Layered Routing in Irregular Networks

Olav Lysne, *Member*, *IEEE*, Tor Skeie, Sven-Arne Reinemo, and Ingebjørg Theiss

Abstract—Freedom from deadlock is a key issue in Cut-Through, Wormhole, and Store and Forward networks, and such freedom is usually obtained through careful design of the routing algorithm. Most existing deadlock-free routing methods for irregular topologies do, however, impose severe limitations on the available routing paths. We present a method called Layered Routing, which gives rise to a series of routing algorithms, some of which perform considerably better than previous ones. Our method groups virtual channels into network layers and to each layer it assigns a limited set of source/destination address pairs. This separation of traffic yields a significant increase in routing efficiency. We show how the method can be used to improve the performance of irregular networks, both through load balancing and by guaranteeing shortest-path routing. The method is simple to implement, and its application does not require any features in the switches other than the existence of a modest number of virtual channels. The performance of the approach is evaluated through extensive experiments within three classes of technologies. These experiments reveal a need for virtual channels as well as an improvement in throughput for each technology class.

Index Terms—Routing functions, interprocessor communication, deadlock avoidance, irregular topologies.

1 Introduction

Following the introduction of networks of workstations to the multiprocessor market, interest in irregular topologies for computer interconnects has increased. Such networks have the advantage that they offer greater wiring flexibility than regular structures such as meshes, tori, and multistage networks. Flexibility in wiring is particularly important for fault tolerance, since networks that were at first regular become irregular when some components fail. For these reasons, a series of commercial technologies that do not place constraints on network topology has emerged [2], [3], [11], [12], [16].

However, routing of irregular networks is more complex, because the routing method needs to be topology agnostic. A well-known method for generating routing algorithms in irregular networks is the UpDown strategy that was first presented in connection with Autonet [16] and later used in Myrinet [3]. It relies on a technique for generating a spanning tree of the network, where the key issue is to assign up directions to all links, such that the root of the spanning tree can be reached from any node following links in the up direction only. Routing is restricted so that no packet can traverse a link in the up direction after having traversed a link in the down direction. Sancho et al. proposed an improvement of the UpDown routing scheme in [19]. Instead of using the usual breadth-first traversal in the generation of the spanning tree, they apply a depth-first search that optimizes the spanning tree. This gives greater flexibility during the generation of the routing tables and, therefore, leads to improved performance. Qiao and Ni take a different approach to the routing of irregular networks [18]. Their method is based on an underlying Eulerian graph, and it routes adaptively between two acyclic unidirectional trails (paths) that contain all the channels (edges) of the network. In order to have more optimized routing paths, different types of short-cut channels are added following heuristic criteria. Another body of work has focused on improving the performance of irregular networks through methods such as virtual channel multiplexing, adaptivity, and shortest-path routing combined with escape paths [20], [22], [23], [24].

The issue of efficient routing in irregular networks was addressed by Flich et al. in [9], [10], where they propose a source routing method where the messages always use shortest paths. The basic idea is to eliminate the restrictions imposed by UpDown routing by letting intermediate nodes eject messages completely and later reinject them, breaking the dependencies of illegal turns. In [21], the same idea was used in an InfiniBand setting, but here illegal turns were broken by letting the packets that use these turns ascend to a higher layer of virtual channels.

This paper presents a concept that we call *layered routing*. The network resources are divided into *layers* and network deadlocks are avoided by preventing portions of traffic from using specific layers. This contrasts with previous approaches that handle deadlock by preventing data packets from using specific paths and thereby severely restrict routing freedom in irregular networks. Our method increases the performance of deterministic and source adaptive routing, meaning that the area of application of our approach is different from the body of work that aims at maximizing the adaptivity in the switches. In addition, our method needs only a very limited number of virtual channels, even for large network sizes, which makes it directly applicable to present-day technologies such as InfiniBand™ [12] and Advanced Switching [2].

Previous solutions related to ours include the approach taken in the Avici Terabit Switch/Router [4], where separate virtual networks (layers) are used for each destination port in a torus topology. This approach will, however, require a very high number of virtual channels for

• *The authors are with Simula Research Laboratory, Box 134 Lysaker, N-0325 Lysaker, Norway.*
E-mail: {olav.lysne, tskeie, svenar, theiss}@simula.no.

Manuscript received 6 Nov. 2003; revised 9 Jan. 2005; accepted 17 Feb. 2005; published online 28 Nov. 2005.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0205-1103.

large networks. It is therefore not suited to present-day technologies such as InfiniBand [12] and Advanced Switching [2]. Another body of related work aims at increasing the adaptivity in the switches. In [13], Linder and Harden achieved deadlock-free, minimal, and adaptive layered routing using virtual channels for regular networks, in particular, for k-ary n-cubes. This method does not, however, generalize easily to arbitrary topologies, and the need for virtual channels grows exponentially with n. Yet another idea is to order the layers. Packets escape from possible deadlocks in a higher layer by making a transition down to a lower level. If the lowest layer is deadlock-free, the entire system will also be [1], [6], [7], [8], [14].

There are problems connected with all of the above approaches. Some of them require extra functionality in the switches or the hosts that not all technologies provide. Others aim at maximizing adaptivity, which results in the delivery of packets being out of order. The extra protocol overhead involved in sorting the packets at the destination is, in some cases, unacceptable, and this is the main reason why many technologies only use deterministic routing.

Our layered routing methodology is very flexible and can be used in a variety of ways. First, we use it to obtain load balancing in the network, then we consider shortest-path routing in irregular networks and various forms of routing adaptivity. Our concept assumes the presence of virtual channels, but otherwise requires no special functionality within the switches.

The paper is organized as follows: In Section 2, we give some basic definitions and notations before we define the concept of layered routing in Section 3. In Section 4, we show how layered routing can be used to obtain load balancing in an UpDown routed network. In Section 5, we present an algorithm that generates deadlock-free and deterministic routing functions with shortest paths using layered routing and, in Sections 6 and 7, we extend the concept of shortest-path layered routing to multiple paths and adaptive routing. Simulation experiments, in which a series of randomly generated topologies were tested for network performance, are described in Section 8. Section 9 concludes.

2 Preliminaries

The definitions in this section mostly adhere to the standard notation and definitions of cut-through switching and graph theory.

Definition 1. *A* network I *is represented by a strongly connected directed graph, $I = G(N, C)$. The vertices of I are the set of nodes (switches) N, whereas the edges are the set of communication channels (possibly virtual), C. Each channel is unidirectional and transmits data from a source node to a destination node. A network channel c_i interconnects the two nodes $src(c_i)$, $dst(c_i) \in N$, which are the source and destination of the channel, respectively.*

A link *is a set of channels $c_{l_1}, c_{l_2}, \ldots, c_{l_n}$ such that either $src(c_{l_i}) = src(c_{l_j})$ and $dst(c_{l_i}) = dst(c_{l_j})$, or $src(c_{l_i}) = dst(c_{l_j})$ and $dst(c_{l_i}) = src(c_{l_j})$ for all i and j.*

Each channel $c \in C$ is part of exactly one link. A subset of the nodes N in the network are called compute nodes. *These are the nodes that generate and consume data traffic.*

We assign a set of *virtual addresses* to each compute node in the network. The set of available virtual addresses is denoted $\mathcal{A}$.

Definition 2. *An* address assignment function $A : \mathcal{A} \longrightarrow N$ *for network I is a function that takes an address a as input and returns a compute node $n \in N$.*

The notion of virtual addresses allows us to have a set of addresses assigned to each destination. This is a feature that is implemented, e.g., in InfiniBand, and it opens up the possibility for multiple paths to exist between each source and destination, even if each switch only implements deterministic routing.

Definition 3. *For network $I = G(N, C)$ and an address assignment function A, a* routing function $R : N \times C \times \mathcal{A} \longrightarrow \mathcal{P}(C)$, *where $\mathcal{P}(C)$ is the power set of C, takes a node n, an input channel c, and a destination address a as parameters, and returns a set of output channels that can be taken from node n for packets entering its channel c and whose destination is $A(a)$. A routing function R is* deterministic *if for all (n, c, a) combinations $R(n, c, a)$ is always singleton; otherwise, it is* adaptive.

Notice that this definition of a routing function allows a node to select different output channels depending on what input channel a packet arrives on. This is not possible in all technologies; indeed, InfiniBand is an example of a technology that will not take input channels into account. However, methods we present in this paper are applicable to any case.

Definition 4. *For a network I, address assignment function A, and routing function R, there exists a* dependency *from channel c_i to c_j iff $c_j = R(dst(c_i), c_i, a)$ for some address a. That is, packets destined for $A(a)$ may use c_j immediately after c_i.*

Definition 5. *The* channel dependency graph *of a network I with respect to a routing function R is a directed graph in which the channels of I constitute the vertices and the dependencies constitute the arcs.*

The following theorem is a straightforward adaptation of a theorem due to Dally and Seitz [5].

Theorem 1. *A network is* free from deadlocks *if the channel dependency graph of its routing function is acyclic.*

The original formulation of Theorem 1 supplied a *necessary* and sufficient condition for freedom from deadlock, but for deterministic routing only. However, it is well-known that the premise in the theorem is a sufficient (but not necessary) condition for deadlock free adaptive routing as well. Sufficient and necessary conditions for deadlock-free adaptive routing functions were later provided by Duato [6], [7].

3 Layered Routing

Our basic mechanism is to divide the physical network into a set of layers, each layer representing a virtual network with the same connectivity as the original physical one.

Definition 6. *A* network layer L_i *of network* I *is a subset of the virtual channels in* I *such that each link has exactly two channels in* L_i*, one in each direction.*

Definition 7. *A set* L *of network layers* $\{L_1, \ldots, L_n\}$ *is a* layering *of a network* I *iff for* $1 \leq i \leq n$ *and* $1 \leq j \leq n$

- L_i *is a layer of* I *for all* i*,*
- L_i *and* L_j *are disjoint for all distinct* i *and* j*, and*
- *for each channel* c *in* I*, there exists an* L_i *such that* c *is in* L_i*.*

The above two definitions allow us to view any layer of a network as a bidirectional virtual network that is isomorphic to the original physical network.

Definition 8. *For a layering* L *of network* I *and a routing function* R*, we say that* R *is* layered *with respect to* L *if* $R(n, c_i, a) = c_j$ *implies that* c_i *and* c_j *are in the same layer* $L_j \in L$ *for all* n*,* c_i*,* a *and* c_j*. By* R_k*, we denote the subrouting function of* R *that is restricted to* $L_k \in L$*.*

This means that a layered routing function will let all packets remain in the layer into which it was first injected. Furthermore, R_i contains all necessary information on the forwarding that can take place in L_i.

Most of the routing functions we develop in this paper are layered and, therefore, we develop below a theory for freedom from deadlock for layered routing functions. This concept will be extended in a later section, where we discuss adaptivity between the layers.

In order to assign names to data streams, we need to define *source/destination pairs* and *source/address pairs*.

Definition 9. *A source/destination pair is an ordered pair of compute nodes, where the first element is called the source and the second element is called the destination. A source/address pair is an ordered pair where the first element is a compute node and the second element is an address. If* $\langle s, a \rangle$ *is a source/address pair and* A *is the relevant address assignment function, then* $\langle s, A(a) \rangle$ *is said to be the* corresponding *source/destination pair.*

Notice that, if more than one address is associated with a compute node, we are able to distinguish between data streams using the different addresses by referring to source/address pairs, rather than source/destination pairs.

Definition 10. *For a layering* L *of network* I*, a* traffic assignment function T *of* L *takes a source/address pair as input and returns a (not necessarily strict) subset of* L*.*

Intuitively, we assume that $T(\langle s, a \rangle)$ is the set of layers that can be used to transmit the traffic from node s to node $A(a)$, using virtual address a as the address in the packet header. This may be a singleton layer, meaning that there is only one allowed layer for this traffic. It can be a strict subset of L or it can be all layers of L, meaning that traffic from s using a as its address can use any layer.

Definition 11. *For a layering* L *of network* I*, a layered routing function* R*, and a traffic assignment function* T *of this layering, there exists a* layered dependency *from channel* c_i *to* c_j *iff there exists a layer* $L_k \in L$*, a source* s*, and a virtual address* a *such that* $c_j = R(dst(c_i), c_i, a)$*, both* c_i *and* c_j *are members of* L_k*, and* $L_k \in T(\langle s, a \rangle)$*.*

The above definition states that there exists a layered dependency from one channel in I to another channel in I if there are packets that, according to L, R, and T, will use the second channel immediately after the first. This is a natural extension into layered routing of the notion of dependency described in the previous section. The notion of dependency graphs and its connection to freedom from deadlock are extended in the same way.

Definition 12. *For a layering* L *of network* I*, a layered routing function* R*, and a traffic assignment function* T *of this layering, the* layered dependency graph of I *with respect to* R*,* L*, and* T *is a directed graph in which the channels of* I *constitute the vertices and the layered dependencies constitute the arcs.*

Theorem 2. *For a layering* L*, a layered routing function* R*, and a traffic assignment function* T*, network* I *is* free from deadlocks *if its layered channel dependency graph with respect to* L*,* R*, and* T *is acyclic.*

The proof of this theorem is a simple rewrite of the proof that was presented by Dally and Seitz [5], in which all references to channel dependencies are replaced with layered (channel) dependencies. The proof is therefore omitted.

4 Load-Balancing of Up-Down Routing (MROOTS)

The predominant routing paradigm for irregular networks is the UpDown scheme. A well-known weakness of this routing scheme is that it has a strong tendency to generate hot spots around the root of the UpDown structure. In this section, we show how to alleviate this tendency for hot spots by using layered routing.

4.1 UpDown Routing

A distributed algorithm for generating deadlock-free routing functions in irregular networks is described in connection with Autonet [16]. It is based on an algorithm for generating a breadth-first spanning tree of a network that was first presented by Perlman [17].

The basic requirement for the algorithm is that each node be able to communicate with its immediate neighbours. The task of the algorithm is first to choose one node to be the *root* of the UpDown structure, and second to assign a direction to all the links in the network. The directions must be chosen such that

- there are no cycles that follow links only in the up direction,
- there are no cycles that follow links only in the down direction,
- from any node one will reach the root if one follows links only in the up direction, and
- from the root one may reach any node by following links only in the down direction.

In particular, one may choose to let the root be the node with the lowest ID. Thereafter, the directions of the links are chosen so that, for each link, the up-end is the one closer to the root. If the direction of the link is not well-defined from this rule, the up-end is chosen to be at the node with the

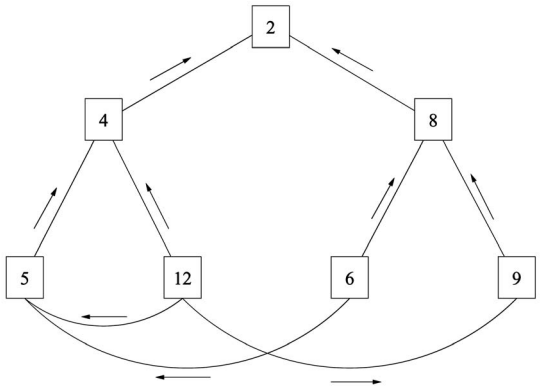

Fig. 1. Assignment of directions to links. The arrows indicate the up direction of the links.

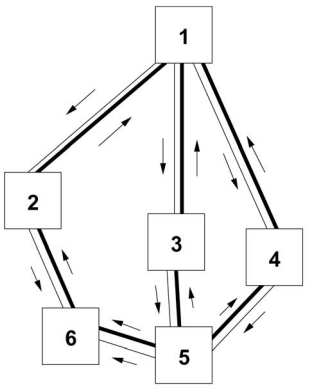

Fig. 2. This figure illustrates a network with layers (black and white). The black layer has node no. 1 as the root, and the white layer has node no. 6 as the root. The arrows indicate the up-direction for each channel.

lower ID. Fig. 1 shows an example of an irregular topology, with the direction of the links assigned by the algorithm indicated by arrows.

In the next phase of the algorithm, the topology of the entire spanning tree is distributed to all nodes, which use this information to fill their routing tables. Freedom from deadlock is guaranteed by a restriction such that no node must allow a packet to use an up-link after it has used a down-link. This does not jeopardize connectedness, as there will always be a path between any pair of nodes that first follows a (possibly empty) sequence of links in the up direction, followed by a (possibly empty) sequence of links in the down direction. For a complete description of the original algorithm, see [16] and, for a more recent improvement, see [19].

4.2 Multiple Roots

The UpDown algorithm is able to find a deadlock-free routing function for any arbitrary network topology. Its drawback is that all network traffic is initially directed toward the root, which renders the area around the root a bottleneck in the system. This reduces the overall performance of the network.

Our approach is to use layered routing to distribute the load of the hot-spots generated by the UpDown routing algorithm to various places in the network. We assume a network I and a layering $L = \{L_1, \ldots, L_m\}$ of I. Furthermore, we assume an address assignment function A that assigns a single address from $\mathcal{A}$ to each compute node. The method works as follows:

Step 1: For each layer $L_i \in L$, choose a node n_i^r—the root node of layer i—in such a way that the shortest distance between n_i^r and the root node n_j^r chosen for some previously treated layer j is maximized. If this is ambiguous, choose the node with the lowest node ID.

Step 2: For each layer $L_i \in L$, generate the routing function R_i by using UpDown routing with n_i^r as the root of the UpDown structure. Let $R = \cup_{i=1}^{m} R_i$.

Step 3: Generate the traffic assignment function T according to the requirements for packet ordering: If the packets are required to be delivered in order, ensure that T returns with a singleton layer for every source/address pair, otherwise let T return all layers in L for every source/address pair.

The above guarantees that a separate UpDown structure will be built for each root, which will result in a separate routing function for each layer. The distance between the roots of these layers is maximized, thus the potential hot-spots near the roots are distributed. Freedom from deadlock is guaranteed by layered routing, since each routing function R_i will work on its own separate set of virtual channels. The nodes injecting packets into the network can decide which layer a packet should be injected into based on T.

Fig. 2 illustrates a network with two layers, each with a different UpDown structure. In this figure, we see that, in the layer depicted by black virtual channels, switch 1 is a potential hot spot. In the layer with white channels, however, the potential hot-spot created by the UpDown routing algorithm will be centred around switches 2, 6, and 5. In sum, the potential hot-spots generated by the UpDown routing algorithm are distributed fairly evenly around the network.

4.3 A Distributed Algorithm to Compute the MROOTS Routing Tables

A feature of the UpDown algorithm is that it is completely distributed. The choice of root, the distribution of the topology information, and the setup of routing tables is done by the switches themselves communicating only with their immediate neighbors. Even if this requires a certain amount of additional functionality in the switches that not all technologies provide, its advantages in resilience are so significant that we should provide a distributed algorithm for our approach to load balancing as well. Fortunately, this is fairly straightforward.

Note that, at a given point in the standard UpDown algorithm, all nodes get information about the entire topology. If, from this point, they use a deterministic algorithm to calculate the roots and the structure of each layer, they will all conclude with the same result. This means that all nodes can calculate the structure of the UpDown trees of the new layers independently. From this structure they can, in turn, calculate their routing tables. Regarding the traffic assignment function, the easiest solution is to require all sources to distribute their traffic as evenly between the layers as possible.

5 Layered Shortest Path Routing (LASH)

One general problem that arises in irregular network structures is that it is difficult to achieve deadlock-free routing where every packet takes the shortest physical path. As a result, most existing methods for shortest-path routing in irregular networks provide shortest paths only relative to some constraint. UpDown routing as defined above is an example of this, in that it supports shortest paths only relative to the constraint that no up channel can be used after a down channel.

In this section, we describe how one can use layered routing to allow for true shortest-path routing in irregular topologies. This is done by first identifying a routing function R that finds one shortest physical path between every source and destination. Thereafter, we generate a traffic assignment function T that assigns the source/destination pairs to different layers in such a way that freedom from deadlock in each individual layer is guaranteed.

5.1 Generation of R and T

Below, we give an algorithm for mapping source/destination pairs onto virtual layers. We assume that a network I and a layering L of that network is given. Furthermore, we assume that L has n layers and that we have an address assignment function A that assigns exactly one address to each traffic node in I.

Step 1: Let $T(\langle s, a\rangle) =$ undefined for all source/address pairs, let R_i be empty for all i such that $1 \leq i \leq n$, and let $A(a)$ be undefined for all addresses a.

Step 2: Take one pair of source and destination $\langle s, d\rangle$ that has not yet been processed. For an arbitrary shortest path between this pair, do the following:

Step 2.1: Find a new unused address a and let $A(a) = d$. (For some technologies, this amounts to generating a source routing header that contains all routing information; for others, such as InfiniBand, it amounts to assigning a new address to the destination.)

Step 2.2: Find an existing layer L_i such that letting R_i be enriched to support the path and letting $T(\langle s, a\rangle) = \{L_i\}$ will not close a cycle of dependencies in the layered dependency graph of I. If one exists, let $T(\langle s, a\rangle) = \{L_i\}$, otherwise, leave $T(\langle s, a\rangle)$ unchanged.

Step 3: If there are more pairs of source and destination that have not yet been processed, go to Step 2.

Step 4: (Optional balancing step) Find two existing layers L_j and L_k and a source/address pair $\langle s, a\rangle$ such that the following is true:

- there are more source/address pairs assigned to L_j than to L_k,
- $T(\langle s, a\rangle) = L_j$, and
- letting $T(\langle s, a\rangle) = L_k$ instead and updating R_j and R_k accordingly will not create a cycle of dependencies.

Let $T(\langle s, a\rangle) = L_k$ instead and update R_j and R_k accordingly. Repeat this step until no combination of L_j and L_k and a source/address pair $\langle s, a\rangle$ with the given properties exists.

Lemma 1. *If the above algorithm results in a traffic assignment function in which $T(\langle s, a\rangle) \neq$ undefined for all source/address pairs $\langle s, a\rangle$, the layered network I is free from deadlock with respect to R and T. Furthermore, all packets are routed along shortest paths.*

Proof. Shortest-path routing follows immediately from Step 2.

Assume that the lemma is not true. Then, the algorithm must in some case terminate with a deadlocked system. According to Theorem 2, this means that a cycle in the layered dependency graph must either have been there from the start of the algorithm or have been introduced at some point. From Step 1, we can deduce that the layered dependency graph contains no dependencies and, therefore, no cycles, when the algorithm starts and, hence, that the cycle must have been introduced later on.

The layered dependency graph is only altered by Step 2.2, which builds up the traffic assignment function, and Step 4, which alters it. The cycle in the layered dependency graph must, therefore, have appeared through the assignment or reassignment of a layer to a source/address pair. By inspection of Steps 2.2 and 4, we see that no such assignment will take place if it closes a cycle in the layered dependency graph. Thus, we have a contradiction, and the lemma follows. □

If the network contains fewer layers than are needed to provide shortest-path routing, the algorithm fails by leaving the traffic assignment function undefined for some source/address pairs. The method can, however, easily be adapted in such a way that these pairs are assigned to a layer, but are not routed according to their shortest paths. This can be done either by identifying nonshortest paths that fit into some layer, or by assigning all of these paths to a separate layer that is routed according to an UpDown structure.

The complexity of the algorithm is given by the number of source/destination pairs (N^2) times the number of layers (n) times the complexity of checking for cycles (N). Our tests show that this does not become a problem for modern machines until the networks are quite large (256 switches or more). Even for networks of bigger sizes, one can circumvent the problem by considering more than one source/address pair at a time. In [25], it is demonstrated that such an approach has only a minor effect on the number of virtual channels needed for shortest-path routing.

5.2 Evaluation of the Required Number of Layers

An important issue in the evaluation of layered routing is the number of layers that are needed to grant shortest-path routing to every source/destination pair. The required number of layers depends on both network size and connectivity. In this section, we treat the relation between these two network parameters and the need for virtual layers when using our method.

Some answers are easily derived. If we have minimal connectivity so that the network has the shape of a tree, one virtual layer suffices, because no cycle of dependencies can be closed as long as all routing follows a shortest path. Furthermore, networks with maximal connectivity will also need only one layer, because no packet will traverse more than one link and, so, no channel dependencies will exist.

However, most of the network topologies that are used in practice will fall somewhere in between these two

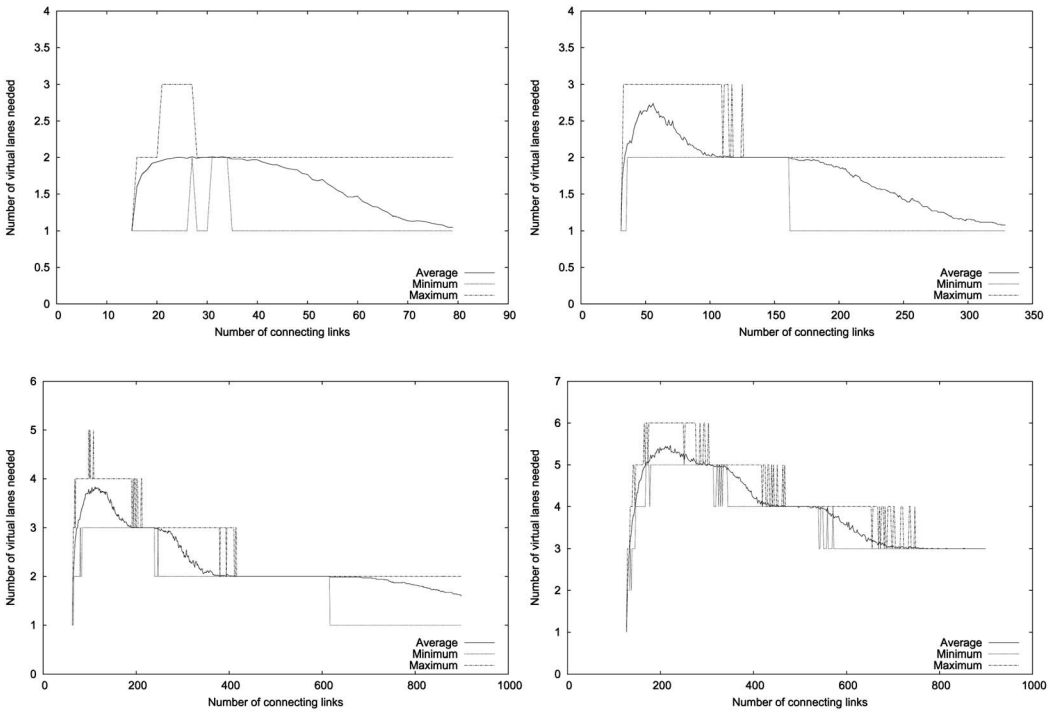

Fig. 3. Number of layers needed to allow shortest paths between all source/destination pairs in LASH on networks with a varying number of switches.

extremes, and in order to evaluate the relation between network connectivity and the need for virtual layers, we conducted a series of experiments. We considered four network sizes of 16 nodes, 32 nodes, 64 nodes, and 128 nodes, respectively. For each of these sizes, we considered a range of connectivities from minimal connectivity and upward toward maximal connectivity, adding one or two links at a time. For each network size and connectivity, we generated 100 random topologies and subjected them to a modified version of the algorithm above, which is always able to provide another virtual layer on a per need basis. Fig. 3 shows the average, the maximum, and the minimum number of virtual lanes needed in each case.

The most striking result from these plots is that the number of virtual layers required is very small. Even for networks as big as 128 switches, we never encountered a single topology that required more than six virtual layers. This is well below the number of layers that may be provided, e.g., in InfiniBand, and in the case of InfiniBand, it leaves more than half of the virtual layers to be used for purposes such as traffic separation and Quality of Service. For networks with 32 switches, even as few as three layers sufficed for all tested topologies. Another surprising outcome is that the variance in the required number of layers is very small. Even with 100 random topologies tested for each point of measurement, the difference between the most demanding and the least demanding topology was hardly ever more than one layer.

In order to obtain a more detailed picture of the relation between the number of switches and the required number of layers, we set up an experiment where we increased the number of nodes in the network, but kept connectivity fixed. It is apparent from our previous plots that, for any of the three considered network sizes, the maximum number of virtual layers is needed when the number of links in the topology is about twice the number of switches. Since this connectivity is a very likely choice for many application areas, we used this connectivity throughout this experiment. The results are plotted in Fig. 4.

Our previous results regarding variance and a very modest need for virtual layers are confirmed in this experiment, but what is new is that the number of required layers appears to follow a logarithmic curve.

6 Multiple Shortest Paths—Source Adaptivity (MP-LASH)

In most networks, there will be more than one shortest path between any source and destination. In the approach described above, we simply assumed a priori that only one such path was chosen. Some network technologies do, however, allow for several paths to be chosen between any pair of nodes. This choice can be made by the source node, either by inserting sufficient information into the header to determine the entire path (source routing) or by defining

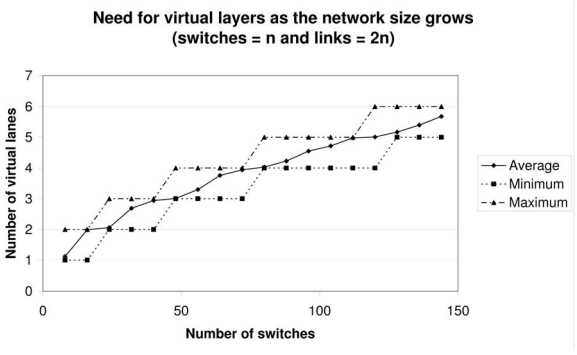

Fig. 4. The ratio between needed number of layers and network size.

multiple addresses for each destination. The routing tables can be set up such that each address follows different paths to the same destination.

This added freedom can also be exploited within the framework of layered routing. There are basically two dimensions of source adaptivity that may be exploited here:

- One can allow *all* shortest paths between any source and destination, assign a distinct destination address or appropriate source routing header to each of them, and let the source node decide which path to use for each packet it sends out.
- For each distinct shortest path in the network, there may be several layers into which this path can fit without creating a deadlock. Which layer to use can be chosen by the source node for each packet.

Let us now modify the algorithm we described above to cater for these added degrees of freedom:

Step 1: Let $T(\langle s, a\rangle) =$ undefined for all source/address pairs, let R_i be empty for all i such that $1 \leq i \leq n$, and let $A(a)$ be undefined for all addresses a.

Step 2: Take one pair of source and destination $\langle s, d\rangle$ that has not yet been processed. For all shortest paths between this pair do the following:

Step 2.1: Find a new unused address a and let $A(a) = d$. (For some technologies, this amounts to generating a source routing header that contains all routing information; for others, such as InfiniBand, it amounts to assigning a new address to the destination.)

Step 2.2: Find an existing layer L_i such that letting R_i be enriched to support the path and letting $T(\langle s, a\rangle) = \{L_i\}$ will not close a cycle of dependencies in the layered dependency graph of I. If one exists, let $T(\langle s, a\rangle) = \{L_i\}$, otherwise, leave $T(\langle s, a\rangle)$ unchanged.

Step 3: If there are more source/destination pairs that have not yet been processed, go to Step 2.

Step 4: (Optional) For all layers $L_i \in L$, go through all source/address pairs $\langle s, a\rangle$ and do the following:

Step 4.1: If letting R_i be enriched to support the path of $\langle s, a\rangle$ and letting the set of layers returned by $T(\langle s, a\rangle)$ be enriched with L_i does not close a cycle of dependencies in the layered dependency graph of I, perform these enrichments. Otherwise, leave R_i and T unchanged.

This algorithm has somewhat higher computational complexity than the one in Section 4.3 in that it considers all possible shortest paths. If we assume that p is the average number of shortest paths between any source/destination pair, n is the number of layers, and N is the number of nodes the complexity is $p \times n \times N^3$ when the optional steps are disregarded.

Apart from the computational complexity, the properties of this algorithm are similar to those of the previous one. If the original layered network has sufficiently many layers, it will succeed in finding a routing function R and a traffic assignment function T that supports all physical shortest paths in the network. The proof of this is also straightforward. All possible shortest paths are treated in Step 2 and freedom from deadlock is guaranteed in that no alterations to T and R are made without ensuring that no cycles are closed in the layered dependency graph (Step 2.2).

The cases where the algorithm fails may also be treated easily:

- If the network is only able to support some of the shortest paths for a source/destination pair, observe that all but one arbitrarily chosen of them can be discarded.
- If $T(\langle s, a\rangle) =$ undefined for some source/address pair $\langle s, a\rangle$ after the algorithm has terminated, one should solve this by using a nonshortest path.

Step 4 of the algorithm is not strictly necessary, as it does not improve its ability to support all possible shortest paths. It does, however, provide load balancing between the different layers and, for technologies that support choice between different paths at the source, it is a very natural extension.

The task of the source node during the operation is first to select, from the set of shortest paths available, the physical path of the packet (or packet stream) to be sent. This provides the destination address (or source routing header) to be used. Thereafter, the traffic assignment function T identifies the layer into which the packet (or stream) is to be injected. If Step 4.1 of the algorithm has been run, the source must, at this point, choose from a set of possible layers provided by T.

The number of layers needed to make all shortest paths available for different network sizes is plotted in Fig. 5. Again, we varied the network connectivity and, for each connectivity, generated 100 topologies. Here, also, the number of layers necessary to allow all shortest paths is quite modest. For example, for 64 node networks, we never saw the need for more than six virtual layers to accommodate all shortest paths. The variance is still low, and the maximum number of layers needed occurred when there were about twice as many links as nodes in the network.

7 Layered Routing with Switch Adaptivity (A-LASH)

Another aspect of choosing between different paths is that of adaptivity in the switches. Basically, we view the introduction of adaptivity as a method that provides all possible shortest paths in a network, but where the choice between different paths is made by the switches.

The problem in this case is that we cannot know in advance which path a packet will follow. Therefore, the algorithm for generating the routing strategy must, in this case, cater for a nondeterministic choice of path. This has the implication that we cannot treat one path at a time; we must consider all possible paths a packet may take and assign all of these paths to a layer at once.

However, it is possible to improve the above scheme. In order to find a finer granularity than considering all possible paths a packet may take in one go, consider the following: The dependencies that are introduced by the first switch after the source of a packet stream will never contribute to a cyclic dependency. The reason for this is that this dependency starts from an injection link. There can never be a dependency going *to* an injection link, since this is the first link that any packet will traverse. Therefore, we

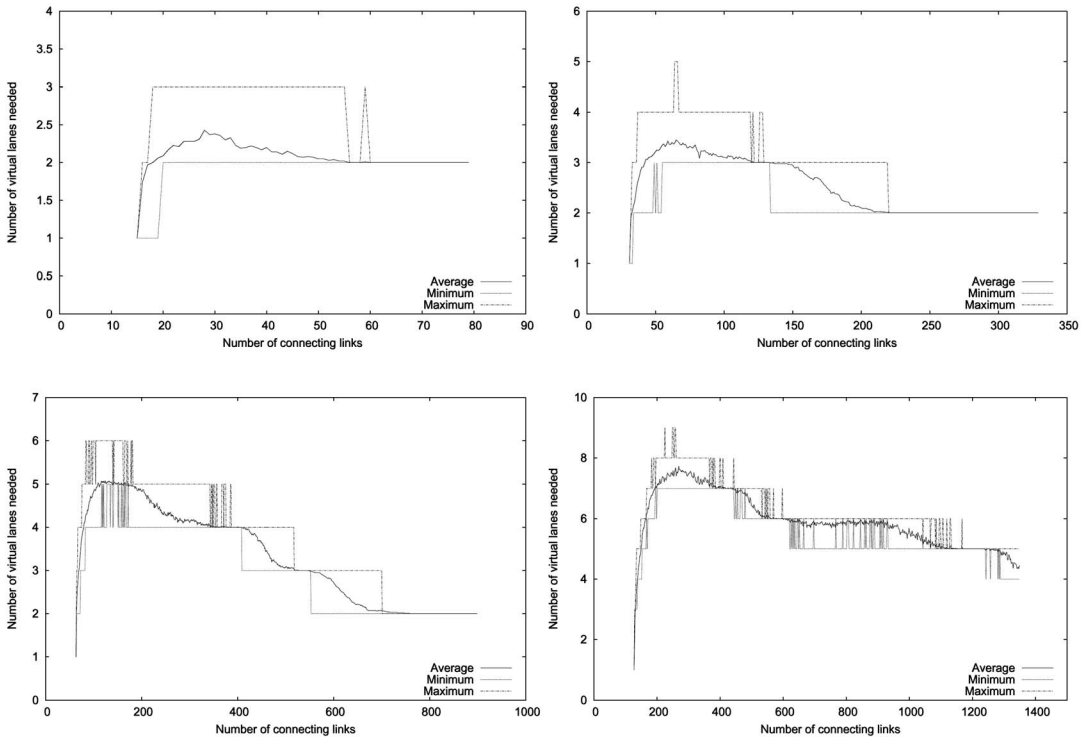

Fig. 5. Number of layers needed to allow all shortest paths between all source/destination pairs in MP-LASH on networks varying number of switches.

can group the shortest paths between every source and destination according to the link they acquire from the first switch. The above observation results in the following algorithm:

Step 1: Let $T(\langle s, a\rangle) =$ undefined for all source/address pairs $\langle s, a\rangle$, and let R_i be empty for all i such that $1 \leq i \leq n$.

Step 2: Take one source/destination pair $\langle s, d\rangle$ that has not yet been treated.

Step 2.1: For each link l, exiting the switch to which the source is connected and that is used by at least one shortest path between $\langle s, d\rangle$, do the following:

Step 2.1.1 Assign a new unique address to the destination. Let $\langle s, a\rangle$ denote the source/address pair associated with the new destination address.

Step 2.1.2: Associate to $\langle s, a\rangle$ all the shortest paths between the source/destination pair $\langle s, d\rangle$ and that uses l.

Step 2.1.3: Find an existing layer i such that letting R_i be enriched to support all paths associated to $\langle s, a\rangle$ and letting $T(\langle s, a\rangle) = \{L_i\}$ does not close a cycle of dependencies in the layered dependency graph of I. If one exists, let $T(\langle s, a\rangle) = \{L_i\}$ and enrich R_i, otherwise leave both unchanged.

Step 3: If there are more source/destination pairs that have not yet been treated, go to Step 2.

Step 4: (Optional) For all layers $L_i \in L$, go through all source/address pairs $\langle s, a\rangle$.

Step 4.1: If letting R_i be enriched to support the paths associated to $\langle s, a\rangle$ and letting the set of layers returned by $T(\langle s, a\rangle)$ be enriched with L_i does not close a cycle of dependencies in the layered dependency graph of I, perform these enrichments. Otherwise, leave R_i and T unchanged.

Again, the properties of this algorithm are similar to those of the previous ones. The complexity is the same, since every shortest path is considered exactly once in each layer (disregarding the optional steps). If the original layered network has sufficiently many layers, it will succeed in finding an adaptive routing function R and a traffic assignment function T that supports all physical shortest paths in the network. The proof of this is also straightforward.

When the adaptivity in routing choices is moved into the switches, it is natural to ask whether we can allow adaptivity between the layers as well. The answer is that it is indeed possible for cut-through and store and forward networks. However, it requires that we lift the restriction on layered routing that all packets remain in the layer into which they are injected.

Theorem 3. *In Cut Through and Store and Forward networks, the routing function R resulting from the above algorithm can be extended so that, for any $\langle s, a\rangle$, all packets associated with $\langle s, a\rangle$ can move freely between all layers in $T(\langle s, a\rangle)$ without creating deadlocks.*

Before we present the proof, we introduce some necessary definitions and a theorem from José Duato [7] on which the proof is built.

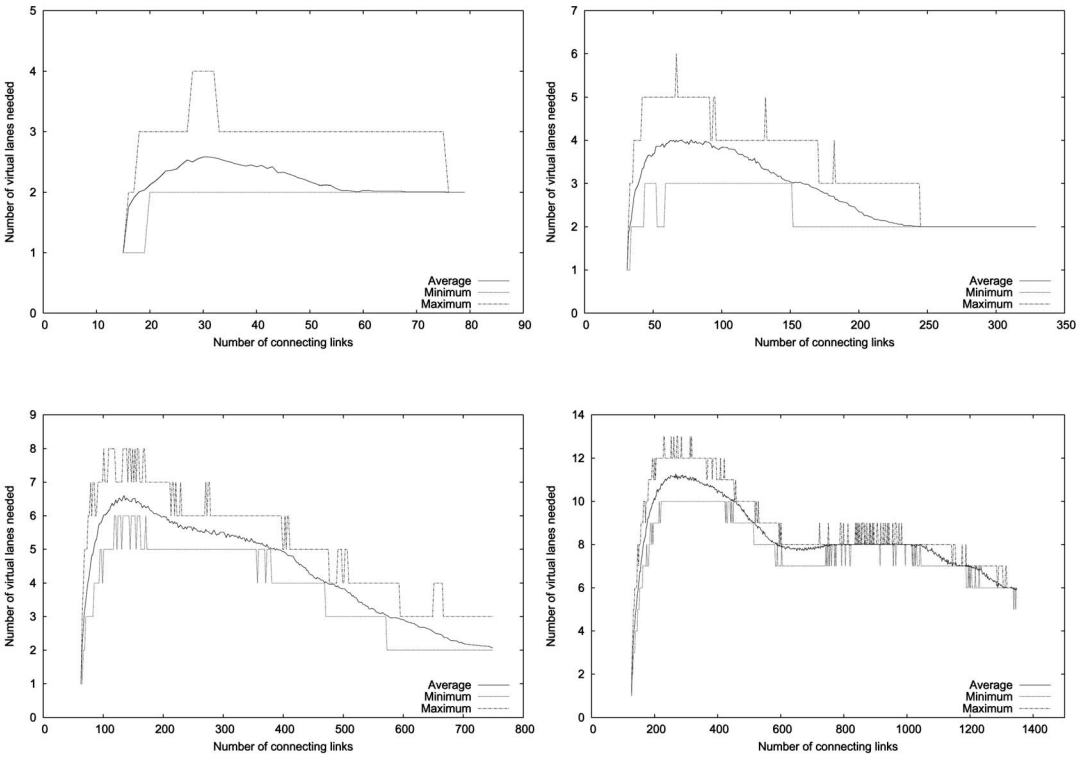

Fig. 6. Number of layers needed to allow all shortest paths between all source/destination pairs in A-LASH on networks with varying number of switches.

Definition 13. *The routing function R' is a* routing subfunction *of R if $R'(c_1, c_2, a) \subseteq R(c_1, c_2, a)$ for all c_1, c_2, and a.*

The above means that it is possible to find a routing subfunction from any routing function simply by removing some routing choices.

Definition 14. *Assume the routing function R and its routing subfunction R'. There is a* cross dependency *from channel c to channel c' if there are packets that*

1. *will not reach c according to R' only,*
2. *may reach c according to R, and*
3. *will use c' after c according to R'.*

Definition 15. *An* extended *channel dependency graph of a routing subfunction R' of R is the channel dependency graph of R' enriched with the cross dependencies of R' with respect to R.*

Theorem 4. *A connected routing function R for an interconnection network I is deadlock-free iff there exists a routing subfunction R' that is connected and has no cycles in its extended channel dependency graph.*

The proof of this theorem is lengthy, and it is therefore not repeated here. The proof of Theorem 3 is as follows:

Proof. Let R' be the routing function that results from the algorithm and R be the routing function that, in addition, lets all packets associated with $\langle s, a\rangle$ move freely between all layers in $T(\langle s, a\rangle)$. Clearly, R' is a routing subfunction of R. There are no cross dependencies between R' and R in the extended channel dependency graph, because R' does not restrict any buffer with respect to what packets can end up there. Furthermore, it is clear from the algorithm that the (layered) channel dependency graph of R' is free from cycles. Our theorem thus follows from Theorem 4. □

The effect on the number of layers needed in order to support all possible shortest paths is, of course, also of interest. We ran experiments equivalent to those reported for deterministic and source adaptive LASH, and the results are plotted in Fig. 6.

8 Evaluation of Performance Gains

We conducted a series of network simulations in order to evaluate the performance characteristics of layered routing in different variants. The experiments were divided into three categories based on the functionality of the underlying technology.

The first category covers routing methods that can be utilized by technologies that use deterministic routing. In this category, we have layered routing with multiple roots in which each pair of source and destination is assigned to only one layer (D-MROOTS), and layered shortest-path routing with only one path between each source and destination (LASH). For LASH, whenever the number of shortest paths to choose from was greater than 1, we chose one at random. These two methods have been compared with a deterministic version of UpDown routing using a depth-first spanning tree [19].

In the second category, we place methods that work on deterministically routed networks, but where the source node can choose adaptively between several different paths for each destination. The members of this category are

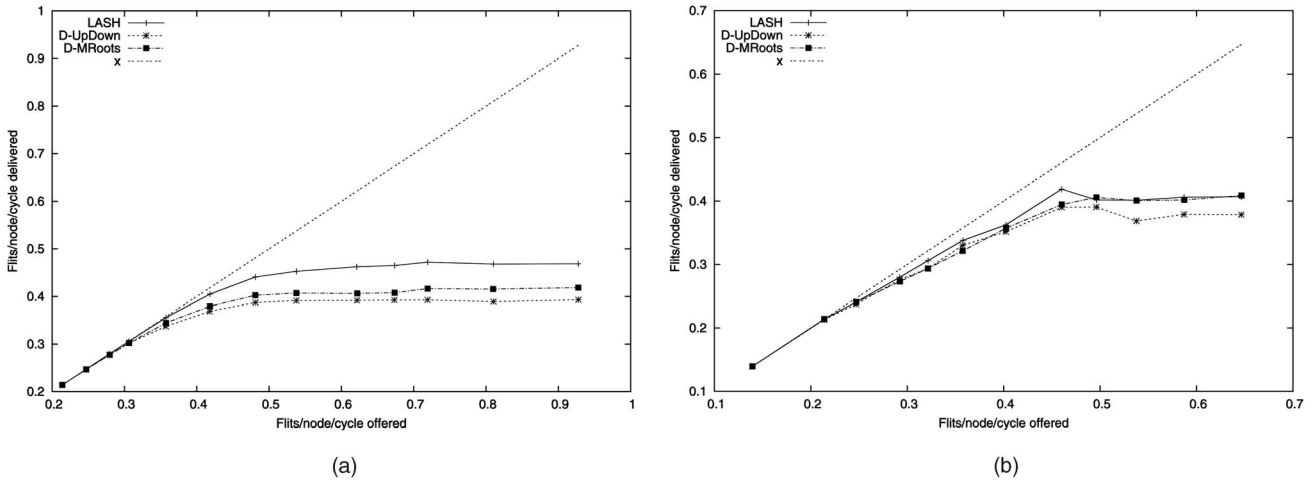

Fig. 7. Deterministic routing, 16 nodes. (a) Uniform and (b) pairwise.

layered routing with multiple shortest paths (MP-LASH) and layered routing with multiple roots, where each source can choose between all layers for all destinations (MROOTS). For reference, we include the results for deterministic LASH in the plots, so that the effect of source adaptivity can be evaluated.

In the third category, we place those routing methods that require adaptive routing facilities in the switches. In these plots, we put results for adaptive layered shortest path routing (A-LASH), and we compare it with adaptive UpDown routing, as well as adaptive routing with escape paths, as described in [23].

We considered network sizes of 16, 32, and 64 switches. For each network size, we generated 16 different topologies randomly, each of which had twice as many links as switches. Thereafter, one traffic node was attached to each switch, acting as a packet source and packet drain. All displayed results in Figs. 7, 8, 9, 10, 11, 12, 13, 14, and 15 represent averages over the 16 random topologies.

The speed of each link is one transfer unit, called *flit*, per "time unit," and the experiments were conducted with both a uniform address distribution and pairwise traffic. For pairwise traffic, a new set of pairs was drawn for each topology and load. Throughout the experiments, we used Virtual Cut Through flow control and a uniform packet size of 32 flits. Packet generation was governed by a normal approximation of the Poisson distribution. For each topology and load, we let the simulation run until the average latency had stabilized, before the measurements were started. Thereafter, the simulation was run for another 50,000 time units. In D-MROOTS, each layer has its own root and, consequently, its own routing function. The roots in the different layers of D-MROOTS were chosen so that the minimal distance between two roots was maximized in order to balance traffic [15].

8.1 Routing Methods for Deterministic Technologies

In order to compare the deterministic routing methods under equal conditions, we had to let LASH, D-MROOTS, and UpDown have the same number of virtual channels available per physical channel. Therefore, for each generated topology, we ran the algorithm in Section 5 in order to generate the LASH-routing algorithm with the minimal number of layers that guaranteed shortest-path routing. Thereafter, UpDown

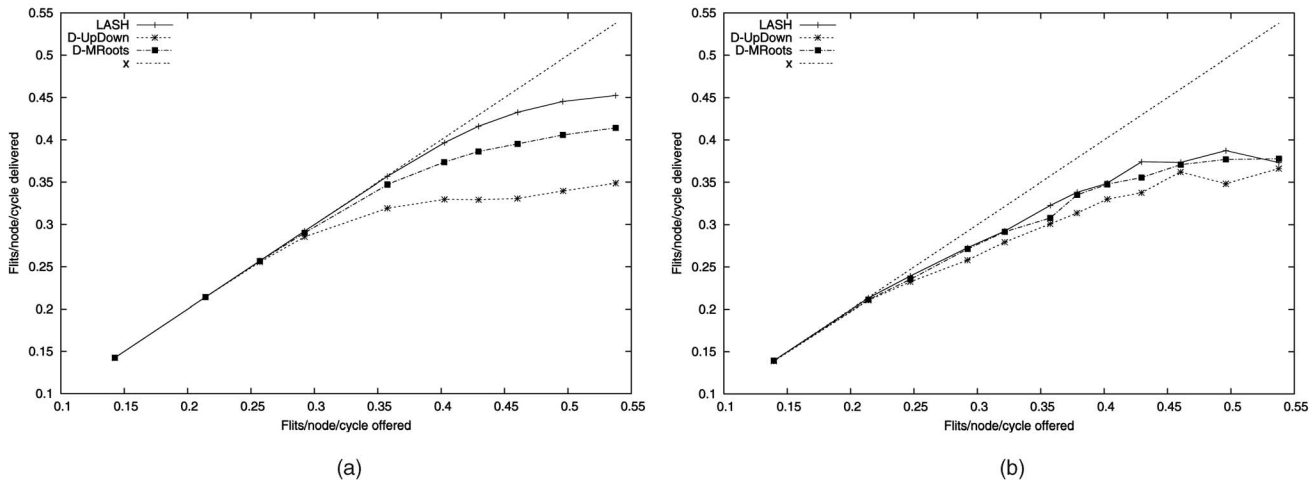

Fig. 8. Deterministic routing, 32 nodes. (a) Uniform and (b) pairwise.

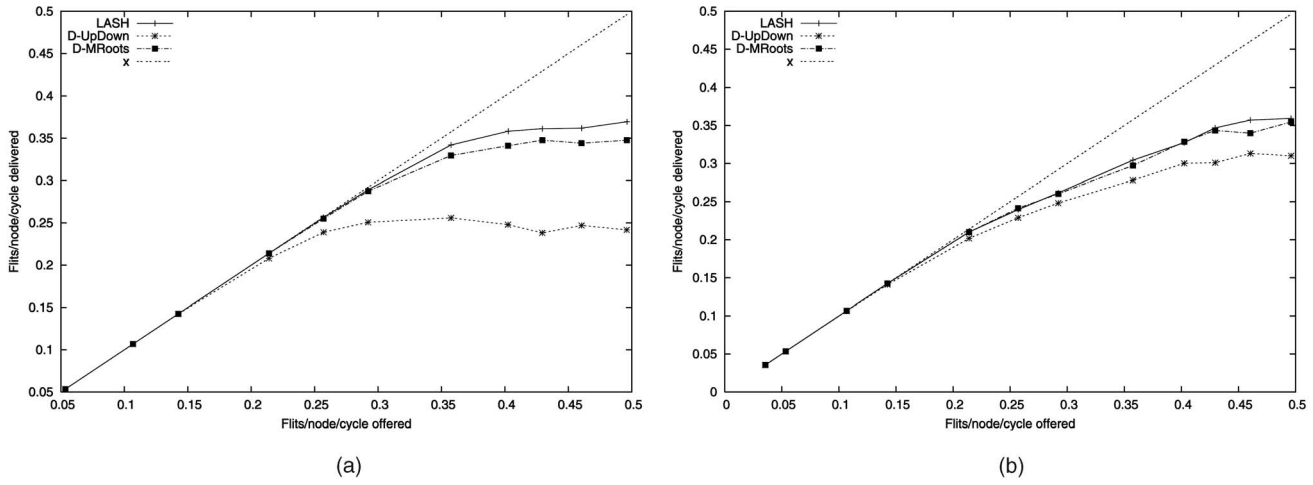

Fig. 9. Deterministic routing, 64 nodes. (a) Uniform and (b) pairwise.

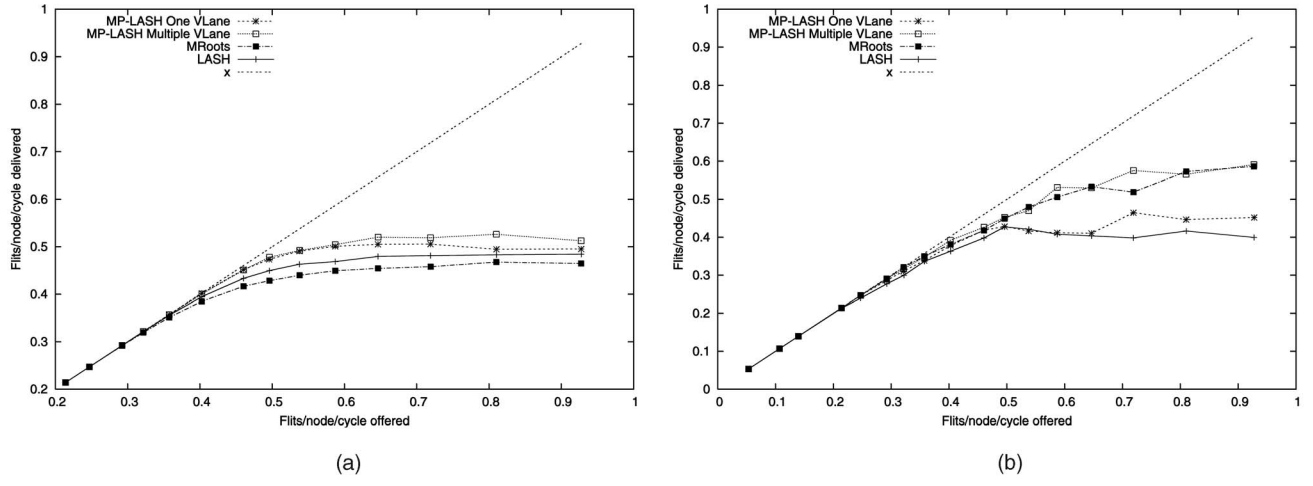

Fig. 10. Source adaptive routing, 16 nodes. (a) Uniform and (b) pairwise.

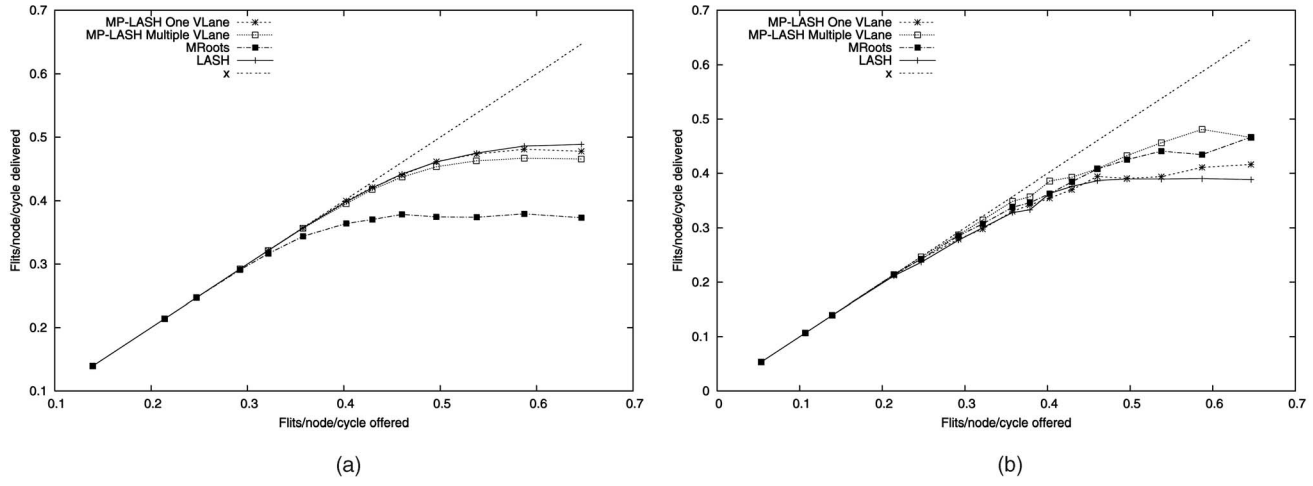

Fig. 11. Source adaptive routing, 32 nodes. (a) Uniform and (b) pairwise.

and D-MROOTS were allowed to use the same number of virtual lanes as LASH needed for that topology. The two latter routing methods were, however, only allowed to use one given layer for any source/destination pair. This guarantees that packets will be delivered in the correct order, which is a distinguishing factor of true deterministic routing. The choice of layer was made by distributing the source/destination pairs as evenly between the layers as possible. The results are

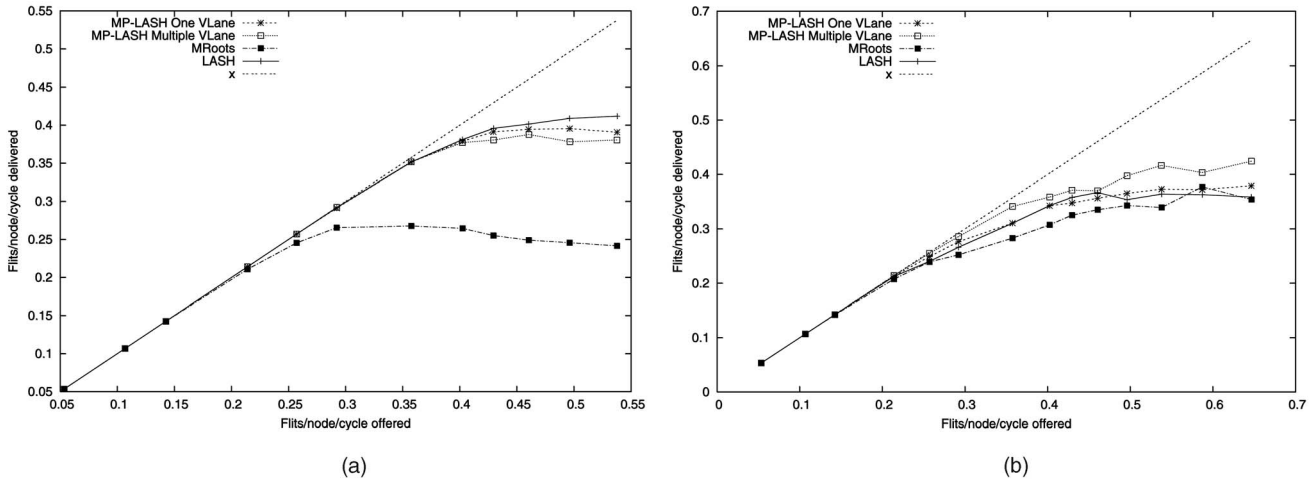

Fig. 12. Source adaptive routing, 64 nodes. (a) Uniform and (b) pairwise.

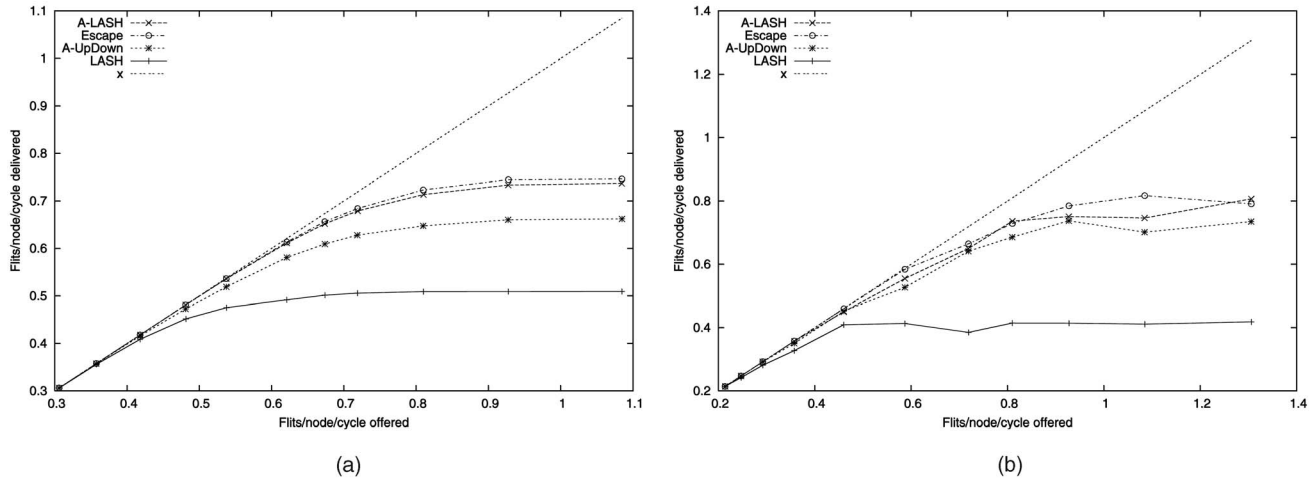

Fig. 13. Switch adaptive routing, 16 nodes. (a) Uniform and (b) pairwise.

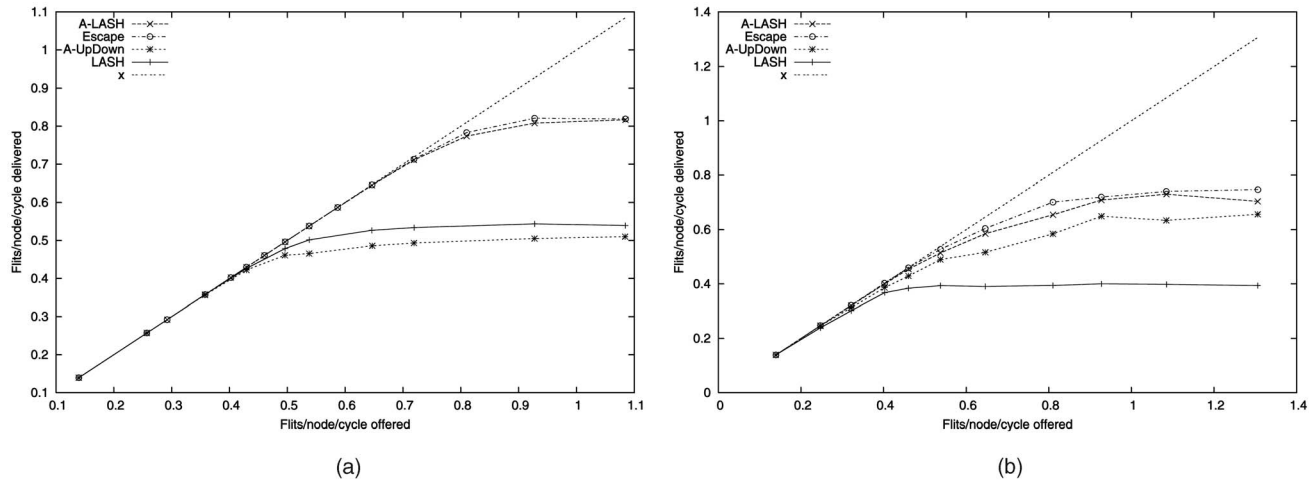

Fig. 14. Switch adaptive routing, 32 nodes. (a) Uniform and (b) pairwise.

displayed in Figs. 7, 8, and 9 for networks of sizes 16, 32, and 64 nodes, respectively.

The most apparent result here is that the layered routing techniques LASH and D-MROOTS perform consistently better than UpDown for uniform traffic. Furthermore, LASH performs consistently better than D-MROOTS, although the difference here is smaller. This is to be expected, because the hot-spot that UpDown creates around the root is, in D-MROOTS, replaced by a set of more lightly loaded roots. Furthermore, LASH should be expected to perform better than the two others since it has no concept of

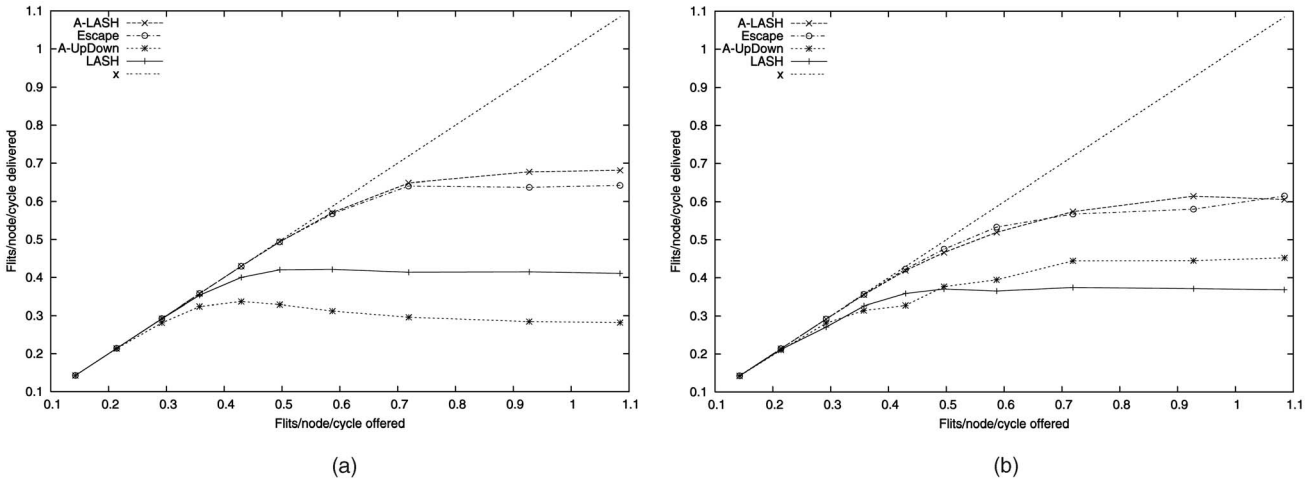

Fig. 15. Switch adaptive routing, 64 nodes. (a) Uniform and (b) pairwise.

roots and does not inherently generate hot-spots in the network.

The tendency for LASH and DMRoots to perform better than UpDown is there for pairwise traffic as well, although the differences between the different methods are far smaller there. In order to explain this, we must first observe that the important contribution of LASH and D-MROOTS is that they alleviate the tendency for congestion that will appear around the root of the UpDown tree. In a uniform traffic pattern, it is likely that the limiting factor of the throughput lies in this congested area. In pairwise traffic, however, each node sends packets to one other node exclusively. This means that, in generating random pairs, there may easily be other areas in the network that are as limiting performance-wise as the root is. This benefit of LASH and D-MROOTS will, therefore, be of less significance.

8.2 Source Adaptive Routing Methods

Source-adaptive routing methods allow the source to choose adaptively one of several paths to each destination. The layered routing functions that open for this are MROOTS and MP-LASH. In the experiments, we chose to test two versions of MP-LASH: one where the source could choose between different layers for each path (i.e., Step 4 of the algorithm in Section 6 was executed), and one where each path was assigned to exactly one layer. For MP-LASH, the source/address pairs were distributed among the layers as evenly as possible. Whenever a source can choose between several paths, this choice is made at random.

Again, we let the number of virtual channels available for each method be identical by letting the algorithm for MP-LASH (Section 6) first decide the number of layers that MP-LASH needed to guarantee shortest-path routing. Thereafter, MROOTS and LASH were allowed to use the same number of virtual layers as MP-LASH needed for that topology. The results are displayed in Figs. 10, 11, and 12 for networks of sizes 16, 32, and 64 nodes, respectively.

The plots show that, for uniform traffic, MROOTS performs consistently poorer than the other three methods, although the difference is not very significant for small networks. In fact, there is no improvement from D-MROOTS to MROOTS. This means that the extra load-balancing that source adaptivity provides does not improve MROOTS, as it is already fairly well-balanced. In fact, using all possible layers for all source destination pairs in MROOTS negatively affects throughput for large networks. This is because one congested layer will influence all source destination pairs. A less intuitive result is that the three other routing methods perform almost identically. In particular, one would expect that the source adaptivity exploited in MP-LASH should enable it to outperform LASH. However, it turns out that choosing the packet paths arbitrarily on the fly, as the two versions of MP-LASH do, does not yield any benefit in load balancing under uniform traffic, compared to choosing the paths arbitrarily in advance, as LASH does. If our versions of MP-LASH were able to choose packet paths based on knowledge of where there was congestion in the network, the picture might be different. We do, however, not see this as a viable option for source adaptive routing, so we choose to disregard this possibility. Another nonintuitive result is that MP-LASH gains nothing from being supplied with multiple virtual lanes under uniform traffic conditions. In fact, we see that this even results in a slight drop in throughput for large networks. There are two reasons for this result. First, the effect of one single congested layer is more severe with multiple virtual lanes, since more source-destination pairs will be affected. Second, when more source-destination pairs use one single layer, the layered dependency graph in this layer will contain more dependencies. Therefore, the congestion three generated by one hot-spot in one layer will spread to more links, which will result in poorer overall throughput.

For pairwise traffic, however, the picture changes. Here, MP-LASH performs consistently best, and LASH performs relatively poorly for small networks, although it improves as the networks grow larger. Furthermore, MROOTS performs relatively well compared to its performance for uniform traffic. The differences between the methods are, however, relatively small. In pairwise traffic where the pairs are generated randomly, there is a tendency for the network to be loaded very unevenly compared to uniform traffic. The ability of MROOTS to spread traffic in the network by not using shortest paths is, therefore, beneficial, because a root placed in an area through which few pairs communicate will lead to exploitation of more of the network resources. The reason that LASH performs relatively poorly here is that, for pairwise traffic, each link accommodates relatively few source destination pairs because there are fewer source/destination pairs that are active. Therefore, for LASH, a heavily loaded link has to accommodate all traffic generated

by a small number of sources. MP-LASH will even this out over more links, and uniform traffic improves the situation by giving the link smaller fractions of traffic from each source.

8.3 Switch Adaptive Routing Methods

The adaptive layered routing function that is of relevance for networks with real adaptivity is described in Section 7. We call this algorithm A-LASH. The most relevant routing methods of comparison are, of course, real adaptive routing with escape paths, as in [23], and adaptive UpDown routing. For fair comparison, we let the algorithm for A-LASH decide how many layers it is needed in order to guarantee shortest-path routing and, thereafter, we let UpDown and Escape routing benefit from the same number of layers. In particular, we let UpDown routing have adaptivity between both paths and layers. For Escape, we let all layers but one be open for fully adaptive routing. The last layer was used as an escape layer that was routed according to UpDown. The results are displayed in Figs. 13, 14, and 15, for networks of sizes 16, 32, and 64 nodes, respectively. For reference, we add the results for deterministic LASH.

The most striking result is that A-LASH and Escape perform almost identically for all topology sizes and both traffic patterns. The probable reason for this is that they are very similar. They both allow any packet to use any shortest path. The apparent drawback of Escape routing that the packets in the escape layers use nonshortest paths is balanced by the drawback of A-LASH that there are fewer virtual choices available for each packet. Further, adaptive UpDown routing scales poorly with network size, even compared to deterministic LASH. Whereas UpDown routing performs relatively well for networks with 16 nodes, it degrades to be poorer than deterministic LASH in 32-switch networks for uniform traffic.

Here, also, we see a tendency for the differences between the routing methods to diminish for pairwise traffic. Again, this is due to the fact that the many hot-spots generated by random pairs can, to a more limited extent, be ameliorated by routing strategies than can the fewer hot-spots generated by uniform traffic and shortest-path routing.

8.4 Limiting the Number of Layers

All the comparisons of routing strategies we have presented above are based on allowing the same number of virtual channels to each strategy. Furthermore, the number of virtual channels used in each comparison has been imposed by the strategy that requires the largest number of layers. This is a reasonable strategy for comparison, because new technologies tend to have far more virtual channels available than are ever needed by our methods (e.g., InfiniBand™ and AS have up to 16 virtual channels each). There will, however, be cases where the number of layers available are truly limited, either because the virtual channels are needed for something other than effective routing (e.g., Quality of Service), or because only a limited number of the specified virtual channels have been implemented.

Let us assume that, in the cases where the number of layers available for LASH is too small, we allow one of the available layers to be routed according to the UpDown scheme. Clearly, if there were only one layer available, LASH would degenerate to being identical to UpDown. By adding one layer at a time, the improvement toward the full value of LASH as demonstrated in the simulations would be stepwise. For networks of 32 nodes, the full value of LASH would have been reached already at two layers in many cases, and three layers in all cases, as can be seen in Fig. 3. For networks with 64 switches, the full value of LASH is realized with three layers in most cases and four layers in almost all cases.

A similar argument can be developed for MP-LASH. Assume that, when exposed to a limited number of layers, the algorithm first uses only one path per source/destination pair. Thereafter, it uses one layer for UpDown routing when there are not enough layers for single shortest paths. In that case, the method will first degenerate to ordinary LASH (and, thereby, improve if the network is large, according to our results), and then to UpDown.

For A-LASH we could make a similar analysis, but it would not be as relevant. The reason is that A-LASH's main competitor is Escape routing. While A-LASH would clearly lose much of its adaptivity when the number of layers are reduced, Escape routing remains fully adaptive down to two virtual layers. A-LASH will, therefore, be able to compete with Escape routing only under the assumption that a sufficient number of layers are present.

9 Conclusion

We have presented a method for generating routing functions for irregular network topologies. The method is based on using virtual channels to divide the physical network into different logical layers. Instead of restricting the paths that a packet can take through a network, we avoid deadlock by restricting the layers that each packet can use.

The method can be used to obtain load balancing as well as shortest-path routing. It can be applied to a wide range of technologies ranging from true deterministic ones, via source adaptive ones, to technologies where the switches can choose adaptively between multiple output ports for each packet. The required functionality in the switches is very limited. Extensive experiments have determined that the number of virtual channels that are needed for our method is very modest, even for quite large networks. None of the many thousands of topologies that we have tested have required more than six virtual channels for LASH, which is easily accommodated in most new and emerging network technologies. Our method can be used even if the technology has limited (as in e.g., InfiniBand) or even no (as in AS) means for letting packets migrate between layers.

Our simulation results show that the added routing efficiency of our method yields huge performance gains compared to standard UpDown routing in deterministic network technologies. For source adaptive technologies, our method yields even greater improvement for small and medium sized networks, in particular, for pairwise traffic. For larger networks, source adaptivity does not pay off in network performance compared to our method used with deterministic routing.

For switch adaptive technologies, however, there seems to be no gain in our method compared to fully adaptive routing with escape paths as it is described in [23]. The reason for this is that the routing flexibility provided by fully adaptive routing is comparable to the routing flexibility provided with our method. There are, however, many technologies that do not provide switch adaptivity, either because they require packets to be delivered in order, or because they could not afford the added complexity in the switch. The main contribution of our method is, therefore, that it provides significantly higher performance

than competing methods to technologies that do not have switch adaptivity. This comes with no other requirement on the technology than that it should support a limited number of virtual channels per physical channel.

REFERENCES

[1] K.V. Anjan and T.M. Pinkston, "An Efficient, Fully Adaptive Deadlock Recovery Scheme: DISHA," *Proc. Int'l Symp. Computer Architecture-95,* pp. 201-210, 1995.

[2] ASI-SIG, "Advanced Switching Core Architecture Specification," http://www.asi-sig.org/, 2004.

[3] N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J.N. Seizovic, and W.-K. Su, "Myrinet—A Gigabit-per-Second Local-Area Network," *IEEE Micro,* 1995.

[4] W.J. Dally, "Scalable Switching Fabrics for Internet Routers," Avici Systems Inc. White Paper, 2004.

[5] W.J. Dally and C.L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Trans. Computers,* vol. 36, no. 5, pp. 547-553, May 1987.

[6] J. Duato, "A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 6, no. 10, pp. 1055-1067, Oct. 1995.

[7] J. Duato, "A Necessary and Sufficient Condition for Deadlock-Free Routing in Cut-Through and Store-and-Forward Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 7, no. 8, pp. 841-854, Aug. 1996.

[8] J. Duato and T.M. Pinkston, "A General Theory for Deadlock-Free Adaptive Routing Using a Mixed Set of Resources," *IEEE Trans. Parallel and Distributed Systems,* vol. 12, no. 12, pp. 1219-1235, Dec. 2001.

[9] J. Flich, M.P. Malumbres, P. López, and J. Duato, "Performance Evaluation of a New Routing Strategy for Irregular Networks," *Proc. 2000 Int'l Conf. Supercomputing (ICS),* pp. 34-43, 2000.

[10] J. Flich, P. López, M.P. Malumbres, and J. Duato, "Boosting the Performance of Myrinet Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 13, no. 7, pp. 693-709, July 2002.

[11] R.W. Horst, "Tnet: A Reliable System Area Network," *IEEE Micro,* vol. 15, no. 1, pp. 37-45, 1995.

[12] InfiniBand℠ Architecture Specification, InfiniBand℠ Trade Assoc., 2005.

[13] D.H. Linder and J.C. Harden, "An Adaptive and Fault Tolerant Wormhole Routing Strategy for k-Ary n-Cubes," *IEEE Trans. Computers,* vol. 40, no. 1, pp. 2-12, Jan. 1991.

[14] Z. Liu and A.A. Chien, "Hierarchical Adaptive Routing: A Framework for Fully Adaptive and Deadlock-Free Wormhole Routing," *Proc. Sixth Symp. Parallel and Distributed Processing,* pp. 688-695, 1994.

[15] O. Lysne and T. Skeie, "Load Balancing of Irregular System Area Networks through Multiple Roots," *Proc. Second Int'l Conf. Comm. in Computing (CIC 2001),* 2001.

[16] M.D. Schroeder et al., "Autonet: A High-Speed, Self-Configuring Local Area Network Using Point-to-Point Links," SRC Research Report 59, Digital Equipment Corp., 1990.

[17] R. Perlman, "An Algorithm for Distributed Computation of a Spanning Tree in an Extended LAN," *Proc. Ninth Data Comm. Symp.,* pp. 44-53, 1985.

[18] W. Qiao and L.M. Ni, "Adaptive Routing in Irregular Networks Using Cut-Through Switches," *Proc. 1996 Int'l Conf. Parallel Processing (ICPP '96),* pp. 52-60, 1996.

[19] J.C. Sancho, A. Robles, and J. Duato, "An Effective Methodology to Improve the Performance of the Up*/Down* Routing Algorithm," *IEEE Trans. Parallel and Distributed Systems,* vol. 15, 2004.

[20] J.C. Sancho and A. Robles, "Improving Minimal Adaptive Routing in Networks with Irregular Topology," *Proc. 13th Int'l Conf. Parallel and Distributed Computing Systems (PDCS '00),* 2000.

[21] J.C. Sancho, A. Robles, J. Flich, P. Lopez, and J. Duato, "Effective Methodology for Deadlock-Free Minimal Routing in Infiniband Networks," *Proc. Int'l Conf. Parallel Processing,* pp. 409-418, Aug. 2002.

[22] F. Silla and J. Duato, "On the Use of Virtual Channels in Networks of Workstations with Irregular Topology," *Proc. 1997 Parallel Computing, Routing and Comm. Workshop,* 1997.

[23] F. Silla and J. Duato, "High-Performance Routing in Networks of Workstations with Irregular Topology," *IEEE Trans. Parallel and Distributed Systems,* vol. 11, no. 7, pp. 699-719, July 2000.

[24] F. Silla, J. Duato, A. Sivasubramaniam, and C.R. Das, "Virtual Channel Multiplexing in Networks of Workstations with Irregular Topology," *Proc. Fifth Int'l Conf. High Performance Computing,* pp. 147-154, 1998.

[25] T. Skeie, O. Lysne, and I. Theiss, "Layered Shortest Path (LASH) Routing in Irregular System Area Networks," *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS),* Apr. 2002.

Olav Lysne received the masters degree in 1988 and the Dr. Scient. degree in 1992, both from the University of Oslo. He is a research director at Simula Research Laboratory, and professor of computer science at Simula Research Laboratory and the University of Oslo. His early research contributions was in the field of algebraic specification and term rewriting, with a particular emphasis on automated deduction. While working in this field, he was a visiting researcher at Université de Paris-Sud. In later years, he has mainly been working on switching techniques such as wormhole switching and virtual cut through, focusing on problems like effective routing, fault tolerance, and quality of service. In this field, he has been a member of the program committees of several of the most renowned Conferences, like HPCA, ICPP, EuroPar, and HiPC. He has participated in a series of former and upcoming European projects (MACRAME, ARCHES, SCI-Europe, SIVSS), and published approximately 60 academic papers. He is a member of the IEEE.

Tor Skeie received the masters degree in 1993 in computer science from the University of Oslo and became a Dr. Scient. in 1998, also at the University of Oslo. He is an associate professor at Simula Research Laboratory and the Department of informatics, University of Oslo, Norway, as well as being affiliated with ABB Corporate Research. For years, he has been working in the interconnection networking field. In particular, he has focused on developing routing functionality, fault-tolerant methodologies, and predictable service. At ABB Corporate Research, he has conducted several international projects studying the applicability of Ethernet with regard to industrial use. The key research topics have been the road to deterministic Ethernet and to achieve highly precise time synchronization of intelligent electrical devices across switched Ethernet.

Sven-Arne Reinemo recieved the MS degree in computer science from the University of Oslo, Norway, in 2000. He is currently a PhD student at Simula Research Laboratory and the University of Oslo, where he is participating in the ICON project. The focus of his PhD work is on quality of service in interconnection networks.

Ingebjørg Theiss received the master's degree in 1999 and Dr. Scient. degree in 2004, both from the University of Oslo. She is currently a postdoctorate at Simula Research Laboratory, participating in the European project SIVSS. She has mainly been working on interconnection networks, focusing on problems like deadlock free effective routing and fault tolerance.