n_j, almacenar el paquete. En caso contrario, usar c_{0i}, si $j < i$ o c_{1i}, si $j > i$. Como se puede ver, la dependencia cíclica ha sido eliminada ya que después de usar el canal c_{03}, se alcanza el nodo n_0. Así, como todos los destinos tienen un índice mayor que n_0, no es posible pedir el canal c_{00}. Observar que los canales c_{00} y c_{13} nunca son usados. Además, la nueva función de encaminamiento está libre de bloqueos mortales. Veamos que no existe ninguna configuración de bloqueo mortal intentando construir una. Si hubiese un paquete almacenado en la cola del canal c_{12}, estaría destinado a n_3 y los flits podrían avanzar. Así que c_{12} debe estar vacío. Además, si

hubiese un paquete almacenado en la cola de c_{11}, estaría destinado a n_2 ó n_3. Como c_{12} está vacío, los flits pueden avanzar y c_{11} debe estar vacío. Si hubiese un paquete almacenado en la cola de c_{10}, estaría destinado a n_1, n_2 o n_3. Como c_{11} y c_{12} están vacíos, los flits pueden avanzar y c_{10} debe estar vacío. De forma similar, se puede demostrar que el resto de canales pueden vaciarse.

Cuando se considera encaminamiento adaptativo, los paquetes tienen normalmente varias opciones en cada nodo. Incluso si una de estas elecciones es un canal usado por otro paquete, pueden estar disponibles otras elecciones de encaminamiento. Así, no es necesario eliminar todas las dependencias cíclicas, supuesto que cada paquete puede encontrar siempre un camino hacia su destino utilizando canales que no estén involucrados en dependencias cíclicas. Esto se muestra con el siguiente ejemplo:

Consideremos un anillo unidireccional con cuatro nodos denotados por n_i, $i = \{0, 1, 2, 3\}$ y dos canales unidireccionales conectando cada par de nodos adyacentes, excepto los nodos n_3 y n_0 que están enlazados con un único canal. Sea c_{Ai}, $i = \{0, 1, 2, 3\}$ y c_{Hi}, $i = \{0, 1, 2\}$ los canales de salida del nodo n_i. La función de encaminamiento puede formularse como sigue: Si el nodo actual n_i es igual al nodo destino n_j, almacenar el paquete. En caso contrario, usar o bien c_{Ai}, $\forall j \neq i$ o bien c_{Hi}, $\forall j > i$. La figura 3.34 muestra la red.

Existen dependencias cíclicas entre los canales c_{Ai}. Efectivamente, un paquete en el nodo n_0 destinado al nodo n_2 puede reservar c_{A1} y después solicitar c_{A2} y c_{H2}. Un paquete en el nodo n_2 destinado a n_0 puede reservar c_{A2} y después solicitar c_{A3}. Finalmente, un paquete en el nodo n_3 destinado al nodo n_1 puede reservar c_{A3} y después solicitar c_{A0} y c_{H0}.

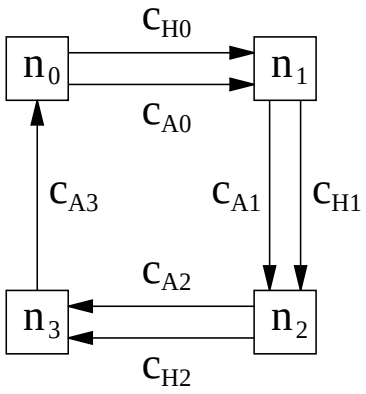


Figura 3.34: Red del ejemplo anterior.

Sin embargo, la función de encaminamiento está libre de bloqueos mortales. Aunque nos centraremos en la conmutación segmentada, el siguiente análisis también es válido para otras técnicas de conmutación. Veamos que no existe ninguna configuración de bloqueo intentando construir una. Si hubiese un paquete almacenado en la cola del canal c_{H2}, estaría destinado a n_3 y los flits podrían avanzar. Así, c_{H2} debe estar vacía. Además, si hubiese un paquete almacenado en la cola de c_{H1}, estaría destinado a n_2 ó n_3. Como c_{H2} está vacío, los flits pueden avanzar y c_{H1} se vaciará. Si hubiesen flits

almacenados en la cola de c_{H0}, estarán destinados a n_1, n_2 ó n_3. Incluso si su flit cabecera estuviese almacenado en c_{A1} ó c_{A2}, como c_{H1} y c_{H2} están vacíos, los flits pueden avanzar y c_{H0} se vaciará.

Así, no es posible establecer una configuración bloqueada utilizando únicamente canales c_{Ai}. Aunque existe una dependencia cíclica entre ellos, c_{A0} no puede contener flits destinados a n_0. Esta configuración no sería legal ya que n_0 no puede enviar paquetes hacia si mismo a través de la red. Para cualquier otro destino, estos flits pueden avanzar ya que c_{H1} y c_{H2} están vacíos. De nuevo, c_{A0} puede vaciarse, rompiendo la dependencia cíclica. Por lo tanto, la función de encaminamiento es libre de bloqueos mortales.

Este ejemplo muestra que los bloqueos pueden evitarse incluso cuando existen dependencias cíclicas entre algunos canales. Obviamente, si existieran dependencias cíclicas entre todos los canales de la red, no habrían caminos de escape para los ciclos. Así, la idea clave consiste en proporcionar un camino libre de dependencias cíclicas para escapar de los ciclos. Este camino puede considerarse como un camino de escape. Observar que al menos un paquete en cada ciclo debe poder seleccionar el camino de escape en el nodo actual, cualquiera que sea su destino. En nuestro caso, para cada configuración legal, el paquete cuyo flit cabecera se encuentra almacenado en el canal c_{A0} puede destinarse a n_1, n_2 o n_3. En el primer caso, este puede ser emitido de forma inmediata. En los otros dos casos, el paquete puede usar el canal c_{H1}.

Las dependencias entre los canales pueden agruparse para simplificar el análisis de los *deadlocks*. Una forma conveniente es el *grafo de dependencias entre canales*[2] que denotaremos por: $D = (G, E)$. Los vértices de D son los canales de la red de interconexión I. Los arcos de D son los pares de canales (c_i, c_j) tales que existe una dependencia de canales de c_i a c_j.

Teorema. *Una función de encaminamiento determinista, R, para una red de interconexión, I, está libre de bloqueos si, y sólo si, no existen ciclos en su grafo de dependencias entre canales, D.*

Prueba: $\Rightarrow$ Supongamos que una red tiene un ciclo en D. Dado que no existen ciclos de longitud 1, este ciclo debe tener una longitud de dos o más. Así, podemos construir una configuración de bloqueo llenando las colas de cada canal en el ciclo con flits destinados a un nodo a una distancia de dos canales, donde el primer canal en el encaminamiento pertenece al ciclo.

$\Leftarrow$ Supongamos que una red no tiene ciclos en D. Dado que D es acíclico, podemos asignar un orden total entre los canales de C de tal manera que si c_j puede ser utilizado inmediatamente después de c_i entonces $c_i > c_j$. Consideremos el menor canal en esta ordenación con una cola llena c_l. Cada canal c_n a los que c_l alimenta es menor que c_l y por lo tanto no tiene una cola llena. Así, ningún flit en la cola de c_l está bloqueado, y no tenemos una situación de bloqueo mortal.

[2]Este grafo fue propuesto por los investigadores Dally y Seitz en su artículo *Deadlock-free message routing in multiprocessor interconnection networks*. IEEE Transactions on Computers, vol. C-36, no. 5, pp. 547-553, May 1987.

3.3.4 Algoritmos deterministas

Los algoritmos de encaminamiento deterministas establecen el camino como una función de la dirección destino, proporcionando siempre el mismo camino entre cada par de nodos. Debemos diferenciar el encaminamiento determinista del encaminamiento inconsciente (*oblivious*). Aunque ambos conceptos han sido considerados a veces como idénticos, en el último la decisión de encaminamiento es independiente del (es decir, inconsciente del) estado de la red. Sin embargo, la elección no es necesariamente determinista. Por ejemplo, una tabla de encaminamiento puede incluir varias opciones como canal de salida dependiendo de la dirección destino. Una opción específica puede seleccionarse aleatoriamente, cíclicamente o de alguna otra manera que sea independiente del estado de la red. Un algoritmo de encaminamiento determinista siempre proporcionaría el mismo canal de salida para el mismo destino. Mientras que un algoritmo determinista es inconsciente, lo contrario no es necesariamente cierto.

El encaminamiento determinista se hizo muy popular cuando Dally propuso el mecanismo de conmutación segmentada. La conmutación segmentada requiere muy poco espacio de buffer. Los encaminadores en este caso son compactos y rápidos. Sin embargo, la segmentación no funciona eficientemente si una de las etapas es mucho más lenta que el resto de etapas. Así, estos encaminadores tienen el algoritmo de encaminamiento implementado en hardware. No cabe, por tanto, sorprenderse que los diseñadores eligieran los algoritmos de encaminamiento más sencillos para de esta forma conseguir un encaminamiento hardware tan rápido y eficiente como fuera posible. La mayoría de los multicomputadores comerciales (Intel Paragon, Cray T3D, nCUBE-2/3) y experimentales (Stanford DASH, MIT J-Machine) usan un encaminamiento determinista.

En este apartado presentaremos los algoritmos de encaminamiento deterministas más populares. Obviamente, los algoritmos más populares son también los más simples.

Encaminamiento por orden de la dimensión

Como ya vimos, algunas topologías pueden descomponerse en varias dimensiones ortogonales. Este es el caso de los hipercubos, mallas, y toros. En estas topologías, es fácil calcular la distancia entre el nodo actual y el nodo destino como suma de las diferencias de posiciones en todas las dimensiones. Los algoritmos de encaminamiento progresivos reducirán una de estas diferencias en cada operación de encaminamiento. El algoritmo de encaminamiento progresivo más simple consiste en reducir una de estas diferencias a cero antes de considerar la siguiente dimensión. A este algoritmo de encaminamiento se le denomina encaminamiento por dimensiones. Este algoritmo envía los paquetes cruzando las dimensiones en un orden estrictamente ascendente (o descendente), reduciendo a cero la diferencia en una dimensión antes de encaminar el paquete por la siguiente.

Para redes n-dimensionales e hipercubos, el encaminamiento por dimensiones da lugar a algoritmos de encaminamiento libres de bloqueos mortales. Estos algoritmos son muy conocidos y han recibido varios nombres, como encaminamiento XY (para mallas 2-D) o *e-cubo* (para hipercubos). Estos algoritmos se describen en las figuras 3.35 y 3.36, respectivamente, donde *FirstOne()* es una función que devuelve la posición del primer bit puesto a uno, e *Internal* es el canal de conexión al nodo local. Aunque estos algoritmos asumen que la cabecera del paquete lleva la dirección absoluta del nodo destino, las primeras sentencias de cada algoritmo calculan la distancia del nodo

actual al nodo destino en cada dimensión. Este valor es el que llevaría la cabecera del paquete si se utilizase un direccionamiento relativo. Es fácil de demostrar que el grafo de dependencias entre canales para el encaminamiento por dimensiones en mallas n-dimensionales e hipercubos es acíclico.

Aunque el encaminamiento por orden de dimensión se suele implementar de manera distribuida usando una máquina de estados finita, también puede implementarse usando en encaminamiento fuente (*street-sign routing*) o una tabla de búsqueda distribuida (*interval routing*).

Bloqueos en toros

El grafo de dependencias entre canales para los toros tiene ciclos, como ya vimos para el caso de anillos unidireccionales en la figura 3.37. Esta topología fue analizada por Dally y Seitz, proponiendo una metodología para el diseño de algoritmos de encaminamiento deterministas a partir del teorema visto en el apartado 3.3.3.

Algorithm: XY Routing for 2-D Meshes
Inputs: Coordinates of current node $(Xcurrent, Ycurrent)$
and destination node $(Xdest, Ydest)$
Output: Selected output $Channel$
Procedure:
$Xoffset := Xdest - Xcurrent$;
$Yoffset := Ydest - Ycurrent$;
if $Xoffset < 0$ **then**
$Channel := X-$;
endif
if $Xoffset > 0$ **then**
$Channel := X+$;
endif
if $Xoffset = 0$ **and** $Yoffset < 0$ **then**
$Channel := Y-$;
endif
if $Xoffset = 0$ **and** $Yoffset > 0$ **then**
$Channel := Y+$;
endif
if $Xoffset = 0$ **and** $Yoffset = 0$ **then**
$Channel := Internal$;
endif

Figura 3.35: El algoritmo de encaminamiento XY para mallas 2-D.

La metodología comienza considerando una función de encaminamiento conexa y su grafo de dependencias entre canales D. Si éste no es acíclico, se restringe el encaminamiento eliminando arcos del grafo D para hacerlo acíclico. Si no es posible conseguir un grafo acíclico sin que la función de encaminamiento deje de ser conexa, se procede a añadir arcos a D asignando a cada canal físico un conjunto de canales virtuales. Utilizando esta metodología se establece una ordenación total entre los canales virtuales, que se etiquetan a continuación según esta ordenación. Cada vez que se rompe un ciclo dividiendo un canal físico en dos canales virtuales, se introduce un nuevo índice para

Algorithm: Dimension-Order Routing for Hypercubes
Inputs: Addresses of current node *Current*
and destination node *Dest*
Output: Selected output *Channel*
Procedure:
$offset := Current \oplus Dest$;
if $offset = 0$ **then**
$Channel := Internal$;
else
$Channel :=$ FirstOne($offset$);
endif

Figura 3.36: El algoritmo de encaminamiento por dimensiones para hipercubos.

establecer el orden entre los canales virtuales. Además, al eliminar un ciclo mediante la utilización de un nuevo canal virtual a cada canal físico, al nuevo conjunto de canales virtuales se le asigna un valor diferente del índice correspondiente. Veamos la aplicación de esta metodología a anillos unidireccionales y a n-cubos k-arios.

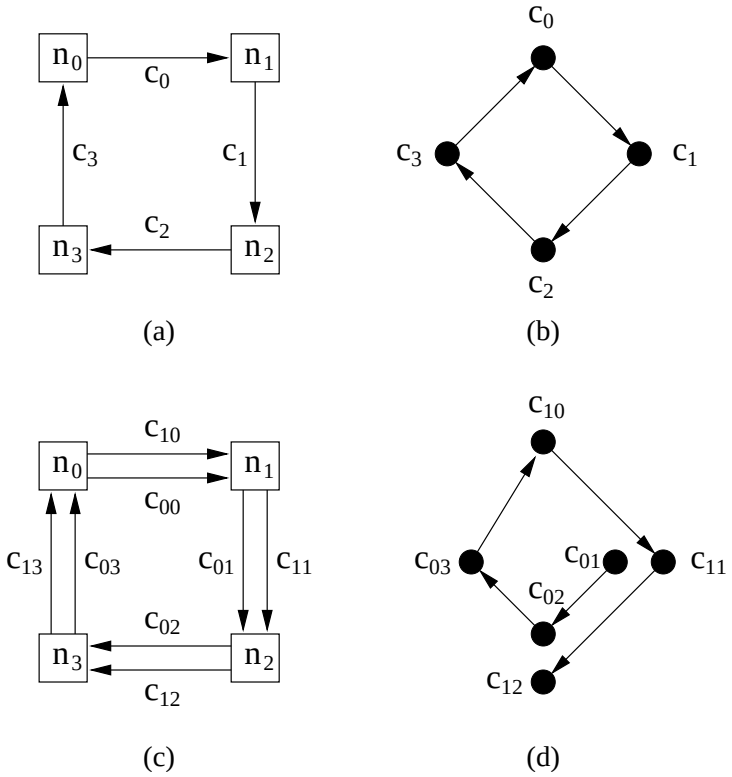


Figura 3.37: Grafo de dependencias entre canales para anillos unidireccionales.

Consideremos un anillo unidireccional con cuatro nodos denotados por n_i, $i = \{0, 1, 2, 3\}$ y un canal unidireccional conectando cada par de nodos adyacentes. Sea c_i, $i = \{0, 1, 2, 3\}$, el canal de salida del nodo n_i. En este caso, es sencillo definir una función de encaminamiento conexa. Se puede enunciar como sigue: Si el nodo actual n_i es igual al nodo destino n_j, almacenar

el paquete. En caso contrario, usar c_i, $\forall j \neq i$. La figura 3.37(a) muestra la red. La figura 3.37(b) muestra que el grafo de dependencias entre canales para esta función de encaminamiento presenta un ciclo. Así, siguiendo la metodología propuesta, cada canal físico c_i se divide en dos canales virtuales, c_{0i} y c_{1i}, como muestra la figura 3.37(c). Los canales virtuales se ordenan de acuerdo con sus índices. La función de encaminamiento se redefine para que utilice los canales virtuales en orden estrictamente decreciente. La nueva función de encaminamiento puede formularse como sigue: Si el nodo actual n_i es igual al nodo destino n_j, almacenar el paquete. En caso contrario, usar c_{0i}, si $j < i$ o c_{1i}, si $j > i$. La figura 3.37(d) muestra el grado de dependencias entre canales para esta función de encaminamiento. Como puede observarse, el ciclo se ha eliminado ya que después de usar el canal c_{03}, se alcanza el nodo n_0. De este modo, como todos los nodos destino tienen un índice mayor que n_0, no es posible pedir el canal c_{00}. Obsérvese que los canales c_{00} y c_{13} no están presentes en el grafo ya que nunca se usan.

Es posible extender la función de encaminamiento de anillos unidireccionales para su uso en n-cubos k-arios unidireccionales. Al igual que en el caso anterior, cada canal físico se divide en dos canales virtuales. Además, se añade un nuevo índice a cada canal virtual. Cada canal virtual vendrá etiquetado por c_{dvi}, donde $d, d = \{0, \ldots, n-1\}$ es la dimensión que atraviesa el canal, $v, v = \{0, 1\}$ indica el canal virtual, e $i, i = \{0, \ldots, k-1\}$ indica la posición dentro del anillo correspondiente. La función de encaminamiento envía los paquetes siguiendo un orden ascendente en las dimensiones. Dentro de cada dimensión, se utiliza la función de encaminamiento para los anillos. Es fácil comprobar que esta función encamina los paquetes en orden estrictamente decreciente de los índices de los canales. La figura 3.38 muestra el algoritmo de encaminamiento por dimensiones para los 2-cubos k-arios (toros bidimensionales).

3.3.5 Algoritmos parcialmente adaptativos

En esta sección nos centraremos en el estudio de algoritmos que incrementan el número de caminos alternativos entre dos nodos dados. Dependiendo de si podemos utilizar cualquier camino mínimo entre un nodo origen y un nodo destino, o únicamente algunos de ellos, los algoritmos pueden clasificarse en totalmente adaptativos o parcialmente adaptativos, respectivamente.

Los algoritmos parcialmente adaptativos representan un compromiso entre la flexibilidad y el coste. Intentan aproximarse a la flexibilidad del encaminamiento totalmente adaptativo a expensas de un moderado incremento en la complejidad con respecto al encaminamiento determinista. La mayoría de los algoritmos parcialmente adaptativos propuestos se basan en la ausencia de dependencias cíclicas entre canales para evitar los bloqueos. Algunas propuestas intentan maximizar la adaptabilidad sin incrementar los recursos necesarios para evitar los bloqueos. Otras propuestas intentan minimizar los recursos necesarios para obtener un cierto nivel de adaptabilidad.

Los algoritmos totalmente adaptativos permiten la utilización de cualquier camino mínimo entre el nodo origen y el nodo destino, maximizando el rendimiento de la red (*throughput*). La mayoría de los algoritmos propuestos se basan en el teorema

Algorithm: Dimension-Order Routing for Unidirectional 2-D Tori
Inputs: Coordinates of current node $(Xcurrent, Ycurrent)$
and destination node $(Xdest, Ydest)$
Output: Selected output $Channel$
Procedure:
$Xoffset := Xdest - Xcurrent$;
$Yoffset := Ydest - Ycurrent$;
if $Xoffset < 0$ **then**
$Channel := c_{00}$;
endif
if $Xoffset > 0$ **then**
$Channel := c_{01}$;
endif
if $Xoffset = 0$ **and** $Yoffset < 0$ **then**
$Channel := c_{10}$;
endif
if $Xoffset = 0$ **and** $Yoffset > 0$ **then**
$Channel := c_{11}$;
endif
if $Xoffset = 0$ **and** $Yoffset = 0$ **then**
$Channel := Internal$;
endif

Figura 3.38: El algoritmo de encaminamiento por dimensiones para toros 2-D unidireccionales.

presentado por Dally y Seitz, requiriendo la ausencia de dependencias cíclicas entre canales, lo que da lugar a un gran número de canales virtuales. Utilizando la técnica de canales de escape propuesta por Duato, es posible conseguir un encaminamiento totalmente adaptativo minimizando el número de recursos.

En esta sección nos centraremos en el estudio de dos algoritmos de encaminamiento, uno parcialmente adaptativo, basado en restringir la adaptabilidad en dos dimensiones, y otro totalmente adaptativo, basado en la idea de caminos de escape que se esbozó en el apartado 3.3.3.

Encaminamiento adaptativo por planos

El objetivo del encaminamiento adaptativo por planos es minimizar los recursos necesarios para conseguir un cierto nivel de adaptatividad. Fue propuesto por Chien y Kim para mallas n-dimensionales e hipercubos. La idea del encaminamiento por planos es proporcionar adaptabilidad en dos dimensiones en un momento dado. Así, un paquete se encamina adaptativamente en una serie de planos 2-D.

La figura 3.39 muestra como funciona el encaminamiento adaptativos por planos. Un encaminamiento totalmente adaptativo permite que un paquete pueda encaminarse en el subcubo m-dimensional definido por el nodo actual y destino, como muestra la figura 3.39(a) para tres dimensiones. El encaminamiento por planos restringe el encaminamiento a utilizar en primer lugar el plano A_0, después moverse al plano A_1, y así sucesivamente, tal como muestran las figuras 3.39(b) y 3.39(c) para tres y cuatro

dimensiones, respectivamente. Dentro de cada plano están permitidos todos los caminos. El número de caminos en un plano dependerá de la distancia en las dimensiones correspondientes.

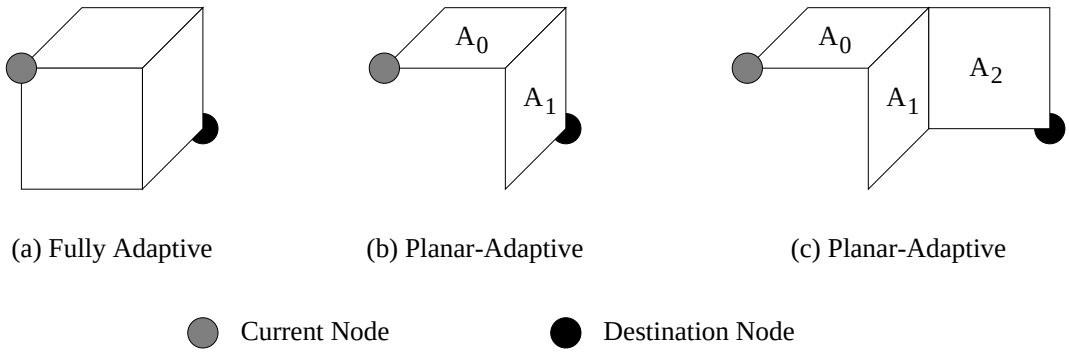


Figura 3.39: Caminos permitidos en el encaminamiento totalmente adaptativo y adaptativo por planos.

Cada plano A_i está formado por dos dimensiones, d_i y d_{i+1}. Existen un total de $(n-1)$ planos adaptativos. El orden de las dimensiones es arbitrario. Sin embargo, es importante observar que los planos A_i y A_{i+1} comparten la dimensión d_{i+1}. Si la diferencia en la dimensión d_i se reduce a cero, entonces el paquete puede pasar al plano A_{i+1}. Si la diferencia en la dimensión d_{i+1} se reduce a cero mientras que el paquete se encuentra en el plano A_i, no existirán caminos alternativos al encaminar el paquete a través del plano A_{i+1}. En este caso, el plano A_{i+1} puede ser omitido. Además, si en el plano A_i, la diferencia en la dimensión d_{i+1} se reduce a cero en primer lugar, el encaminamiento continuará exclusivamente en la dimensión d_i hasta que la diferencia en esta dimensión se reduzca a cero. Por lo tanto, con el fin de ofrecer tantas alternativas de encaminamiento como sea posible, se dará una mayor prioridad a los canales de la dimensión d_i cuando estemos atravesando el plano A_i.

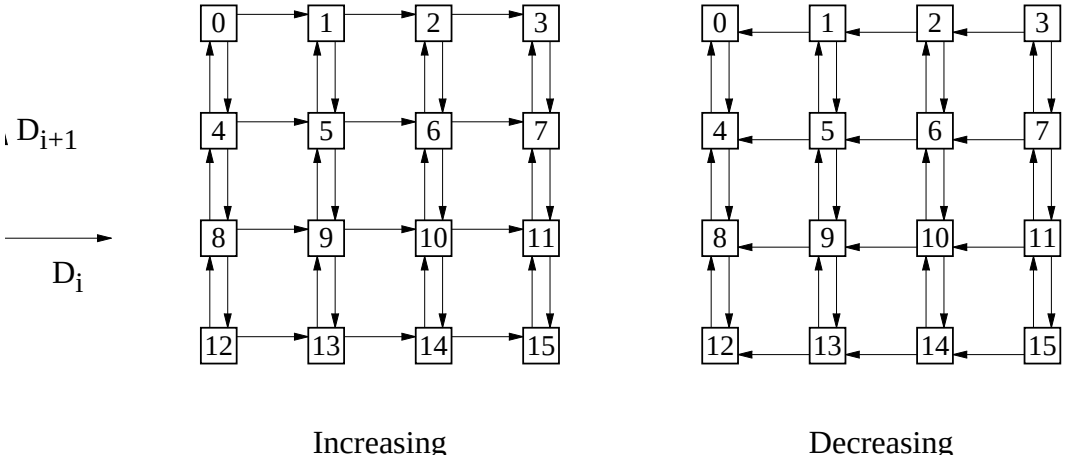


Figura 3.40: Redes crecientes y decrecientes en el plano A_i para el encaminamiento adaptativo por planos.

Tal como hemos definido el encaminamiento por planos, son necesarios tres canales virtuales por canal físico para evitar bloqueos en mallas y seis canales virtuales para

evitar bloqueos en toros. A continuación analizaremos las mallas más detenidamente. Los canales en la primera y última dimensión necesitan únicamente uno y dos canales virtuales, respectivamente. Sea $d_{i,j}$ el conjunto de j canales virtuales que cruzan la dimensión i de la red. Este conjunto puede descomponerse en dos subconjuntos, uno en la dirección positiva y otro en la dirección negativa. Sean $d_{i,j}+$ y $d_{i,j}-$ dichos conjuntos.

Cada plano A_i se define como la combinación de varios conjuntos de canales virtuales:

$$A_i = d_{i,2} + d_{i+1,0} + d_{i+1,1}$$

Con el fin de evitar los bloqueos, el conjunto de canales virtuales en A_i se divide en dos clases: redes *crecientes* y *decrecientes*. Las redes crecientes están formadas por los canales $d_{i,2}+$ y $d_{i+1,0}$. La red decreciente está formada por $d_{i,2}-$ y $d_{i+1,1}$ (Figura 3.40). Los paquetes que cruzan la dimensión d_i en dirección positiva utilizan la red creciente de A_i. Al no existir relación entre la redes crecientes y decrecientes de A_i, y al cruzarse los planos secuencialmente, es fácil de comprobar que no existen dependencias cíclicas entre los canales. Por lo tanto, el algoritmo de encaminamiento adaptativo por planos es libre de bloqueos.

Modelo de giro

3.3.6 Algoritmos completamente adaptativos

Salto negativo

Redes virtuales

Redes deterministas y adaptativas

Algoritmo de Duato

Es posible implementar un algoritmo totalmente adaptativo para n-cubos k-arios utilizando únicamente tres canales virtuales por canal físico. Basándonos en la idea de utilizar un algoritmo libre de bloqueo que actúe como vías de escape y añadiendo canales adicionales a cada dimensión que puedan utilizarse sin restricciones, y que nos proporcionan la adaptabilidad, es posible obtener un algoritmo totalmente adaptativo.

Como algoritmo de base usaremos una extensión del algoritmo adaptativo que vimos para anillos unidireccionales. La extensión para anillos bidireccionales es directa. Simplemente, se usa un algoritmo de encaminamiento similar para ambas direcciones de cada anillo. Dado que los paquetes utilizan únicamente caminos mínimos, no existen dependencias entre los canales de una dirección y los canales de la dirección opuesta. Por lo tanto, el algoritmo de encaminamiento para anillos bidireccionales está libre de bloqueos. La extensión a n-cubos k-arios bidireccionales se consigue utilizando el encaminamiento por dimensiones. Para conseguir un encaminamiento mínimo totalmente adaptativo se añade un nuevo canal virtual a cada canal físico (uno en cada dirección). Estos canales virtuales pueden recorrerse en cualquier sentido, permitiendo cualquier camino mínimo entre dos nodos. Se puede demostrar que el algoritmo resultante es libre de bloqueos.

3.3.7 Comparación de los algoritmos de encaminamiento

En esta sección analizaremos el rendimiento de los algoritmos de encaminamiento deterministas y adaptativos sobre varias topologías y bajo diferentes condiciones de tráfico. Dado el gran número de topologías y algoritmos de encaminamiento existentes, una evaluación exhaustiva sería imposible. En lugar de esto, nos centraremos en unas cuantas topologías y algoritmos de encaminamiento, mostrando la metodología usada para obtener unos resultados que nos permitan una evaluación preliminar. Estos resultados se pueden obtener simulando el comportamiento de la red bajo una carga sintética. Una evaluación detallada requiere usar trazas que sean representativas de las aplicaciones bajo estudio.

La mayoría de los multicomputadores y multiprocesadores actuales usan redes de baja dimensión (2-D ó 3-D) o toros. Por lo tanto, usaremos mallas 2-D y 3-D así como toros para evaluar los distintos algoritmos de encaminamiento. Además, la mayoría de estas máquinas utilizan encaminamiento por dimensiones, aunque el encaminamiento totalmente adaptativo ha empezado a ser introducido tanto en máquinas experimentales como comerciales. Por tanto, analizaremos el comportamiento de los algoritmos de encaminamiento por dimensiones y totalmente adaptativos (estos últimos requieren dos conjuntos de canales virtuales: un conjunto para el encaminamiento determinista y el otro para el encaminamiento totalmente adaptativo).

A continuación realizaremos una breve descripción de los algoritmos de encaminamiento que utilizaremos. El algoritmo determinista para mallas cruza las dimensiones en orden creciente. En principio, no requiere canales virtuales, aunque pueden utilizarse para aumentar el rendimiento. En este último caso, se seleccionará el primer canal virtual libre. El algoritmo totalmente adaptativo para mallas consiste en dos canales virtuales, uno que se utiliza como vía de escape y que permite el encaminamiento siguiendo el algoritmo X-Y y el otro encargado de hacer posibles todos los caminos mínimos entre el nodo actual y el nodo destino. Cuando existen varios canales de salida libres, se da preferencia al canal totalmente adaptativo en la menor dimensión útil, seguido por los canales adaptativos según el orden creciente de dimensiones útiles. Si se usan más de dos canales virtuales, el resto de canales virtuales permiten un encaminamiento totalmente adaptativo. En este caso, los canales virtuales se seleccionan de tal manera que se minimice la multiplexación de canales. El algoritmo determinista para toros requiere dos canales virtuales por canal físico, como vimos en la sección anterior. Cuando se utilizan más de dos canales virtuales, cada par de canales adicionales tienen la misma funcionalidad de encaminamiento que el primer par. También evaluaremos un algoritmo parcialmente adaptativo para toros, basado en la extensión del algoritmo parcialmente adaptativo que vimos para anillos unidireccionales. El algoritmo extendido usa canales bidireccionales siguiendo caminos mínimos. Además, las dimensiones se cruzan en orden ascendente. El algoritmo totalmente adaptativo para toros requiere un tercer canal virtual, el resto de canales se usan como en el caso del algoritmo parcialmente adaptativo. De nuevo, en el caso de existir varios canales de salida libres, se dará preferencia a los canales totalmente adaptativos en la menor dimensión útil, seguido de los canales adaptativos según el orden de dimensiones útiles.

A no ser que se diga lo contrario, los parámetros de simulación son los siguientes: 1 ciclo de reloj para calcular el algoritmo de encaminamiento, para transferir un flit de un buffer de entrada a un buffer de salida, o para transferir un flit a través de un canal físico. Los buffers de entrada y salida tienen una capacidad variable de tal manera que la capacidad de almacenamiento por canal físico se mantiene constante. Cada nodo

tiene cuatro canales de inyección y recepción de paquetes. Además, la longitud de los mensajes se mantiene constante a 16 flits (más 1 flit de cabecera). También se supondrá que el destino de los mensajes sigue una distribución uniforme.

Encaminamiento determinista vs. adaptativo

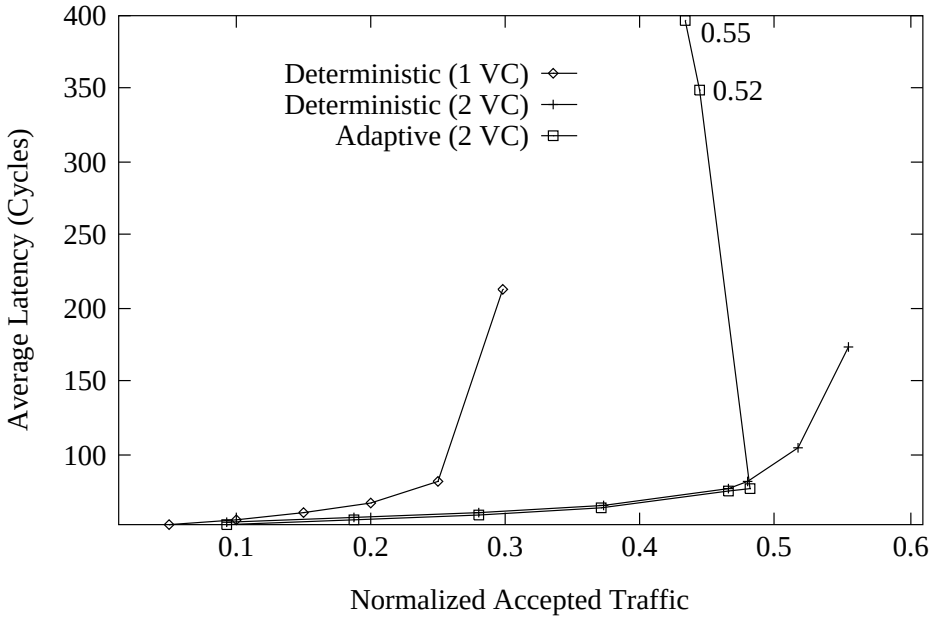


Figura 3.41: Latencia media del mensaje vs. tráfico normalizado aceptado para mallas 16×16 para una distribución uniforme del destino de los mensajes.

La figura 3.41 muestra la latencia media de un mensaje en función del tráfico normalizado aceptado en una malla 2-D. La gráfica muestra el rendimiento del encaminamiento determinista con uno y dos canales virtuales, y totalmente adaptativo (con dos canales virtuales). Como se puede apreciar, el uso de dos canales virtuales casi dobla el rendimiento del algoritmo determinista. La principal razón es que cuando se bloquean los mensajes, el ancho de banda del canal no se malgasta ya que otros mensajes pueden utilizarlo. Por lo tanto, añadiendo unos pocos canales virtuales se consigue reducir la contención e incrementar la utilización del canal. El algoritmo adaptativo consigue el 88% del rendimiento conseguido por el algoritmo determinista con el mismo número de canales. Sin embargo, la latencia es casi idéntica, siendo ligeramente inferior para el algoritmo adaptativo. Así, la flexibilidad adicional del encaminamiento totalmente adaptativo no consigue mejorar el rendimiento cuando el tráfico presenta una distribución uniforme. La razón es que la red está cargada de forma uniforme. Además, las mallas no son regulares, y los algoritmos adaptativos tienden a concentrar tráfico en la parte central de la bisección de la red, reduciendo la utilización de los canales existentes en el borde de la malla.

Obsérvese la existencia de una pequeña degradación del rendimiento cuando el algoritmo adaptativo alcanza el punto de saturación. Si la tasa de inyección se mantiene constante en este punto, la latencia se incrementa considerablemente mientras que el tráfico aceptado decrece. Este comportamiento es típico de los algoritmos de encaminamiento que permiten dependencias cíclicas entre canales.

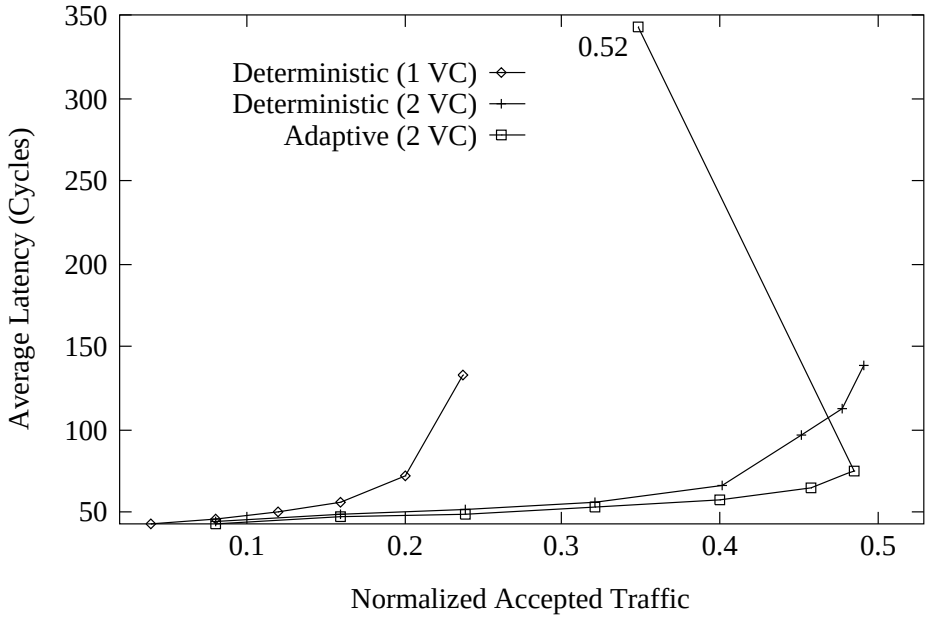


Figura 3.42: Latencia media del mensaje vs. tráfico normalizado aceptado para mallas $8 \times 8 \times 8$ para una distribución uniforme del destino de los mensajes.

La figura 3.42 muestra la latencia media de los mensajes en función del tráfico normalizado aceptado en una malla 3-D. Esta gráfica es bastante similar a la de las mallas 2-D. Sin embargo, existen algunas diferencias significativas. Las ventajas de usar dos canales virtuales en el algoritmo determinista son más evidentes en el caso de mallas 3-D. En este caso, el rendimiento es el doble. Además, el algoritmo totalmente adaptativo alcanza el mismo rendimiento que el algoritmo determinista con el mismo número de canales virtuales. La reducción de la latencia conseguida por el algoritmo totalmente adaptativo es más evidente en las mallas 3-D. La razón es que los mensajes tienen un canal adicional para elegir en la mayoría de los nodos intermedios. De nuevo, existe una degradación del rendimiento cuando el algoritmo adaptativo alcanza el punto de saturación. Esta degradación es más pronunciada que en las mallas 2-D.

La figura 3.43 muestra la latencia media de los mensajes en función del tráfico aceptado en un toro 2-D. La gráfica muestra el rendimiento del encaminamiento determinista con dos canales virtuales, parcialmente adaptativo con dos canales virtuales, y totalmente adaptativo con tres canales virtuales. Tanto el algoritmo parcial como totalmente adaptativo incrementan el rendimiento considerablemente en comparación con el algoritmo determinista. El algoritmo parcialmente adaptativo incrementa el rendimiento en un 56%. La razón es que la utilización de los canales no está balanceada en el algoritmo de encaminamiento determinista. Sin embargo, el algoritmo parcialmente adaptativo permite a la mayoría de los mensajes elegir entre dos canales virtuales en lugar de uno, reduciendo la contención e incrementando la utilización de los canales. Obsérvese que esta flexibilidad adicional se consigue sin incrementar el número de canales virtuales. El algoritmo totalmente adaptativo incrementa el rendimiento de la red en un factor de 2.5 en comparación con el algoritmo determinista. Esta mejora considerable se debe principalmente a la posibilidad de cruzar las dimensiones en cualquier orden. Al contrario que en las mallas, los toros son topologías regulares. Así, los algoritmos adaptativos son capaces de mejorar la utilización de los canales distribuyendo el tráfico de forma uniforme a través de la red. Los algoritmos parcial y totalmente adaptativos

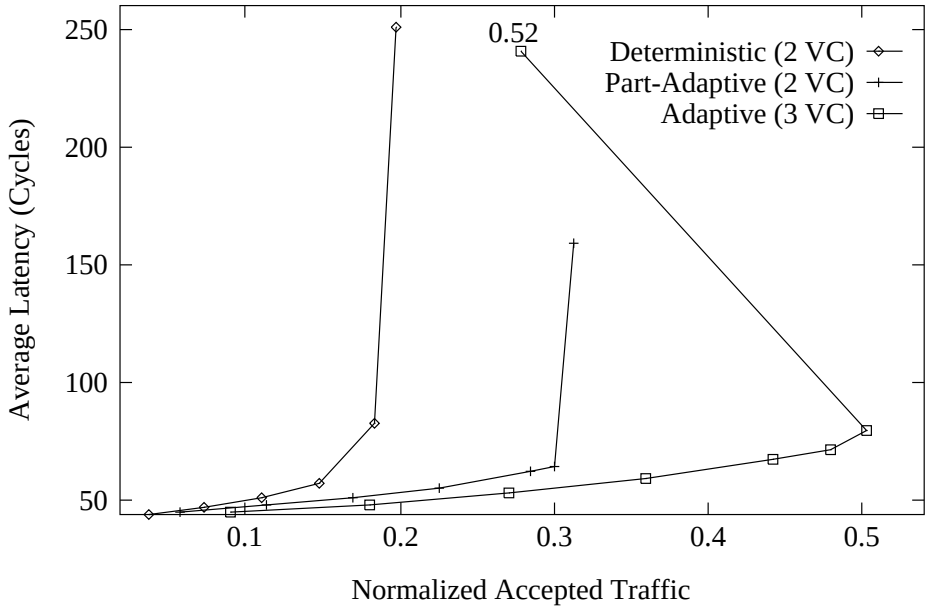


Figura 3.43: Latencia media del mensaje vs. tráfico normalizado aceptado para toros 16×16 para una distribución uniforme del destino de los mensajes.

también consiguen una reducción de la latencia de los mensajes con respecto al algoritmo determinista para todo el rango de carga de la red. De manera similar, el algoritmo totalmente adaptativo reduce la latencia con respecto al parcialmente adaptativo. Sin embargo, la degradación en el rendimiento a partir del punto de saturación reduce el tráfico aceptado por la red al 55% de su valor máximo.

La figura 3.44 muestra la latencia media de los mensajes en función del tráfico aceptado en un toro 3-D. Además de los algoritmos analizados en la figura 3.43, esta gráfica muestra también el rendimiento del algoritmo parcialmente adaptativo con tres canales virtuales. Al igual que en el caso de las mallas, los algoritmos de encaminamiento adaptativos se comportan comparativamente mejor en un toro 3-D que un toro 2-D. En este caso, los algoritmos parcial y totalmente adaptativos incrementan el rendimiento de la red por un factor de 1.7 y 2.6, respectivamente, sobre el rendimiento obtenido con el algoritmo determinista. La reducción en la latencia también es más acusada que en el caso de toros 2-D. La gráfica también muestra que añadiendo un canal virtual al algoritmo parcialmente adaptativo el rendimiento de la red no mejora significativamente. Aunque aumenta el rendimiento en un 18%, también se incrementa la latencia. La razón es que el algoritmo parcialmente adaptativo con dos canales virtuales ya permite el uso de dos canales virtuales en la mayoría de los mensajes, permitiendo la compartición del ancho de banda. Así, añadir otro canal virtual tiene poco impacto en el rendimiento. Este efecto es similar si se consideran otras distribuciones del tráfico en la red. Este resultado también confirma que la mejora conseguida por el algoritmo totalmente adaptativo se debe principalmente a la posibilidad de cruzar las dimensiones en cualquier orden.

La figura 3.45 muestra la desviación estándar de la latencia en función del tráfico aceptado por la red en un toro 2-D. Como se puede observar, un mayor grado de adaptabilidad supone además una reducción de la desviación con respecto al valor medio. La razón es que el encaminamiento adaptativo reduce considerablemente la contención en los nodos intermedios, haciendo que la latencia sea más predecible.

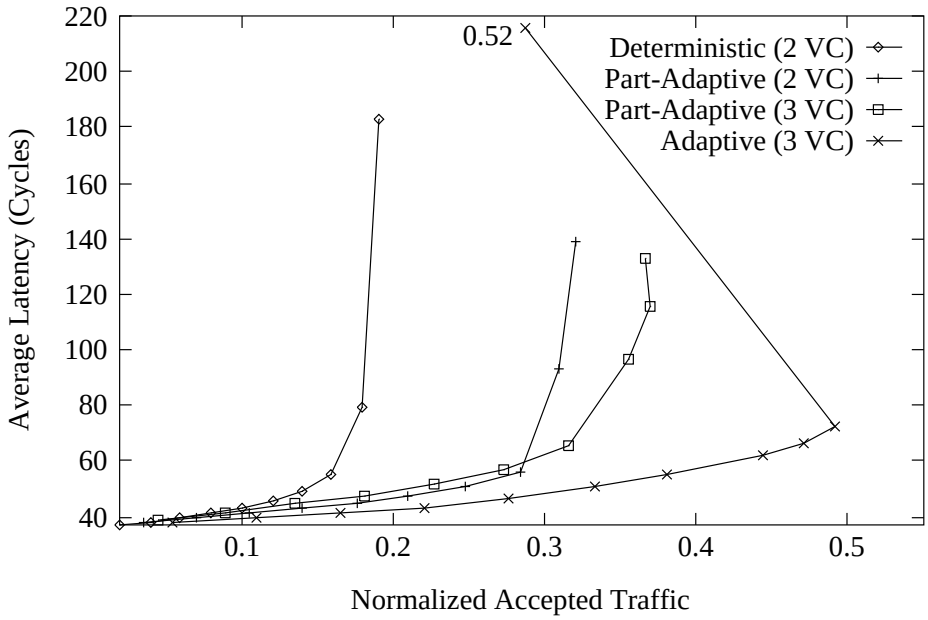


Figura 3.44: Latencia media del mensaje vs. tráfico normalizado aceptado para toros $8 \times 8 \times 8$ para una distribución uniforme del destino de los mensajes.

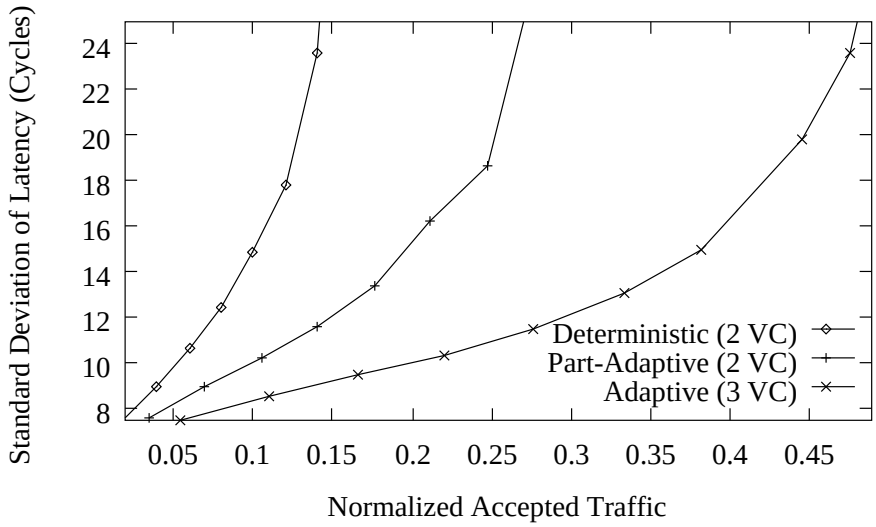


Figura 3.45: Desviación estándar de la latencia vs. tráfico normalizado aceptado en un toro $8 \times 8 \times 8$ para una distribución uniforme del destino de los mensajes