# Minimal Adaptive Routing with Limited Injection on Toroidal $k$-ary $n$-cubes

Fabrizio Petrini and Marco Vanneschi

Dipartimento di Informatica, Università di Pisa,
Corso Italia 40, 56125 Pisa, Italy,
tel +39 50 887248, fax +39 50 887226
e-mail: {petrini,vannesch}@di.unipi.it

August 4, 1996

**Abstract**

Virtual channels can be used to implement deadlock free adaptive routing algorithms and increase network throughput. Unfortunately, they introduce asymmetries in the use of buffers of symmetric networks as the toroidal $k$-ary $n$-cubes.

In this paper we present a minimal adaptive routing algorithm that tries to balance the use of the virtual channels by limiting the injection of new packets into the network. The experimental results, conducted on a 256 nodes torus, show that it is possible to increase the saturation point and to keep the network throughput stable at high traffic rates. The comparison with the Chaos router, a non minimal cut-through adaptive routing, shows that our algorithm obtains similar performance results using only a small fraction of buffers and a simpler router model.

**Keywords**: Interconnection networks, $k$-ary $n$-cubes, tori, wormhole routing, flow control.

## 1 Introduction

The processing nodes of a massively parallel computer exchange data and synchronize with one another by passing messages over an interconnection network. The interconnection network is often the critical component of a large parallel computer because performance is very sensitive to network latency and throughput and because the network accounts for a large fraction of the cost of the machine.

## 1.1 Interconnection networks

An interconnection network is characterized by its **topology**, **routing** and **flow control**. The topology of a network is the arrangement of nodes and channels into a graph. Routing specifies how a packet chooses a path in this graph. Flow control deals with the allocation of channel and buffer resources to a packet as it traverses this path.

Interconnection networks can be classified according to different characteristics. Their topologies fall into two classes: **static** (or direct) and **dynamic** (or indirect).

In a static network, point to point links interconnect the network nodes in some fixed topology; a regular topology as a toroidal mesh or a hypercube is common.

A dynamic network allows the interconnection pattern among the network nodes to be varied dynamically: this is accomplished by some form of switching. Examples of dynamic networks include crossbars, multistage interconnection networks, and many bus based networks.

Both meshes and hypercubes belong to the general class of the **$k$-ary $n$-cube** networks. A $k$-ary $n$-cube is characterized by its dimension $n$ and radix $k$, and has a total of $k^n$ nodes. The $k^n$ nodes are organized in an $n$-dimensional mesh, with $k$ nodes in each dimension. The binary hypercube is a special case of $k$-ary $n$-cube with $k = 2$. Also, the two-dimensional mesh is another special case with $n = 2$. A $k$-ary $n$-cube with wraparound links is often called **torus connected**. Figure 1 shows an example of $k$-ary $n$-cube.

Most static interconnection networks provide multiple physical paths for routing a message between two given nodes. This introduces the problem of choosing a route between many alternatives.

The simplest approach is the **deterministic** one, where the route is fully determined by the source and destination addresses. This has the advantage of being very simple, but is unable to adapt the network to conditions such as congestion or failures. **Adaptive routing** is very important to provide a network performance which is less sensitive to the communication pattern. In this case, the paths can be chosen according to the degree of congestion of the node where the routing decision is taken. A **minimal** adaptive routing algorithm limits the path selection to the shortest paths between any given pair $\langle source, destination \rangle$. With a **non minimal** routing algorithm, the selected path may not always be a shortest path.

Modern parallel routers significantly reduce average latency by using wormhole flow control [11]. Wormhole is a flow control strategy that divides each packet in elementary units called **flits** and advances each flit as

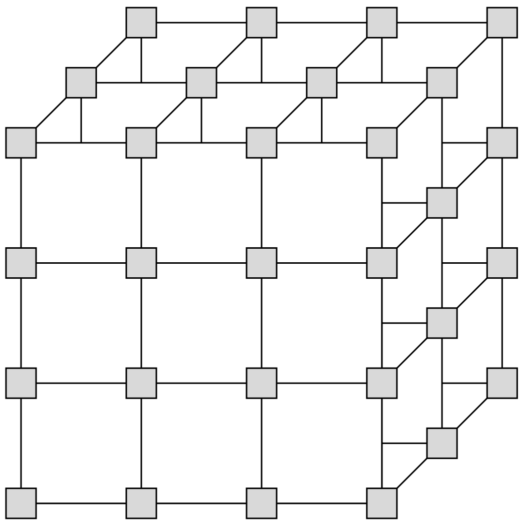

Figure 1: A 4-ary 3-cube.

soon as it arrives at a node, in a pipeline fashion. Wormhole is attractive because it reduces the latency of message delivery compared to store and forward and requires only a few flit buffers per node. Network throughput of wormhole routed networks can be increased by organizing the flit buffers associated with each physical channel into several virtual channels [8]. These virtual channels are allocated independently to different packets and compete with each other for the physical bandwidth. This decoupling allows active messages to pass blocked messages using network bandwidth that would otherwise be wasted. Adding virtual channels to an interconnection network is analogous to adding lanes to a street network. A network without virtual channels is composed of one-lane streets. In such a network, a single blocked packet blocks all following packets. Adding virtual channels to the network adds lanes to the streets allowing blocked packets to be passed.

## 1.2 Related work

Many wormhole adaptive algorithms have been developed for interconnection networks in the $k$-ary $n$-cube family. These algorithms display interesting tradeoffs between their degree of adaptivity and the number of virtual channels needed. If the dimensions are routed strictly in order, as in row column or e-cube routing, no deadlocks can result in a mesh and binary $n$-cube, even with one virtual channel for physical channel. Introducing adaptivity usually requires an increase in the number of virtual channels. For example, Linder and Harden [19] describe fully adaptive minimal algorithms for $k$-ary

$n$-cubes with unidirectional and bidirectional links. For the unidirectional case, $(n+1)$ virtual channels are needed for physical channel. For $k$-ary $n$-cubes with bidirectional links, $2^{n-1}$ virtual channels are needed per physical link in each direction, if the network has no wrap around links. With toroidal cubes the number increases to $(n+1)2^{n-1}$.

Chien and Kim [7] presented an approach to trade off adaptivity against the number of virtual channels. Their algorithm, called **planar adaptive routing**, is minimal and partially adaptive. This approach involves examining the routing dimensions in pairs, and constraining the routing choices at any time to one or two dimensions. This, in general, is less flexible than the fully adaptive algorithm of Linder and Harden, but requires only a constant number of virtual channels, regardless of the network dimension. For example, in a $k$-ary $n$-cube without wrap around connections, only three virtual channels for each physical link are required.

The **turn model** [17] prevents some of the transitions between dimensions, and generalizes to multidimensional meshes and binary $n$-cubes. This scheme requires only a virtual channel per physical channel, is non minimal, and partially adaptive. The approach for two-dimensional meshes works by disallowing two of the eight possible turns a packet may take.

The turn model can be used in conjunction with virtual channels to increase adaptivity and to generalize to torus connected $k$-ary $n$-cubes. Thus, instead of prohibiting some turns, the packet can be switched to a different virtual channel upon taking such a turn. Dally and Aoki [9] used this idea to design partially adaptive non minimal routing algorithms for the class of $k$-ary $n$-cubes. In their algorithms, each packet carries with it a dimension reversal number which keeps track of the number of times the packet has been routed from a channel in one dimension $p$ to a channel in a lower dimension $q < p$.

To reduce the number of virtual channels, non monotonic allocation strategies have been proposed [14] [3]. They are summarized in a necessary and sufficient condition for deadlock free adaptive routing [15].

The rest of this paper is organized as follows. Section 2 motivates our work. Section 3 describes the routing algorithm and the limited injection mechanisms. The relevant details of the network model are discussed in Section 4. Using this model, we tune the injection mechanisms and evaluate the communication performance of a toroidal 16-ary 2-cube in Section 5. Finally Section 6 concludes the paper, summarizing the results.

# 2 Motivation

To achieve maximum performance in a particular network, each of the resources should be equally loaded. If one component reaches saturation before the rest, the network will tend to slow to the load which can be handled by the saturated component. Tori are **symmetric** or **isotropic** networks, that is they look identical when viewed from every node and every edge. Thus they should have no particular problem of congestion under uniform random traffic. However, the introduction of virtual channels perturbs the routing and flow control mechanisms in a manner which re-introduces non-uniformities in the use of the buffers [4].

In [1] it is shown that, as the network gets closer to the saturation point, there is a perceptible difference in the efficiencies of nodes at different locations because of asymmetric loads on the virtual channels imposed by the deterministic routing algorithm. A common solution to implement adaptive routing is to divide the virtual channels into two sets, the **adaptive** channels where packets can freely move minimally or non-minimally towards the destination and the **deterministic** or **escape** channels where any progress is made according to a deterministic algorithm [9] [16].

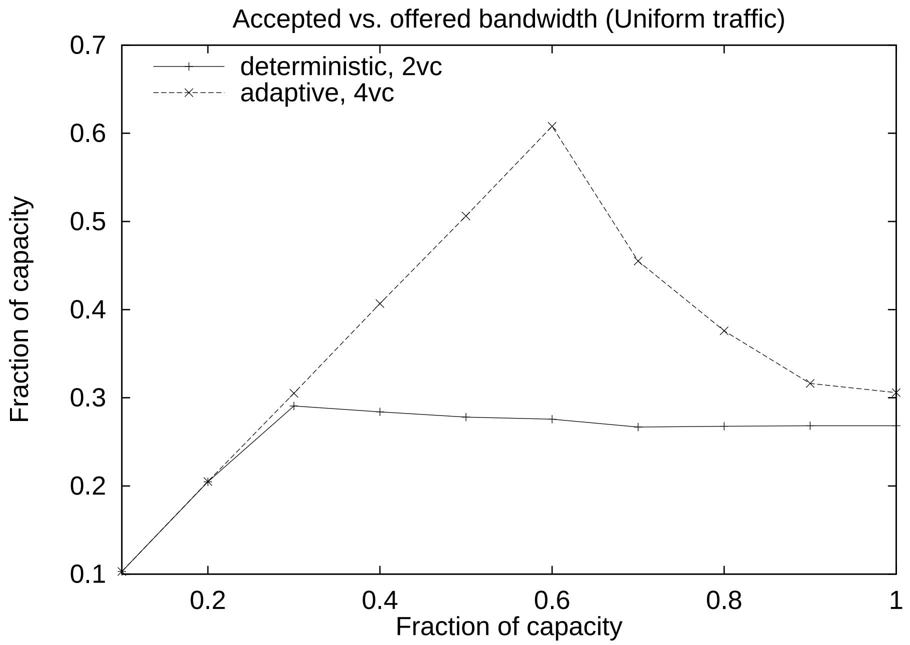


Figure 2: Throughput in a toroidal 2-ary 16-cube under random uniform traffic. We compare an adaptive algorithm based on Duato's methodology with four virtual channels and a deterministic dimension order routing with two virtual channels.

In this case, when the network operates above saturation, new packets coming from the processing nodes consume all available virtual channels in the adaptive pool and some of the deterministic channels. In Figure 2 we can see that the throughput of an adaptive algorithm based on Duato's methodology falls down to the level of a deterministic routing above saturation. Sustained throughput above saturation is an important parameter, because many parallel programs are either bandwidth-limited or bursty and work above the saturation point for long periods of their execution time [21]. It is worth noting that this is a peculiar problem of direct interconnection networks, because in multistage networks or fat-trees new packets can only be injected from the external levels and do not interfere with the intermediate routing switches [20].

In this paper we face the problems originated by the asymmetries introduced by the virtual channels by properly tuning the flow control algorithms that regulate the injection of new packets into the network.

# 3 The routing algorithm

An elegant approach to avoid deadlock in wormhole routed networks only requires the absence of cyclic dependencies on a subset of the virtual channels [15]. As a consequence, it is possible to build simple and efficient adaptive routing algorithms by dividing the virtual channels mapped on each physical link into two classes. Only one of these classes must guarantee deadlock freedom, while the remaining channels can be used in almost any way.

In our case, the first class is composed of two deterministic channels, that support an improved dimension order deterministic algorithm [16]. The remaining channels, defined adaptive channels, are used to route the packets on any available minimal path.

A central point of our algorithm is the interface between the processor and the router. We assume that packets can enter the network passing through a set of **injection** channels placed between the processor and the router. Also, these packets can only use a limited number of adaptive channels, defined **source throttling** channels.

We represent each node of the $k$-ary $n$-cube as an $n$-tuple $\langle x_0, \dots x_{n-1} \rangle$, $x_i \in \{0, \dots k-1\}$. Each node has $2 * n$ physical connections and several virtual channels or lanes are multiplexed on these links. The virtual channels are implemented by means of a set of input and output buffers. Figure 3 summarizes the notation used in the rest of the paper.

The routing algorithm, sketched in Figure 4, is logically organized in three steps.

1. If the current node where the routing decision is taken $\langle x_0, \dots x_{n-1} \rangle$ is equal to the destination $\langle y_0, \dots y_{n-1} \rangle$, the packet is sent to the local interface.

| | | |
|---|---|---|
| $n$ | = | the number of dimensions |
| $k$ | = | the number of nodes along each dimension |
| $2*n$ | = | the number of physical channels per node |
| $ada$ | = | the number of adaptive lanes |
| $inj$ | = | the number of injection lanes ($inj \leq ada$) |
| $thr$ | = | the number of source throttling lanes ($thr \leq ada$) |
| $a_{i,j}$ | = | adaptive lane, ( $0 \leq i < 2*n,\ 0 \leq j < ada$ ) |
| $d_{i,j}$ | = | deterministic lane, ( $0 \leq i < 2*n,\ 0 \leq j < 2$ ) |

Figure 3: Legend.

2. Otherwise we look for adaptive channels on any minimal path that leads to the destination. The predicate **free()** is true when is applied to an output lane that is not bound with any input lane and is empty. The emptiness condition requires that both output lane and the corresponding input lane in the partner node are empty, that is a queue must be fully emptied before accepting another header flit. Thus, when a packet is blocked, it will always occupy the head of an input lane.

3. If no adaptive lane is available, we must restrict the routing algorithm to the deterministic lanes. We find the first dimension $h$ where the current node and the destination differ. The predicate **positive_direction()** is true if the minimal path passes through the node in the positive direction of dimension $h$. Its tuple differs in the value of the $h$-th field, which is $(x_h+1) \bmod k$. We can use both lanes if the coordinate of the current node is less than the one of the destination or if we are on the positive border of the torus and one lane otherwise. Messages going towards negative destinations can be routed according to a symmetric relation. It is worth noting that the routing decision is not influenced by the provenience of the packet. The deterministic algorithms based on static channel dependency graphs [8] force a packet inside a given class of channels once it has entered one of them. The increased degree of flexibility offers a sensible performance improvement, as will be shown in the following sections. A formal proof of the deadlock freedom of this algorithm can be found in [15].

This basic algorithm suffers from some serious post-saturation problems. When the offered load reaches the saturation point the interconnection network becomes unstable and the performance drops down to the level of the basic deterministic algorithm [22]. We limit the injection of new packets by properly tuning two parameters:

1. the number of virtual channels that interface the processor with the router, called injection channels and

2. the number of adaptive channels on the physical links that can be used to route the packets, defined source throttling channels [9].

Also, deterministic lanes are prioritized over adaptive lanes. This provides a back pressure on the adaptive lanes that, on its turn, furtherly limits the injection of new packets.

Figures 5 and 6 provide a schematic description of the injection algorithm.

On the one hand, it has been shown that the global performance of the network is limited by a single injection channel [1]. On the other hand, if do not pose any limitation we can flood the network with packets that cannot easily make any progress towards the destination. In the following section we will analyze in depth the various tradeoffs originated by our approach.

# 4 The network model

This section presents a router model and a simulation environment, that are used in the following sections to analyze the performance of the $k$-ary $n$-cubes under random uniform traffic.

Figure 7 outlines the internal structure of the router. We can distinguish the external channels or links, the input and the output buffers that implement the virtual channels and an internal crossbar. The router has $2 * n$ bidirectional channels and each channel on the single direction is logically composed of three interfaces: a **data path** that transmits messages on a flit level, a **ready** line that flags the presence of a flit on the data path and specifies the virtual channel where the flit is to be stored and an **ack** line in the reverse direction that sends an acknowledgment every time buffer space is released in the input lanes. Also, there is another internal channel, not shown in the picture, interfacing the processor with router that contains the injection channels. Our algorithm is compared with the Chaos router, a cut-through version of the hot potato routing [5]. The Chaos router has a single packet-sized lane for each input and output buffer plus an internal multiqueue that can keep up to five packets.

A flit is moved from an output lane to the corresponding input lane in a neighboring node in $T_{link}$ cycles, when there is at least a free buffer position. Each output lane has associated a counter that is initialized with the total number of buffers in the input lane, it is decremented after sending a flit and it is incremented upon receiving and acknowledgment. When multiple lanes

```
if ( ⟨x_0, ... x_{n-1}⟩ = ⟨y_0, ... y_{n-1}⟩ )
then
    // the packet has reached the destination
else
    A = {a_{i,j} | free(a_{i,j}) ∧ a_{i,j} is on a minimal path towards ⟨y_0, ... y_{n-1}⟩}
    if ( A ≠ ∅ )
    then
        // route the packet on an adaptive lane
    else
        h = min{k | x_k ≠ y_k ∧ x_l = y_l, ∀l < k}
        if ( positive_direction( x_h, y_h ) )
        then
            if ( x_h < y_h ∨ x_h = n − 1 )
            then
                // try to route the packet on lanes d_{h,0} or d_{h,1}
            else
                // try to route the packet on lane d_{h,0}
        else
            if ( x_h > y_h ∨ x_h = 0 )
            then
                // try to route the packet on lanes d_{h+n,0} or d_{h+n,1}
            else
                // try to route the packet on lane d_{h+n,0}
```

Figure 4: The routing algorithm.

```
A' = {a_{i,j} | free(a_{i,j}) ∧ j < thr ∧ a_{i,j} is on a minimal path towards ⟨y_0, ... y_{n-1}⟩}
if ( A' ≠ ∅ )
then
    // route the packet on an adaptive lane
```

Figure 5: The injection algorithm.

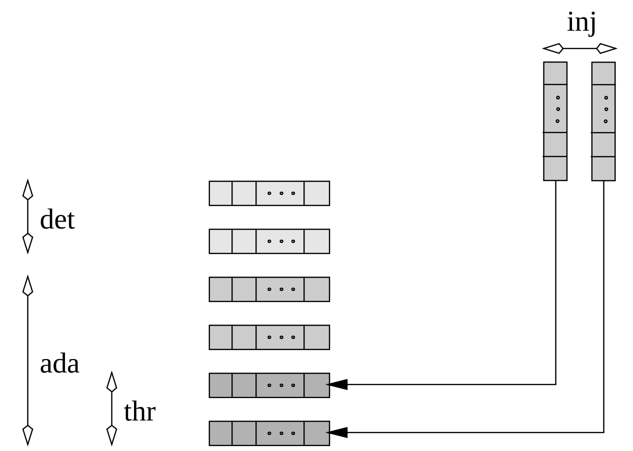


Figure 6: Limitations in the injection of new packets: $inj$ injection channels interface the processor with the router. They can only use the first $thr$ channels out of the $ada$ adaptive channels available on each physical channel.

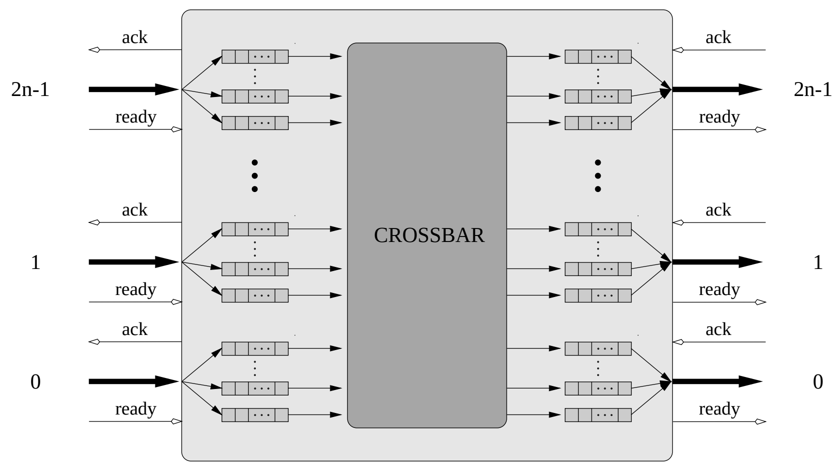


Figure 7: The internal structure of the router

are enabled, an arbiter picks one of them, giving priority to the deterministic lanes. When the flit is put on the data path, the ready line signals the virtual channel where the flit must be stored. Assuming $v$ virtual channels for each link, the ready line can be coded with $\lceil \log_2 v \rceil$ bits.

When a header flit reaches the top of an input lane, the routing algorithm tries to establish a path in the crossbar with a suitable output lane that satisfies the **free()** predicate. This path will remain in action till the transmission of the tail flit of the packet. Our model allows the routing of a single header at a time every $T_{routing}$ cycles. The extra complexity of a parallel router has been shown to give little or no advantage in terms of performance in the presence of wormhole or cut-through flow control [5].

Although a physical link services in each direction at most one virtual channel every $T_{link}$ cycles, multiple virtual channels can be active at the input and output ports of the crossbar. The internal flit propagation takes $T_{crossbar}$ cycles. Every time a flit is moved from an input lane to the corresponding output lane, a feedback is sent back to the neighboring node to update the counter of free positions.

This model is evaluated in the SMART (Simulator of Massive ARchitectures and Topologies) environment [23]. Implemented in C++, SMART is an object-oriented discrete-event simulation tool for evaluating massively parallel architectures. Configuring some shell scripts, it is possible to select the network topology, the internal router policies and the traffic pattern generated by each node. The simulator allows the definition of the packet length, inter-arrival times, number of virtual channels and buffers for both input and output lanes. Also, it is possible to monitor several metrics and time-dependent events, that are gathered in trace files. SMART supports three families of topologies: $k$-ary $n$-cubes, $k$-ary $n$-flies and $k$-ary $n$-trees and a node architecture with processing capabilities and a memory hierarchy.

The experiments in this paper evaluate a toroidal 16-ary 2-cube with 256 processing nodes. The number of virtual channels is varied between two and six and both input and output lanes have two buffer positions. This implies that there are four flit buffers for each virtual channel: the minimum buffer requirement is therefore 40 and the maximum 120 flit buffers. The link, routing and crossbar delays are all equalized to a single cycle, in order to model the router as a balanced pipeline. This can be considered unrealistic with the current technology: ongoing research on router implementations would suggest that the routing delay is at least two, three times the link and the crossbar delay [16] [5] [6]. However, some aggressive implementations tend to reduce such a delay with an internal pipelined architecture [18] [2] [13] [10]. Each node generates 20-flit packets with exponentially distributed inter-arrival times; the destinations are uniformly distributed over the network. The simulator collects performance data only after 4000 cycles, to allow the network to reach steady state and each simulation is halted after

20000 cycles.

Given that the Chaos router require two packet buffers for each physical link plus five internal packet buffers in the multiqueue, the internal buffer space is 300 flit buffers.

# 5 Experimental results

The performance of an interconnection network under dynamic load is usually assessed by two quantitative parameters, the **accepted bandwidth** or **throughput** and the **latency**.

Accepted bandwidth is defined as the sustained data delivery rate given some offered bandwidth at the network input. Two important characteristics are the saturation point and the sustained rate after saturation. Saturation is defined as the minimum offered bandwidth where the accepted bandwidth is lower than the global packet creation rate at the source nodes. It is worth noting that, before saturation, offered and accepted bandwidth are the same. The behavior above saturation is important because the network and/or the routing algorithm can become unstable, leading to a sharp performance degradation. We usually expect the accepted bandwidth to remain stable after saturation, both in the presence of bursty applications that require peak performance for a short period of time and applications that operate after saturation in normal conditions, e.g. when executing a global communication pattern.

The **network** latency is the average delay spent by a packet in the network, from the insertion of the trailing flit in the router till the reception of the tail flit at the destination. It does not include the source queuing delay. The **end-to-end latency** rises to infinity above saturation and is impossible to gain any information in this case. For this reason, the network latency is often preferred to analyze the network performance.

Table 1 reports the performance of several injection configurations measured at the network capacity. When we use a single adaptive channel (we recall that there are two deterministic channels that must be added to the number of adaptive channels to obtain the total number of virtual channels multiplexed on each physical link) the minimal injection mechanism composed of a single injection and a single source throttling channel provides an acceptable throughput, with 46% of the network capacity, even if it is possible to get a better performance by furtherly limiting the injection of new packets. We can get 53% of the network capacity if we limit the injection of new packets to the last dimension where the source and the destination nodes differ.

The impact of the limited injection mechanisms can be clearly seen with two adaptive channels. The basic configuration with an injection and a source throttling channel reaches 66% of the network capacity with only 125 cycles of average network latency. With two injection channels the throughput drops to about 35% with either one or two source throttling channels. In this case a single injection channel suffices to obtain a good performance.

With four adaptive channels a single injection channel limits the performance to the results obtained with two adaptive channels. On the other hand, all solutions with four injection channels are too loose to guarantee network stability. Two injection channels and a source throttling channel give interesting performance results with 78% of the network capacity. Even if there is a big gap between one and two source throttling channels (the throughput falls down to 40%), these configurations are very similar and the second solution provides only a small increase of injection capability. We can measure the degree of injection with the average number of active injection channels above saturation. Both injection channels are active with two source throttling channels. The degree of injection with a single source throttling channel can be estimated computing the average of the random variable $X$, that represents the number of active channels. The probability of a single active channel, representing two packets that are being sent to destinations that differ in the same single coordinate from the source node, is

$$P(X=1) \;\approx\; 2n\left(\frac{k/2}{k^n}\right)^2 \tag{1}$$

that on 16-ary 2-cube gives

$$P(X=1) \;\approx\; \frac{1}{256}, \quad P(X=2) \;\approx\; 1-\frac{1}{256}, \qquad E(X) \approx 2-\frac{1}{256} \tag{2}$$

where $E(X)$ is the average number of active injection channels. This proves that this configuration is very close to optimality.

Figure 8 and 9 compare the selected configurations of the adaptive algorithm with the deterministic algorithms and the Chaos router. At first glance, we can see in Figure 8 the big gap between the Chaos router, that saturates at 83% of the capacity and the deterministic algorithm based on a static channel dependency graph, whose saturation point is below 30%. The adoption of a dynamic channel dependency graph increases the throughput of the deterministic algorithm of about 10%. We can also see that the adaptive algorithm with an adaptive channel (or three virtual channels, if we consider the deterministic ones) is not properly damped by the limitation mechanisms and its performance experiences a modest drop. Two adaptive channels are probably the best tradeoff between throughput and network latency, with 125 cycles at saturation (only three times the base network latency obtained in the absence of contention) and 66% of the network capacity. Two more adaptive channels narrow the distance with the Chaos router, but this comes

| $ada$ | $inj$ | $thr$ | **Throughput** | **Latency** |
|---|---|---|---|---|
| 1 | 1 | 1 | 0.46 | 163 |
| 2 | 1 | 1 | 0.66 | 125 |
| 2 | 2 | 1 | 0.34 | 242 |
| 2 | 2 | 2 | 0.35 | 354 |
| 4 | 1 | 1 | 0.67 | 131 |
| 4 | 2 | 1 | 0.78 | 203 |
| 4 | 2 | 2 | 0.40 | 304 |
| 4 | 4 | 1 | 0.52 | 444 |
| 4 | 4 | 2 | 0.32 | 624 |
| 4 | 4 | 4 | 0.25 | 592 |

Table 1: Throughput and network latency of some injection configurations when the applied load is equal to the network capacity. The configurations used in the experiments are separated with double horizontal lines.

at the price of an increased network latency, which passes from 125 to more than 200 cycles. This is caused by two factors: the first one is the degree of multiplexing that increases the tail latency, the time needed to absorb the rest of a packet once the header flit has reached the destination. The second factor is the increased number of injection channels which augments the source queuing delay. Other performance results that analyze the performance under bit reversal, complement and transpose traffics, not reported here for brevity, show that the limitation mechanisms have no negative side effects.

It is worth noting that, while these limitation mechanisms can be successfully applied to all the class of $k$-ary $n$-cubes, the optimal configurations vary from network to network. The optimal configuration is also sensitive to the router performance. For example on a bi-dimensional torus with 64 nodes we have an increased injection capability: optimal configurations use more injection and source throttling lanes when the router has the same number of virtual channels. On the other hand, larger networks require less injection and source throttling lanes.

# 6 Conclusion

Virtual channels are an important flow control strategy to avoid deadlocks and to improve performance of wormhole routed networks. Unfortunately, virtual channels introduce asymmetries in the use of buffers in symmetric

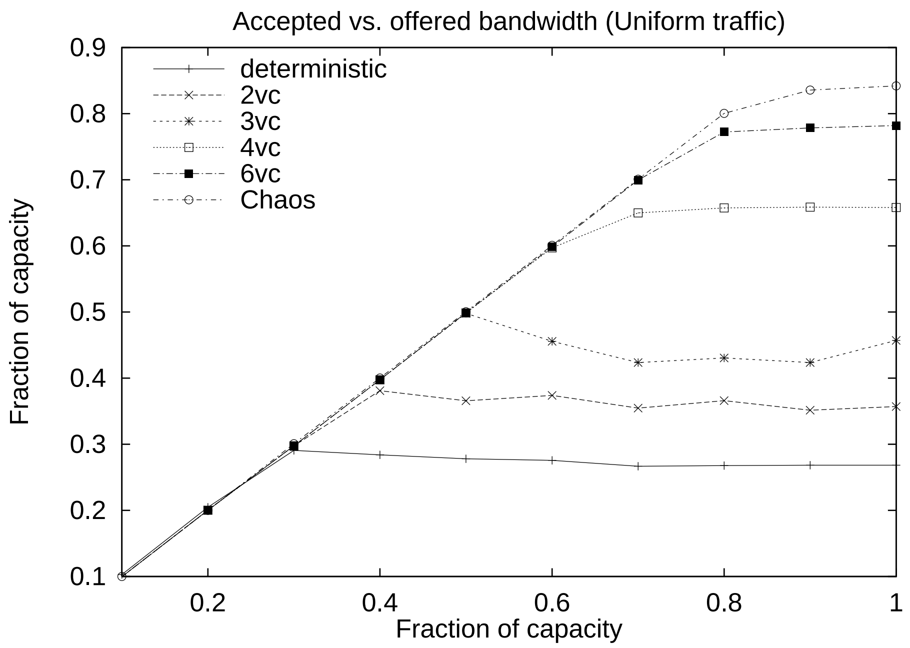


Figure 8: Network throughput

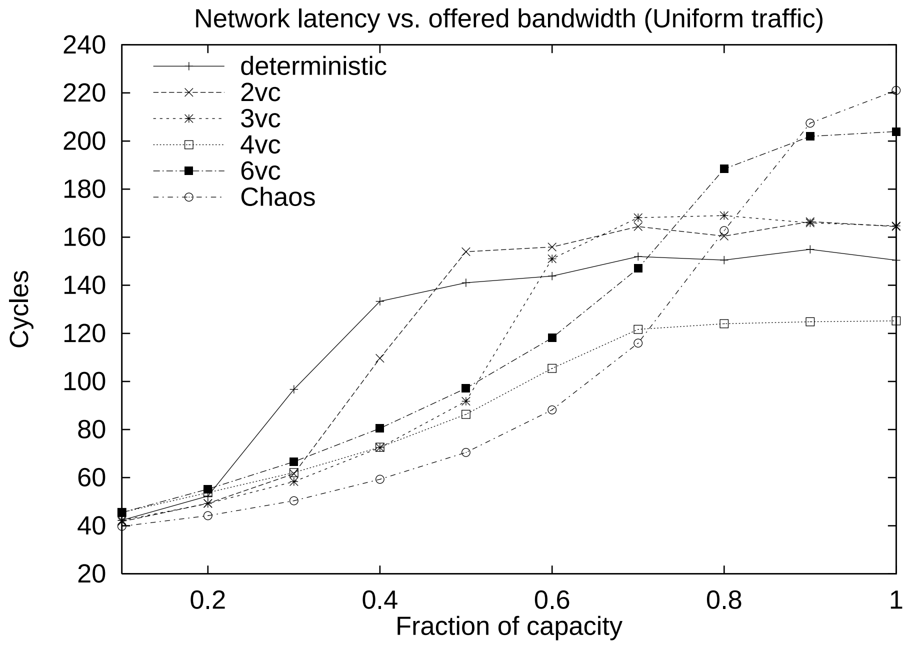


Figure 9: Network latency

networks as the toroidal $k$-ary $n$-cubes. In this paper we have presented a minimal adaptive routing algorithm, based on Duato's methodology [15], that tries to balance the use of the virtual channels by limiting the injection of new packets into the network.

The basic algorithm divides the virtual channels into two classes, the deterministic channels, where packets can advance on according to a dimension order routing algorithm and adaptive channels, that can be used in almost any way. We have augmented the basic algorithm with two injection mechanisms.

1. We limit the number of injection channels, those that interface the processor with the router, to a number smaller or equal to the number of adaptive lanes.

2. We also force packets coming from the injection lanes to use a subset of the adaptive lanes, called source throttling lanes.

These mechanisms have been extensively analyzed through simulation on a toroidal 16-ary 2-cube with 256 nodes. The experimental results have proved that, by properly tuning the injection mechanisms, it is possible to increase the saturation point and keep the network throughput stable after saturation. For example, a properly damped version with four adaptive channels has reached 78% of the capacity, far more than the 25% of the basic algorithm.

The algorithm has been compared with the Chaos router, a non minimal cut-through version of the hot-potato routing which saturates at 83% of the capacity. The version with four adaptive channels has provided a comparable performance using only a small fraction of buffers (120 vs 300) and a simpler router organization. The version with two adaptive channels has also shown an interesting performance compromise, with 66% of the network capacity and only 125 cycles of network latency at saturation, about half the latency of the Chaos router.

# References


[1] Vikram S. Adve and Mary K. Vernon. Performance Analysis of Mesh Interconnection Networks with Deterministic Routing. *IEEE Transactions on Parallel and Distributed Systems*, 5(3):225–246, March 1994.

[2] J. D. Allen, P. T. Gaughan, D. E. Schimmel, and S. Yalamanchili. Ariadne - An Adaptive Router for Fault-tolerant Multicomputers. *Computer Architecture News (Special Issue ISCA'21 Proceedings)*, 22(2):278–288, April 1994.

[3] P. E. Berman, L. Gravano, and G. D. Pifarre. Adaptive Deadlock- and Livelock-Free Routing with all Minimal Paths in Torus Networks. In *Proceedings of the Fourth Symposium on Parallel Architectures and Algorithms*, pages 3–12, 1992.

[4] Kevin Bolding. Non-Uniformities Introduced by Virtual Channel Deadlock Prevention. Technical Report UW-CSE 92-07-07, University of Washington, Department of Computer Science and Engineering, Seattle WA, July 1992.

[5] Kevin Bolding. *Chaotic Routing: Design and Implementation of an Adaptive Multicomputer Network Router.* PhD thesis, University of Washington, Department of Computer Science and Engineering, Seattle, WA, July 1993.

[6] Kevin Bolding, Melanie L. Fulgham, and Lawrence Snyder. The Case for Chaotic Adaptive Routing. Technical Report UW-CSE 94-02-04, University of Washington, Department of Computer Science and Engineering, Seattle WA, February 1994.

[7] Andrew A. Chien and Jae K. Kim. Planar Adaptive Routing: Low-Cost Adaptive Networks for Multiprocessors. In *Proceedings of the 19th Annual Symposium on Computer Architecture*, pages 268–277, May 1992.

[8] William J. Dally. Virtual Channel Flow Control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):194–205, March 1992.

[9] William J. Dally and Hiromichi Aoki. Deadlock-Free Adaptive Routing in Multicomputer Networks Using Virtual Channels. *IEEE Transactions on Parallel and Distributed Systems*, 4(4):466–475, April 1993.

[10] William J. Dally, Larry R. Dennison, David Harris, Kinhong Kan, and Thucydides Xanthopoulos. The Reliable Router: A Reliable and High-Performance Communication Substrate for Parallel Computers. In Kevin Bolding and Lawrence Snyder, editors, *First International Workshop, PCRCW'94*, volume 853 of *LNCS*, pages 241–255, Seattle, Washington, USA, May 1994.

[11] William J. Dally and Charles L. Seitz. The Torus Routing Chip. *Distributed Computing*, 1:187–196, 1986.

[12] William J. Dally and Charles L. Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Transactions on Computers*, C-36(5):547–553, May 1987.

[13] A. DeHon, F. Chong, M. Becker, E. Egozy, H. Minsky, S. Peretz, and T. F. Knight Jr. Metro: A Router Architecture for High-Performance, Short-Haul Routing Networks. *Computer Architecture News (Special Issue ISCA'21 Proceedings)*, 22(2):266–277, April 1994.

[14] José Duato. Deadlock-Free Adaptive Routing Algorithms Multicomputers: Evaluation of a New Algorithm. In *Proceedings of the Third IEEE Symposium in Parallel and Distributed Processing*, pages 840–847, December 1991.

[15] José Duato. A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks. *IEEE Transactions on Parallel and Distributed Systems*, 6(10):1055–1067, October 1995.

[16] José Duato and Pedro López. Performance Evaluation of Adaptive Routing Algorithms for $k$-ary $n$-cubes. In Kevin Bolding and Lawrence Snyder, editors, *First International Workshop, PCRCW'94*, volume 853 of *LNCS*, pages 45–59, Seattle, Washington, USA, May 1994.

[17] C. J. Glass and L. M. Ni. The Turn Model for Adaptive Routing. In *Proceedings of the 19th Annual Symposium on Computer Architecture*, pages 278–287, May 1992.

[18] J. H. Kim, Ziqiang Liu, and Andrew A. Chien. Compressionless Routing: A Framework for Adaptive Fault-tolerant Routing. *Computer Architecture News (Special Issue ISCA'21 Proceedings)*, 22(2):289–300, April 1994.

[19] D. H. Linder and J. C. Harden. An adaptive and Fault-Tolerant Wormhole Routing Strategy for $k$-ary $n$-cubes. *IEEE Transactions on Computers*, 40(1):2–12, January 1991.

[20] Fabrizio Petrini and Marco Vanneschi. $k$-ary $n$-trees: High Performance Networks for Massively Parallel Architectures. Technical Report TR-95-18, Dipartimento di Informatica, Università di Pisa, December 1995.

[21] Fabrizio Petrini and Marco Vanneschi. Latency and Bandwidth Requirements of Massively Parallel Programs: FFT as a Case Study. Technical Report TR-96-2, Dipartimento di Informatica, Università di Pisa, March 1996. Also in Europar96.

[22] Fabrizio Petrini and Marco Vanneschi. Minimal vs. non Minimal Adaptive Routing on $k$-ary $n$-cubes. In *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'96)*, Sunnyvale, California, August 1996.

[23] Fabrizio Petrini and Marco Vanneschi. SMART: a Simulator of Massive ARchitectures and Topologies. submitted to the Third International Symposium on High-Performance Computer Architecture, July 1996.

## Author Biographies

**Fabrizio Petrini** received the Laurea degree in Computer Science from the Computer Science Department of the University of Pisa, Italy, in 1990. He is presently a doctoral candidate in Computer Science at the same Department. He was a research fellow at the Hewlett Packard Pisa Science Center from 1990 to 1993. He also worked at the Hewlett Packard Labs in Palo Alto, CA, in the summer 1991.

His research interests include parallel architectures and algorithms, interconnection networks and abstract machines for massively parallel architectures. He is a student member of the IEEE.

**Marco Vanneschi** graduated in Electronic Engineering at the University of Pisa in 1970. In 1973 he joined the University of Pisa, Department of Computer Science, as assistant professor in Computer Architecture. Since 1981 he is full professor in Computer Science at the same Department.

He was the coordinator of the national project on Parallel Architectures (1989 - 1994), funded by the National Research Council (CNR). He is member of IFIP Working Group 10.3 on Parallel and Distributed Processing and of the coordination group of the ERCIM Parallel Processing Network. He is the head of the scientific committee of the PQE2000 project. The PQE2000 project aims at developing a massively parallel architecture and involves several industries and academic institutions.

His research interests are focused on design and utilization of massively parallel processing and scalable machines: architectures, programming models and methodologies, development and evaluation tools for parallel applications. He is author of more than 90 papers published in international journals and conference proceedings, and of three books on basic computer architecture, advanced computer architecture, and concurrent programming.