Adaptive-Trail Routing and Performance Evaluation in Irregular Networks Using Cut-Through Switches

Wenjian Qiao, *Member*, *IEEE Computer Society*, Lionel M. Ni, *Fellow*, *IEEE*, and Tomas Rokicki

Abstract—Cut-through switching promises low latency delivery and has been used in new generation switches, especially in high speed networks demanding low communication latency. The interconnection of cut-through switches provides an excellent network platform for high speed local area networks (LANs). For cost and performance reasons, irregular topologies should be supported in such a switch-based network. Switched irregular networks are truly incrementally scalable and have potential to be reconfigured to adapt to the dynamics of network traffic conditions. Due to the arbitrary topologies of networks, it is critical to develop an efficient deadlock-free routing algorithm. A novel deadlock-free adaptive routing algorithm called adaptive-trail routing is proposed to allow irregular interconnection of cut-through switches. The adaptive routing algorithm is based on two unidirectional adaptive trails constructed from two opposite unidirectional Eulerian trails. Some heuristics are suggested in terms of the selection of Eulerian trails, the avoidance of long routing paths, and the degree of adaptivity. Extensive simulation experiments are conducted to evaluate the performance of the proposed and two other routing algorithms under different topologies and traffic workloads.

Index Terms—Adaptive routing, cut-through switches, deadlock-free routing, irregular networks, incremental scalability, performance evaluation.

1 Introduction

Switch-based networks have received much attention in both local area networks (LANs) and wide area networks due to their higher network bandwidth and throughput, greater interconnect scalability and flexibility, and better fault handling capability than shared-medium networks. In switched networks, each host computer has a network adapter connecting to a network switch. When the scale of the network increases due to the increasing number of host computers and the increasing demand of aggregate network bandwidth, more switches can be added to the network. The interconnection of those switches defines various network topologies. For cost and performance reasons, switched networks usually assume arbitrary topologies. Although arbitrary topologies do provide the needed flexibility and incremental scalability [1], routing and flow control in such networks are not trivial and have a great impact on the network performance, especially for those emerging cut-through switches.

Several switching techniques have been used in switches to forward packets. It is known that the traditional store-and-forward switching, which buffers an incoming packet entirely before forwarding the packet to an outgoing channel, exhibits high latency. In order to reduce communication latency, the trend of new generation switches is to support cut-through switching. Many cut-through switches are commercially available, such as the DEC GIGAswitch [2] for FDDI networks, the Ancor FCS 266/1062 [3] switches for FCS (Fibre Channel Standards) networks, the HP EtherTwist LAN switch and the IBM 8271 EtherStream switch for switched Ethernet [4], and the Myricom Myrinet [5]. Unlike cell-based switches such as ATM switches, the above frame-based switches allow large packets and can increase the effective channel utilization. We concentrate on these frame-based switches which can support varying sizes of frames.

This paper considers high speed local area networks using cut-through switches, which provide a lower network latency and high incremental scalability. There are lots of challenging issues in such a network environment. A major difficulty in constructing a large-scale network with cut-through switches is to avoid deadlock among simultaneously transmitting frames. The concept of cut-through switching, also known as wormhole routing, has been used in new generation scalable parallel computers, such as the IBM SP [6], Cray T3D [7], MIT J-machine [8], Ncube-3 [9], and Intel Paragon [10]. In order to avoid deadlock, those scalable parallel computers have to use regular network topologies, such as meshes, tori and hypercubes. Many deadlock-free routing algorithms have been proposed for such regular network topologies. Interested readers may refer to [11] for a detailed survey. However, using regular network topologies is impractical and uneconomical for high speed switched networks as regular networks impose stringent interconnect constraints and are not incrementally

- *W. Qiao is with Transarc Corporation, Pittsburgh, PA 15219. E-mail: wenjian@transarc.com.*
- *L.M. Ni is with the Department of Computer Science and Engineering, Michigan State University, E. Lansing, MI 48824. E-mail: ni@cse.mus.edu.*
- *T. Rokicki is with the Hewlett-Packard Laboratories, Palo Alto, CA 94303. E-mail: rokicki@hpl.hp.com.*

Manuscript received 06 Nov. 1996; revised 22 Feb. 1999.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 100336.

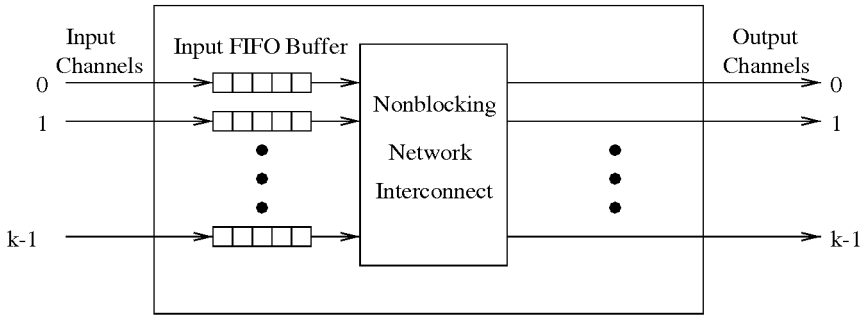


Fig. 1. A generic model of a cut-through switch.

scalable. Thus, a natural approach is to allow irregular switch interconnects like Myrinet [5] and ServerNet [12].

The main contributions of this study include a proposed routing algorithm and comprehensive performance evaluations among three routing algorithms in irregular networks. Due to the irregular network topologies, it is not trivial to develop an efficient routing algorithm with lower transmission latency and higher network throughput. We propose adaptive-trail routing (a novel deadlock-free adaptive routing algorithm) for irregular networks using cut-through switches. In order to avoid deadlock, we use two unidirectional Eulerian trails to help maintain an order of channel dependency and allow reasonable routing. Shortcuts are added to the Eulerian trails to provide more and shorter routing paths. Heuristics are suggested in order to provide better performance. A critical issue is how to increase network throughput and avoid deadlock at the same time.

In order to understand the influence of routing algorithms and network topologies to network performance, we investigate the performance of three existing routing algorithms proposed for cut-through switched networks with arbitrary topologies. The routing algorithms considered in this study are the up*/down* routing [13] used in the DEC AN1 system, the smart-routing [14] proposed for FCS switches, and our proposed routing algorithm [15]. Due to the dynamic nature of networking environments, analytic modeling is unlikely to give any practical insight to the network behavior and performance. Thus, our study is primarily based on intensive simulation experiments under different workloads. The simulation results show that different routing algorithms may gain performance benefits from different network topologies. However, while the up*/down* routing is heavily dependent on network topology, the smart-routing and the adaptive-trail routing have a relatively stable performance under different topologies.

The rest of the paper is organized as follows: In Section 2, we present the background knowledge and review the related routing algorithms. Our proposed routing algorithm is presented in Section 3. Section 4 presents the simulation environment and Section 5 demonstrates the influence of different routing algorithms to network performance under various topologies and traffic conditions. Section 6 concludes the paper and indicates future work for this study. Our algorithm is proved deadlock-free in the Appendix.

2 Background and Related Work

In this section, we first describe the network environment considered in this paper. Then we review the previous work of routing algorithms in cut-through (wormhole) switched networks. Two related routing algorithms for irregular switched networks are briefly presented in order to make a comparison.

2.1 Cut-Through Switched Networks

Fig. 1 shows a generic cut-through switch with k ports. For networks considered in this study, k is usually between 4 and 32. Each port is associated with a pair of input and output channels (or a bidirectional channel). Each port may connect to a node which generates and consumes messages, to a port of another switch which defines the network topology, or leave it open for a future connection. A *node* can be a workstation, a multiprocessor system, or a gateway to another network. A port connecting to a node is called a *terminal port*, a port connecting to another switch is called a *trunk port*, and a channel connecting two switches is called a *trunk channel*. There is no essential difference between a port used to connect to a node and a port used to connect to another switch. A nonblocking network (e.g., a crossbar) is used within a cut-through switch to allow simultaneous connections between different input and output channels [16].

In a cut-through switch, a packet can be transferred to an available output channel as soon as its header has been received and decoded. Most of the cut-through switches support input FIFO buffering for each incoming channel. The input buffer must be able to hold at least the header field of an incoming packet to make a routing decision. A message stays in the input buffer if there is no output channel available. When an input buffer is full due to the blocking of the selected output channel, the packet occupying the input buffer may hold some preceding input buffers in other switches on its path. Buffer capacity is one important parameter influencing the network performance. A larger buffer will reduce the probability of channel blocking, because fewer channels are occupied by a blocked message. Note that cut-through switching is different from virtual cut-through switching [17], where a large buffer will hold the entire packet when the outgoing channel is blocked.

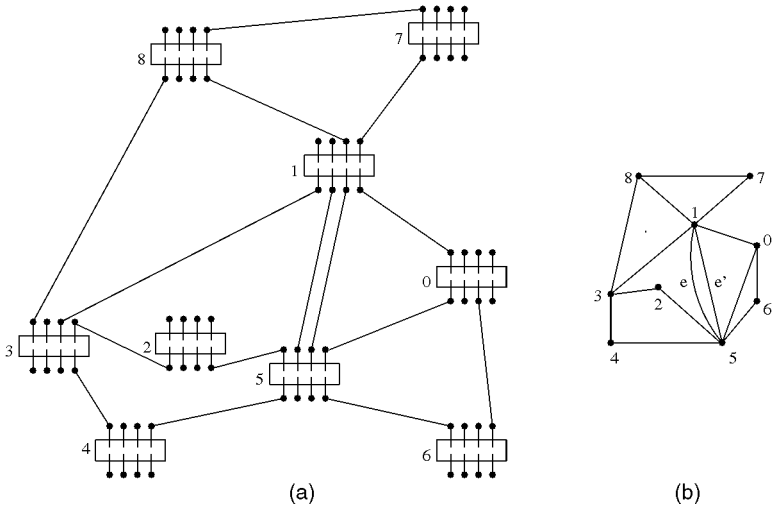


Fig. 2. An example of an arbitrary network topology.

Typically, most of the commercial switches use input FIFO buffers. The FIFO input buffering causes a limited throughput due to output port contention and head-of-line blocking. This problem can be alleviated by having output buffers, shared channel buffers, or arbitrary access input buffers. However, these solutions are quite complicated to implement in practice, and usually can only handle fixed size frames (packets) [18]. Since we consider variable size frames, we adopt input FIFO buffers and will show the impact of buffer capacity on performance.

As we mentioned earlier, regular network topologies do not have good incremental scalability due to the stringent interconnection constraints [1] and may not be able to use the original routing algorithms with faulty nodes. Thus, some arbitrary network topologies with good network performance are highly demanded. An arbitrary network topology is shown in Fig. 2a, where nine 8-port cut-through switches are interconnected. An open port is either connected to a node or open for future usage. The graph in Fig. 2b is used to model this network, where each vertex corresponds to a switch and each edge corresponds to a trunk channel. All channels in the network and edges in the corresponding graph are assumed to be bidirectional, and multiple edges between two vertices are allowed. This graph representation is used to represent network topologies and derive channel dependency graph in this paper.

2.2 Related Work

In a cut-through switch, based on the header information of incoming packets, a routing algorithm selects an outgoing channel to deliver or forward the packet. A critical issue in designing an appropriate routing algorithm in cut-through switched networks is to avoid channel deadlock. A deadlock occurs when there is a cyclic dependency among occupied channels and requesting channels. A well-known solution for deadlock avoidance was proposed in [19] based on restricting the routing algorithm. For example, the turn model proposed in [20] provides deadlock-freedom via prohibiting certain turns.

Some general methodologies [21], [22], [23], [24] have been proposed to allow deadlock-free routing in arbitrary networks. However, it is difficult to directly apply those general ideas to an irregular network. Besides these general methods, only two existing routing algorithms can be directly applied to irregular networks using cut-through switches: the up*/down* routing (**UDR**) used in the DEC AN1 system [13] and the smart-routing (**SR**) [14]. In order to avoid deadlock, the UDR and SR adopt the concept of deadlock-prevention, which never allows the formation of a dependency cycle. Our proposed adaptive-trail routing algorithm (ATR) uses the philosophy in [21], [24], which allows the existence of cycles, but does provide a channel to escape from the cycles. Neither UDR nor SR considers virtual channels. For simple and practical reasons, we do not consider virtual channels in our routing algorithm , but it is an interesting issue worth further study.

The up*/down* routing algorithm (UDR). As proposed in [13], a breadth-first spanning tree is constructed from a specified root. Each channel is assigned a direction based on the spanning tree, with "up" meaning "toward the root". A tie is solved by comparing the ids of two end switches of a channel. With this assignment, the directed channels do not form loops. A legal route is defined to be one that never uses a channel in the "up" direction after it has used one in the "down" direction. The UDR routing is easy to understand and implement, but it may concentrate traffic around the root switch and allow unnecessary long routing paths. Thus, its performance is greatly dependent on the network topology and the selected spanning tree. Since all routing paths satisfying the up*/down* requirement are allowed in the routing table, many very long routing paths may exist and cause poor performance in many cases. We have modified the original algorithm in the following way: if there are multiple routing choices in a routing table entry and the shortest distance among these paths is h, longer routing paths, which have more than two hops and are

longer or equal to $2h$, will be discarded. The modified UDR is called MUDR.

The smart-routing algorithm (SR). This algorithm was proposed in [14]. Based on the network topologies, the SR algorithm builds an explicit channel dependency graph and searches the graph for cycles. For each cycle, a dependency is broken to minimize a heuristic cost function. The procedure terminates when the channel dependency graph has no cycles. The routing is represented by the channel dependency graph. The SR can be used as adaptive routing (SRA) or deterministic routing (SRD). For the SRA an adaptive routing table, which allows multiple routing paths among the switch pairs, is created in each switch. For the SRD a deterministic routing table, which provides only a single path between any two switches, is created in each switch. In this paper, the SR represents both SRA and SRD. The smart-routing has done a good job to balance channel utilization under uniform traffic. However, the traffic balancing, which depends on a linear programming solver, is the most time consuming part when calculating the routing table. Such complexity may not gain significant performance benefit in real networks, since nonuniform (e.g., client/server) traffic is very commonly observed.

3 The Adaptive-Trail Routing Algorithm

Our proposed algorithm [15], [25] is called the adaptive-trail routing algorithm (ATR) and is applicable to any network topology with Eulerian trails. The basic idea of the ATR is to find two opposite unidirectional Eulerian trails to provide reasonable routing paths and control the order of channel dependency. The Eulerian trail is a sequence of channels, which visits each channel once and exactly once so that it can maintain the order of channel dependency. In order to maximize channel utilization and allow more and shorter routing paths, shortcuts are added to the two unidirectional Eulerian trails. The two unidirectional trails with shortcuts are called *adaptive trails*. To avoid deadlock, some shortcuts have to be removed or used in a restricted way based on the channel dependencies along the adaptive trails. However, a dependency cycle is allowed as long as there is an escape channel [21], [24] for that cycle. The allowed paths between pairs along the two adaptive trails define all legal routes. A static routing table is maintained in each switch to carry routing information.

3.1 Definitions

In order to simplify our presentation, we introduce the following definitions.

Definition 1. *Trail, path, and cycle: a trail is a finite sequence of edges (channels) of the form $v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_{m-1} \rightarrow v_m$, in which all the edges are distinct. If vertices $v_0, v_1, \cdots v_{m-1}, v_m$ are also distinct, it is called a path. When $v_0 = v_m$, it is called a cycle.*

A graph G is said to be *connected* if every pair of its vertices are joined by a path. Obviously, all graphs considered in this paper are connected. The *connectivity* of a vertex is defined as the number of edges connected to the vertex. For a pair of vertices u, v in G, $d_G(u, v)$ denotes the length of a shortest path from u to v in G. The *diameter* of a graph is the maximum $d_G(u, v)$.

Definition 2. *Eulerian trail and Eulerian graph: An Eulerian trail of a connected graph is a trail that contains all the edges of the graph. A graph is called an Eulerian graph if and only if it has an Eulerian trail. The sufficient and necessary condition is that all vertices have even degrees or exactly two vertices have odd degrees.*

Definition 3. *Channel, shortcut, and index in a unidirectional Eulerian trail: given a unidirectional Eulerian trail $ET = v_0 \rightarrow \cdots \rightarrow v_i \rightarrow \cdots \rightarrow v_j \rightarrow \cdots \rightarrow v_m$ in graph G, each vertex (switch) along the trail is given an index beginning from 0. In trail ET, a channel between any two subsequent vertices v_i and v_{i+1} is denoted as $c_{i,i+1}$ (i.e., $v_i \rightarrow v_{i+1}$). Given an Eulerian trail ET, a shortcut $s_{i,j}$ between two nonsubsequent vertices v_i and v_j exists if $j > i+1$, $v_j \neq v_{i+1}$, and the channel $v_i \rightarrow v_j$ is in the edge set of G. A shortcut $s_{i,i+1}$ exists between two subsequent vertices v_i and v_{i+1} if there are multiple channels between v_i and v_{i+1} in the network. Let C(ET) denote the set of all unidirectional physical links in ET, which means $v_i \rightarrow v_{i+1} \in C(ET)$, but $v_{i+1} \rightarrow v_i \notin C(ET)$.*

In Definition 3 and the rest of this paper, all trails, channels and shortcuts are unidirectional unless otherwise specified. The concepts of channels and shortcuts are used to logically distinguish physical links' positions in a trail. Any channel or shortcut in an Eulerian trail corresponds to a unidirectional physical link in the network. If v_i and v_j represent the same switch in a network, we say $v_i = v_j$. We use $cs_{i,j}$ to represent either $c_{i,j}$ or $s_{i,j}$ if applicable. If $v_i = v_p$, $v_j = v_q$ and there is no multiple channel between the two switches, $cs_{i,j}$ and $cs_{p,q}$ represent the same physical link.

Let us consider an Eulerian trail

$$8 \rightarrow 1 \rightarrow 0 \rightarrow 6 \rightarrow 5 \overset{e}{\rightarrow} 1 \rightarrow 3 \rightarrow 2 \rightarrow 5 \overset{e'}{\rightarrow} 1 \rightarrow 7 \rightarrow 8 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 0$$

for the network in Fig. 2. For simplicity, only two shortcuts are shown here using the arrowed lines over the trail:

$$\overrightarrow{8 \rightarrow 1 \rightarrow 0 \rightarrow 6 \rightarrow 5 \overset{e}{\rightarrow} 1 \rightarrow 3} \rightarrow 2 \rightarrow 5 \overset{e'}{\rightarrow} \overrightarrow{1 \rightarrow 7 \rightarrow 8} \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 0.$$

In this example, shortcut $s_{0,6}$ (i.e., $\overrightarrow{8 \rightarrow \cdots \rightarrow 3}$) and channel $c_{11,12}$ (i.e., $8 \rightarrow 3$) represent the same physical link. Formally, a shortcut $s_{p,q} \in C$(ET) if and only if there is a k such that $v_k = v_p$ and $v_{k+1} = v_q$.

3.2 Eulerian Trail

Our ATR routing algorithm can be applied to any network which has an Eulerian trail. The algorithm to find an Eulerian trail is well-known. We adopt the same algorithm in the book [26]. A network may have more than one Eulerian trail. Our routing algorithm is based on any Eulerian trail of the network.

Since bidirectional channels are used to interconnect switches, a bidirectional Eulerian trail is considered as two opposite unidirectional Eulerian trails, called ET1 and ET2, respectively. As illustrated in Fig. 4, one Eulerian trail, ET1, for Fig. 2b is

$$0 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 8 \rightarrow 7 \rightarrow 1 \xrightarrow{e'} 5 \rightarrow 2 \rightarrow 3 \rightarrow 1 \xrightarrow{e} 5 \rightarrow 6 \rightarrow 0 \rightarrow 1 \rightarrow 8.$$

We use e and e' to distinguish the two physical links between switches 1 and 5. Another Eulerian trail ET2 can be obtained if we look at the above trail in the reverse order:

$$8 \rightarrow 1 \rightarrow 0 \rightarrow 6 \rightarrow 5 \xrightarrow{e} 1 \rightarrow 3 \rightarrow 2 \rightarrow 5 \xrightarrow{e'} 1 \rightarrow 7 \rightarrow 8 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 0.$$

If a channel is in ET1, it cannot be a channel in ET2.

Obviously, deadlock is impossible if messages are routed along either ET1 or ET2. However, some messages may not be able to use the shortest paths to reach their destinations. Therefore, shortcuts are added to each trail. The trails with shortcuts are called *adaptive trails*. The adaptive trails derived from ET1 and ET2 are called AT1 and AT2, respectively.

3.3 Adaptive Trail

Any channel in the network may be a shortcut in an adaptive trail. However, deadlock is possible if shortcuts are added without any restriction. A deadlock example is shown in Fig. 3, where a shortcut $s_{0,6}$ (i.e., $\overrightarrow{8 \rightarrow \cdots \rightarrow 3}$) is added to ET2. As shown in the figure, deadlock happens when $s_{0,6}$ is occupied by $m1$ and $m3$ is waiting for $c_{11,12}$ (i.e., $8 \rightarrow 3$).

In order to avoid deadlock, shortcuts have to be used with some restrictions. The concept of *channel dependency* was first presented in [19]. A routing algorithm is deadlock-free if there is no cyclic channel dependency in the network [19]. However, such a condition is too restrictive for adaptive routing, where more than one outgoing channel is offered at many switches. It has been shown in both [24] and [21] that an adaptive routing algorithm with cyclic channel dependency can still be deadlock-free as long as some necessary conditions are satisfied. For any detected dependency cycle, we break it if it may cause deadlock, but we allow it if it cannot cause deadlock.

The shortcuts are categorized into three different types: *free-style shortcut*, *destination shortcut*, and *source shortcut*. Fig. 4 shows how to add shortcuts step by step. For a partial trail like $\cdots u_1 \rightarrow \cdots \rightarrow u_j \cdots \rightarrow v_1 \cdots \rightarrow v_k \cdots$, where

$$u_i = u(1 \le i \le j)\ ,\ v_i = v(1 \le i \le k),$$

and no v exists between u_1 and u_j, any allowed shortcut $u \xrightarrow{shortcut} v$ is drawn from u_j to v_1. The order to add different types of shortcuts facilitates deadlock avoidance.

Free-style shortcut (f-shortcut). Given an Eulerian trail ET, a shortcut $s_{p,q}$ is a free-style shortcut if there is an $i < p$ such that $v_i = v_p$ and $v_{i+1} = v_q$. A free-style shortcut must be in C(ET). This type of shortcut can be illustrated by the arrowed line over $u \rightarrow \cdots \rightarrow v$ in the following trail: $\cdots u \rightarrow v \rightarrow a \rightarrow \cdots \rightarrow \overrightarrow{u \rightarrow \cdots \rightarrow v} \rightarrow \cdots$.

To create adaptive trails, free-style shortcuts are first added to each Eulerian trail as illustrated in the first step of Fig. 4. Note that in ET1, the free-style shortcut $s_{10,11}$ (from switch 1 to switch 5) is channel e', while channel $c_{10,11}$ is e. A free-style shortcut can be used by any message if it is on the routing path.

Destination shortcut (d-shortcut). Given an Eulerian trail ET, any shortcut of ET is a candidate destination shortcut. In the second step of Fig. 4, there are many destination shortcuts shown as solid lines below the Eulerian trails. A destination shortcut $s_{p,q}$ can only be used by messages whose destination is v_q. For example, destination shortcut $s_{3,8}$ (from switch 3 from switch 2) in AT1 can only be used by messages whose destination is switch 2.

Source shortcut (s-shortcut). Given an Eulerian trail ET, any shortcut of ET but not in C(ET) is a source shortcut. In the third step of Fig. 4, there are many source shortcuts shown as solid lines above the Eulerian trails. A source shortcut $s_{p,q}$ can only be used by messages whose source is a local host connected to switch v_p. No message can hold a channel and request for a source shortcut. For example, source shortcut $s_{2,7}$ (from switch 4 to switch 5) in AT1 can only be used for messages generated by a host connected to switch 4. Note that channel $s_{6,15}$ (i.e., from switch 1 to switch 8) in AT1 is not a source shortcut, since $1 \rightarrow 8$ is in C(ET1).

Given the above definitions, it is noted that a shortcut $s_{p,q}$ may have more than one shortcut type. For example, $s_{3,8}$ (from switch 3 to switch 2) in AT1 is both a source shortcut and a destination shortcut. In this case, $s_{i,j}$ is used for both purposes.

After adding different shortcuts, we have to detect cyclic channel dependency. Any routing path allowed in the corresponding Eulerian trail will be kept. Whenever a

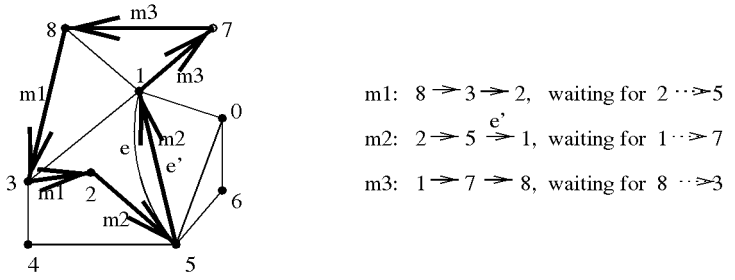


Fig. 3. An example of deadlock.

dependency cycle has to be broken to avoid deadlock, we always remove a shortcut to break the cycle.

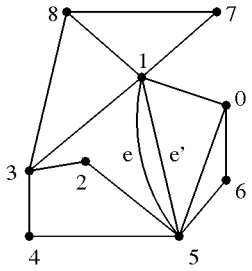
As we mentioned above, different types of shortcuts are used for different purposes. A destination shortcut cannot cause deadlock. Because whenever it is occupied as a destination shortcut, it always delivers a message to the destination and will be released finally. In Fig. 4, there is a dependency

$$3(v_3) \rightarrow 8(v_4) \overset{d-shortcut}{\longrightarrow} 1(v_6)$$

on AT1 and a dependency

$$8(v_0) \rightarrow 1(v_1) \rightarrow 0(v_2) \rightarrow 6(v_3) \rightarrow 5(v_4) \rightarrow 1(v_5) \rightarrow 3(v_6) \overset{d-shortcut}{\longrightarrow} 8(v_{11})$$

on AT2. The two dependencies cause a dependency cycle, but the cycle cannot cause deadlock due to the following:



- The two unidirectional Eulerian trails

Index: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

ET1: 0 → 5 → 4 → 3 → 8 → 7 → 1 →(e') 5 → 2 → 3 → 1 →(e) 5 → 6 → 0 → 1 → 8

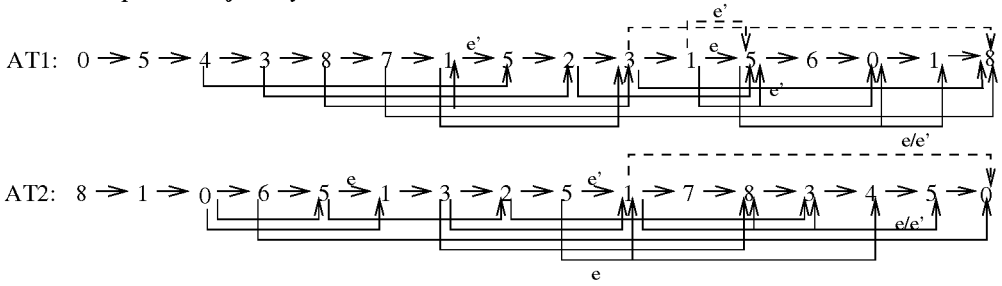
ET2: 8 → 1 → 0 → 6 → 5 →(e) 1 → 3 → 2 → 5 →(e') 1 → 7 → 8 → 3 → 4 → 5 → 0

- First step: add free-style shortcuts

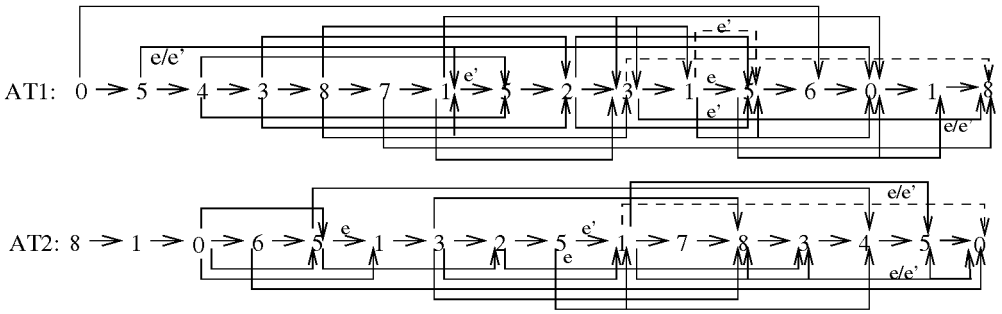
AT1: 0 → 5 → 4 → 3 → 8 → 7 → 1 →(e') 5 → 2 → 3 →(e') 1 →(e) 5 → 6 → 0 → 1 → 8

AT2: 8 → 1 → 0 → 6 → 5 →(e) 1 → 3 → 2 → 5 →(e') 1 → 7 → 8 → 3 → 4 → 5 → 0

- Second step: remove free-style shortcuts that could cause deadlock and add destination shortcuts



- Third step: add source shortcuts and remove some of them causing deadlock:



1. Free-style shortcuts are drawn in dashed line above the Eulerian trails.

2. Destination shortcuts are drawn in solid line below the Eulerian trails.

3. Source shortcuts are drawn in solid line above the Eulerian trails.

Fig. 4. An example of adaptive trails.

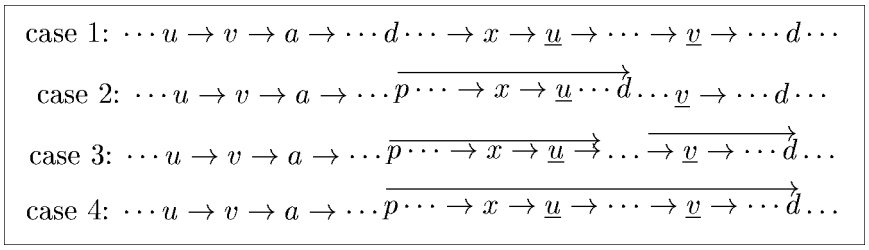


Fig. 5. All configurations $\underline{u} \overset{f-shortcut}{\longrightarrow} \underline{v}$ is removed to avoid deadlock.

If $8(v_4) \to 1(v_6)$ is used as a destination shortcut on AT1, it will be released finally. If $8(v_0) \to 1(v_1)$ is used on AT2, there is another path from 8 to 1 (i.e., $8(v_4) \to 7(v_5) \to 1(v_6)$) available in AT1.

Dependency cycles due to free-style shortcuts may cause deadlock. Consider the following trail, where u and $\underline{u}$ represent a same node and v and $\underline{v}$ represent a same node.

$$\cdots u \to v \to a \to \cdots d \cdots \to x \to \overrightarrow{\underline{u} \to \cdots \to \underline{v}} \to \cdots d \cdots$$

Assume there is a free-style shortcut $\underline{u} \overset{f-shortcut}{\longrightarrow} \underline{v}$. Because $u \to v$ and $\underline{u} \overset{f-shortcut}{\longrightarrow} \underline{v}$ represent the same physical link, there is no way to distinguish them when a message arrives at switch v from u. Given the routing path $u \to v \to a \to \cdots \to d$ in the Eulerian trail, a message from source x to destination d may take path

$$x \to \underline{u} \overset{f-shortcut}{\longrightarrow} \underline{v} \to a \to \cdots \to d.$$

Such a path breaks the channel dependency order given by the Eulerian trail and creates a dependency cycle together with the dependency of $a \to \cdots \to d \to \cdots \to x \to \underline{u}$. It may cause deadlock, so $\underline{u} \overset{f-shortcut}{\longrightarrow} \underline{v}$ has to be removed.

The problem here is that two paths exist to route a message from u to d: one using the channel $u \to v$ and the other using the f-shortcut $\underline{u} \overset{f-shortcut}{\longrightarrow} \underline{v}$. When a message uses $\underline{u} \overset{f-shortcut}{\longrightarrow} \underline{v}$, switch v cannot tell if it is from u or $\underline{u}$. This results in route $x \to \underline{u} \overset{f-shortcut}{\longrightarrow} \underline{v} \to a \to \cdots \to d$, which breaks the channel dependency order along the unidirectional trail. In Fig. 5, we show all the configurations where there are two paths from u to d. Therefore, f-shortcut $\underline{u} \overset{f-shortcut}{\longrightarrow} \underline{v}$ in these cases have to be removed to avoid deadlock.

Deadlock may be caused by a dependency cycle due to source shortcuts, too. In Fig. 6, there is a dependency $a \overset{s-shortcut}{\longrightarrow} b \to \cdots \to c \to d$ on AT1 and a dependency $c \overset{s-shortcut}{\longrightarrow} d \to \cdots \to a \to b$ on AT2. The two dependencies create a cycle and may cause deadlock. Therefore, a source shortcut (e.g., $c \to d$ on AT2) has to be removed to break the cycle. Without loss of generality, we assume AT2 is used to break such a cycle.

Fig. 5 and Fig. 6 have shown all the possible deadlock configurations caused by adding shortcuts. The deadlock detection algorithm will detect them and break a deadlock cycle by removing a shortcut. The detailed proof given in the Appendix shows that the adaptive-trail routing is deadlock-free.

3.4 Routing Tables

After getting the adaptive trails, a static routing table will be constructed in each switch. A routing table has many entries and each entry is indexed by both the destination switch and the incoming channel (port). Some entry may have multiple options of outgoing channels, which are selected by shortest path first policy. We first create routing table for each trail separately; then combine them together and remove redundant entries if any. Adjustment of the routing table may be done according to the heuristics in Section 3.5.

Our scheme scans each trail beginning from the largest index to the smallest index. At the beginning, the routing table for each switch is empty. When a switch is visited, the corresponding entries will be added to the routing table based on its position in the adaptive trail and the shortcuts that could be used. If a switch occurs more than once in the adaptive trail, new entries will be added to the routing table when the algorithm visits it again. Note that the multiple occurrences for a same switch have different indexes along the trail. When a switch is first visited, the pseudocode of the algorithm is shown in Fig. 7.

In Table 1, a routing table is created in switch 5 when it (i.e., v_{11}) is first visited along AT1 of Fig. 4. Note that $5(v_{11}) \overset{d-shortcut}{\longrightarrow} 0(v_{13})$ and $5(v_{11}) \overset{d-shortcut}{\longrightarrow} 1(v_{14})$ can only be used for destination switches 0 and 1, respectively.

Since our algorithm scans the trail from largest index to the smallest index, the algorithm will visit a switch again if the switch has multiple occurrences in the trail. When a

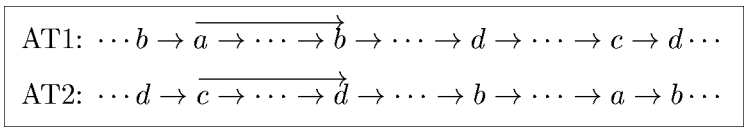


Fig. 6. The deadlock case due to source shortcut.

Input: routing tables of other switches
Output: routing table for Switch v_k
Algorithm:
for each out-channel $cs_{k,q} = v_k \rightarrow v_q$ in the trail
  for each in-channel $cs_{p,k} = v_p \rightarrow v_k$ in the trail, which is not a d-shortcut
    if $cs_{k,q}$ is a s-shortcut
      α = local host port IDs;
    else
      α = local host port IDs and $cs_{p,k}$;
    endif
    add an entry to Switch v_k, where
    destination=v_q, in-channel=α, out-channel=$cs_{k,q}$, distance=1;
    if $cs_{k,q}$ is not a d-shortcut
      for each entry E in routing table of Switch v_q
        if $E's$ incoming channel == $cs_{k,q}$
          add an entry to Switch v_k, where
          destination=$E's$ destination,
          in-channel=α, out-channel=$cs_{k,q}$,
          distance=$E's$ shortest distance + 1;
        endif
      endfor
    endif
  endfor
endfor

Fig. 7. Create routing table when v_k is first visited.

switch is visited again, a key point is to use the output channels available for its current occurrence and the routing entries in the existing routing table. However, we should avoid routing a message to any switch more than once. Fig. 8 shows the pseudocode to create routing table entries when a switch is multiply visited.

In AT1 of Fig. 4, switch 5 has three occurrences: v_1, v_7, and v_{11}. Since our scheme scans the trail from the highest index to the lowest index, v_{11} (switch 5) is visited first and the routing table entries are shown in Table 1. Then, v_7 (also switch 5) is visited and new entries are added by applying Algorithm 8. These new entries for switch 5 are shown in Table 2. Note that for destination switch 6, we do not allow the path

$$5(v_7) \rightarrow 2(v_8) \rightarrow 3(v_9) \rightarrow 1(v_{10}) \stackrel{e/e'}{\rightarrow} 5(v_{11}) \rightarrow 6(v_{12}),$$

because it passes switch 5 twice.

A shortcut offers a shorter routing path, but it may not be available for message routing. Consider that a message M comes from $5(v_7) \rightarrow 2(v_8) \rightarrow 3(v_9) \rightarrow 1(v_{10})$ (on AT1) and goes to switch 0. Although $1(v_{10}) \stackrel{d-shortcut}{\rightarrow} 0(v_{13})$ is the shortest path to M's destination, M may take

$$1(v_{10}) \stackrel{e}{\rightarrow} 5(v_{11}) \rightarrow 6(v_{12}) \rightarrow 0(v_{13})$$

if $1(v_{10}) \stackrel{d-shortcut}{\rightarrow} 0(v_{13})$ is not available when M is routed from switch 1. This results in passing switch 5 twice and wastes channel utilization. Due to this reason, a heuristic in Section 3.5 is suggested to handle such a case.

TABLE 1
Routing Table Entries When switch 5 Is Visited As v_{11}

Dst	Incoming Channels	OutCh 1	OutCh 2
0	$1 \stackrel{e/e'}{\rightarrow} 5, 2 \rightarrow 5$, terminal ports	$5 \rightarrow 0$	$5 \rightarrow 6$
1	$2 \rightarrow 5$, terminal ports	$5 \stackrel{e/e'}{\rightarrow} 1$	$5 \rightarrow 6$
6	$1 \stackrel{e/e'}{\rightarrow} 5, 2 \rightarrow 5$, terminal ports	$5 \rightarrow 6$	none
8	$1 \stackrel{e/e'}{\rightarrow} 5, 2 \rightarrow 5$, terminal ports	$5 \rightarrow 6$	none

Input: routing tables of all switches
Output: updated routing table for Switch v_k
Algorithm:

```
/* First, copy and modify all the entries available in Switch v_k */
for each in-channel c_{p,k} = v_p → v_k in the trail, which is not a d-shortcut
  for each entry E in routing table of Switch v_k
    copy E to a new entry newE in Switch v_k;
    modify newE so that its incoming channel is c_{p,k};
  endfor
endfor

/* Second, create entries unique for v_k */
for each out-channel c_{k,q} = v_k → v_q in the trail
  for each in-channel c_{p,k} = v_p → v_k in the trail, which is not d-shortcut
    if c_{k,q} is a s-shortcut
      α = local host port IDs;
    else
      α = local host port IDs and c_{p,k};
    endif
    add an entry to Switch v_k, where
    destination=v_q, incoming channel=α, outgoing channel=c_{k,q}, distance=1;
    if c_{k,q} is not a d-shortcut
      for each entry E in the routing table of Switch v_q
        if ( (E's incoming channel == c_{k,q}) and
          (the path to E's destination does not have to pass Switch v_k) )
          add an entry to Switch v_k, where
          destination=E's destination,
          in-channel=α, out-channel=c_{k,q},
          distance=E's shortest distance + 1;
        endif
      endfor
    endif
  endfor
endfor
```

Fig. 8. Create the routing table when v_k is visited again.

3.5 Heuristics about Degree of Adaptivity

If a switch appears more than once along an Eulerian trail, there will be more than one outgoing channels from the switch to some destinations. Multiple routing paths from this switch to a destination switch offers a higher degree of adaptivity and redundant routing choices for fault-tolerance. However, if a path is very long, many messages may be blocked due to the long path. Simulation results in [15], [25] show that allowing long paths will cause low network throughput and high message latency. Our heuristics consider multiple routing options in a switch and decide whether to keep a longer routing path or not. Two heuristics are suggested: *dual-path heuristic* and *source heuristic*.

Dual-path heuristic. Suppose switch v appears more than once along the trail. Let v_i and v_j $(j > i)$ denote any two of them. If v_i and v_j have different outgoing channels for a destination d, the dual-path heuristic will decide if the longer path is allowed.

One case is that d appears once:

$$\cdots \rightarrow v_i \rightarrow a \rightarrow \cdots \rightarrow v_j \rightarrow b \rightarrow \cdots \rightarrow d \rightarrow \cdots.$$

In this case, v_i can go to d from either $v_j \rightarrow b$ or $v_i \rightarrow a$ plus some shortcut(s) if any. Suppose the later choice is used. The shortcut may not be actually used in a route if it is not available when a message is scheduled, which results in a longer path and visiting switch v twice. An example is path

$$5(v_7) \rightarrow 2(v_8) \rightarrow 3(v_9) \rightarrow 1(v_{10}) \xrightarrow{e} 5(v_{11}) \rightarrow 6(v_{12}) \rightarrow 0(v_{13})$$

on AT1. Our dual-path heuristic will not keep such a path unless its length is no longer than the path from v_j.

TABLE 2
New Routing Table Entries for switch 5 When It Is Visited As v_7

Dst	Incoming Channels	OutCh 1	OutCh 2	OutCh 3
0	$4 \rightarrow 5$, terminal ports	$5 \rightarrow 0$	$5 \rightarrow 6$	$5 \rightarrow 2$
0	$1 \stackrel{e'}{\rightarrow} 5$	$5 \rightarrow 0$	$5 \rightarrow 6$	none
1	$4 \rightarrow 5$, terminal ports	$5 \stackrel{e/e'}{\rightarrow} 1$	$5 \rightarrow 2$	$5 \rightarrow 6$
2	$1 \stackrel{e'}{\rightarrow} 5, 4 \rightarrow 5$, terminal ports	$5 \rightarrow 2$	none	
3	$1 \stackrel{e'}{\rightarrow} 5, 4 \rightarrow 5$, terminal ports	$5 \rightarrow 2$	none	
6	$1 \stackrel{e'}{\rightarrow} 5, 4 \rightarrow 5$, terminal ports	$5 \rightarrow 6$	none	
8	$1 \stackrel{e'}{\rightarrow} 5, 4 \rightarrow 5$, terminal ports	$5 \rightarrow 2$	$5 \rightarrow 6$	none

Another case is that d appears twice:

$$\cdots \rightarrow v_i \rightarrow a \rightarrow \cdots d \cdots \rightarrow v_j \rightarrow b \rightarrow \cdots \rightarrow d \rightarrow \cdots.$$

When a message comes from $v_i's$ incoming channel, it may use either $v_i \rightarrow a$ or $v_j \rightarrow b$. If one path is much longer than the other, the longer path may not be good for performance because it consumes many channel resources. If the length of the shorter path is h, our dual-path heuristic will discard the longer path if its length is greater than $2h$.

Source heuristic. There may be more than one routing path between a source and a destination. If the shortest distance among these options is h, our heuristic will not keep an option in the corresponding entry if its length is greater than $2h$. Such a heuristic is called the *source heuristic*, which is only applied to terminal incoming ports. In Fig. 4, since the shortest path from switch 0 to 1 is $0 \rightarrow 1$ (on AT1), the longer path $0 \rightarrow 6 \rightarrow 5 \rightarrow 1$ (on AT2) will not be used by messages coming from a terminal port in switch 0.

3.6 Effect of Eulerian Trails

So far, we use the well-known algorithm to randomly choose an Eulerian trail. Different Eulerian trails may result in performance difference, which has been observed in our simulations [15], [25]. In order to get a better performance, a heuristic to find an appropriate Eulerian trail is needed.

Given two bidirectional Eulerian trails, the following method is used to choose one of them, which may provide a better performance. First, the adaptive trails corresponding to each bidirectional Eulerian trail are constructed. Let $r(u, v)$ be the shortest distance from u to v allowed by the adaptive trails. Let

$$\frac{\delta(u, v) = r(u, v) - d_G(u, v)}{d_G(u, v)},$$

where $d_G(u, v)$ is the shortest distance from u to v allowed by the network. Let Δ be the sum of all $\delta(u, v)$. In general, adaptive trails with smaller Δ will have a better performance. However, it is not guaranteed that the adaptive trails with minimum Δ will provide the best performance. Also, it is still an open issue about how to construct an Eulerian trail with the minimum Δ.

In our simulations, we select several bidirectional Eulerian trails by varying the starting switch and the way to select the next channel (see [25]). For each bidirectional Eulerian trail, we create the corresponding adaptive trails and compute Δ. An Eulerian trail with the smallest Δ is selected.

3.7 Time Complexity

The whole algorithm is simple to implement. Let m be the number of trunk channels and n be the number of switches in a network. As shown in [25], the time complexity of finding an Eulerian trail, adding all shortcuts, removing free-style shortcuts for deadlock avoidance, removing source shortcuts for deadlock avoidance, and creating routing tables are $O(m)$, $O(m^2)$, $O(m^2)$, $O(m^2)$, and $O(mn)$, respectively. Thus, the total time complexity is $O(m^2)$.

4 Simulation Environment and Performance Metrics

A simulator has been implemented to demonstrate the performance of UDR, SR, and ATR routing algorithms under various topologies and traffic workloads. In order to evaluate network performance, a simulator should consider the following workloads: traffic patterns, message size distributions, and temporal distribution [27]. There are many network parameters (e.g., topologies, placement of nodes, destination distributions, message size, buffer size, message injection rate, location of servers, etc.) and combinations of these parameters. However, due to the space limitation, we can only show the performance under a limited subset of parameter combinations.

4.1 Network Workload and Parameters

Two destination distributions (i.e., traffic patterns) are considered:

1. *Uniform traffic:* a node uniformly communicates with any other node. Two nodes connected to the same switch may communicate with each other.
2. *Client/server traffic:* k nodes are servers and all the other nodes are clients.

For a client, a certain rate (75 percent in this paper) of traffic is sent uniformly among those servers and the rest of

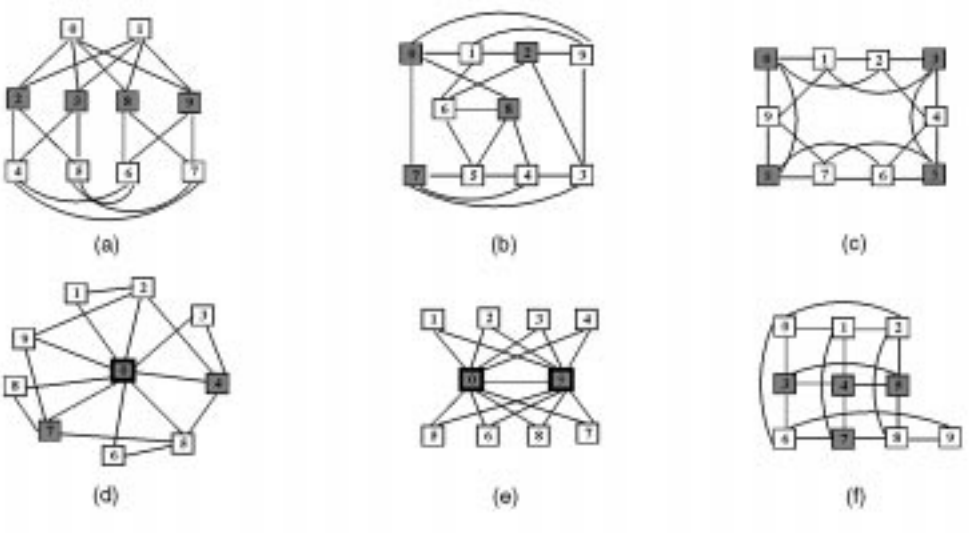


Fig. 9. Six topologies considered in simulation (shadowed switches will host servers in client/server traffic).

traffic is sent uniformly among those clients. A server sends messages to the other clients and servers with uniform distribution.

In a real network, a client may not uniformly communicate with all servers; instead, it may communicate with a specific server more frequently. However, there are too many cases in such a nonuniform client/server pattern. For simplicity, we choose the above model. In terms of k, we use four or five servers out of 75 nodes in our simulations.

We use two message size distributions:

1. Fixed size messages: the message size is always 1,000 bytes.
2. Bursty messages: bursty traffic happens at a specific rate and bursty size is 10,000 bytes.

Other messages are 100 bytes long. The bursty rate is five percent in this paper.

The fixed message size and the uniform traffic distribution are combined to demonstrate the maximum sustained throughput or worst case latency. A client/server distribution is more realistic than uniform distribution and useful to model network hot spots. In actual network applications, the bursty message distribution and the hot-spot destination pattern are commonly observed.

A time unit is defined as the time needed to transmit a byte via a channel, which is decided by channel bandwidth. We assume the header for each message is 10 bytes. Thus, the switch delay for a routing decision is 10 times the unit (i.e., the input buffer should have at least 10 bytes to hold the complete header field). The input FIFO buffer capacity is 100 bytes by default. However, we also measure the performance of 1,000 bytes buffer capacity to show the effect of the buffer size.

In a cut-through switch, it is not necessary to limit the size of a packet smaller than the input buffer capacity. We assume variable size packets and consider each message as a packet. Because the ATR uses escape channels to avoid deadlock, a buffer cannot be assigned to a new packet unless it is empty. However, it is not necessary for the SR and UDR. Thus, the SR and UDR may take more benefits of large buffer capacity, since a single buffer can hold more than one packet. But this benefit may not be significant if the buffer capacity is much smaller than the message size.

We use finite input source to model the message generation in each node, because it is more realistic than the infinite source model typically used by other researchers. Moreover, due to the limitation of memory space, the oversaturated behavior is extremely difficult to measure using the infinite source model. A finite source model may have up to K outstanding messages. In high performance computing, most parallel programs use blocking send to deliver messages. Thus, the next send will not be initiated until the previous one is completed. In a client/server environment, a server may receive several requests and have several messages waiting on the queue to send. For simplicity, we consider the case of $K = 1$, which means next message cannot be generated unless the previous message has completely left the source.

Let x be the time duration between the time when the current message has completely left the source buffer and the time when the next message is generated in each node. Thus, x is a random variable based on an exponential distribution. The smaller x is, the heavier the workload is.

4.2 Network Topologies

The topologies used to measure network performance are shown in Fig. 9, where shadowed switches will host servers in client/server traffic. In order to make a rather fair comparison, all topologies have the same number of switches and nodes. There are 10 switches and 75 nodes in each network. The number of trunk channels, the node distributions, diameters, and average path length are shown in Table 3. We assume that all switches have the same number of bidirectional ports and the number of used ports in each switch is similar. Open ports are left for future usage. Based on the node distribution, each switch should have at least 12 ports.

TABLE 3
The Network Configurations for Simulation

Switch	Number of Nodes					
	A	B	C	D	E	F
0	7	7	7	3	3	8
1	8	8	8	8	9	7
2	8	7	8	8	9	8
3	8	8	7	8	8	8
4	7	8	8	8	9	8
5	8	8	7	8	9	7
6	7	7	7	8	9	7
7	7	7	8	8	8	7
8	8	8	8	8	8	8
9	7	7	7	8	3	7
total nodes	75	75	75	75	75	75
total trunk channels	20	20	20	18	17	19
diameter	2	3	3	2	2	3
avg. path length	1.399	1.442	1.50	1.510	1.631	1.454

The *path length* from node u to v is defined as the number of trunk channels traversed by the path. For any two nodes in a same switch, the path length between them is defined as 0. The *average path length* is the average length of the shortest paths between nodes over all pairs of nodes which is defined as

$$\frac{\sum_{i,j \in nodes} shortest\_path\_length(i,j)}{number\_of\_node\_pairs}.$$

Note that the average length of paths allowed by routing may be longer than the average path length in the network due to the deadlock avoidance and adaptive routing. A good routing algorithm should keep the average length of routing paths as small as possible.

In this paper, we do not intend to evaluate the effect of network topology on performance. We choose several topologies in order to demonstrate the performance trend for our routing algorithms. Some differences among the six topologies are briefly summarized in the following: A, B, and C have the same number of trunk channels for each switch, but they have different diameters and the average path lengths. The number of trunk channels are unevenly distributed among switches in D and E. Channels in F are basically evenly distributed among switches except that switch 9 has only two trunk channels, which may cause bottleneck for traffic coming from or going to switch 9.

The performance of UDR and ATR depends on the selection of the root switch and the selection of Eulerian trails. For the ATR, we adopt all the heuristics in Sections 3.5 and 3.6 and select one bidirectional Eulerian trail with the smallest Δ (see Section 3.6) among three randomly selected bidirectional Eulerian trails. For the UDR, we select the switch with the maximum number of trunk channels as the root. If there is a tie, the root is selected randomly from the switches with maximum number of channels. To assign the direction for a channel whose end switches are at the same level, we assume that the "up" direction is always from the switch with a smaller id to the switch with a larger id. The MUDR selects the same root as the UDR in the same topology.

4.3 Performance Metrics

From an application program's point of view, the most important performance metric of a network is *communication latency*, which is the time interval between the instant that a send command, either explicit or implicit, is issued until the instant that the message is completely received by the recipient. The metric communication latency consists of network latency (including the blocking time due to unavailable output channels) and software latency. Since our purpose is to compare the network latency, we do not consider software latency in this study.

From the network point of view, an important metric is *network throughput*, which is defined as the total amount of message data transmitted in the network per unit of time. The maximum network throughput is obtained without the consideration of network contention (due to limited channel bandwidth, number of channels, network topology, and routing) and output contention (due to destination distributions). In our simulations, the maximum network throughput is 75 bytes/time_unit, because there are 75 nodes in each network. However, due to the limitation of network contention and output contention, the maximum *sustained network throughput* is usually much smaller than the maximum network throughput. The sustained network throughput and average message transmission latency are measured as follows:

- $sustained\_throughput = \frac{total\_transmitted\_bytes}{total\_time}$
- $latency = last\_byte\_arrival\_time - message\_generation\_time$

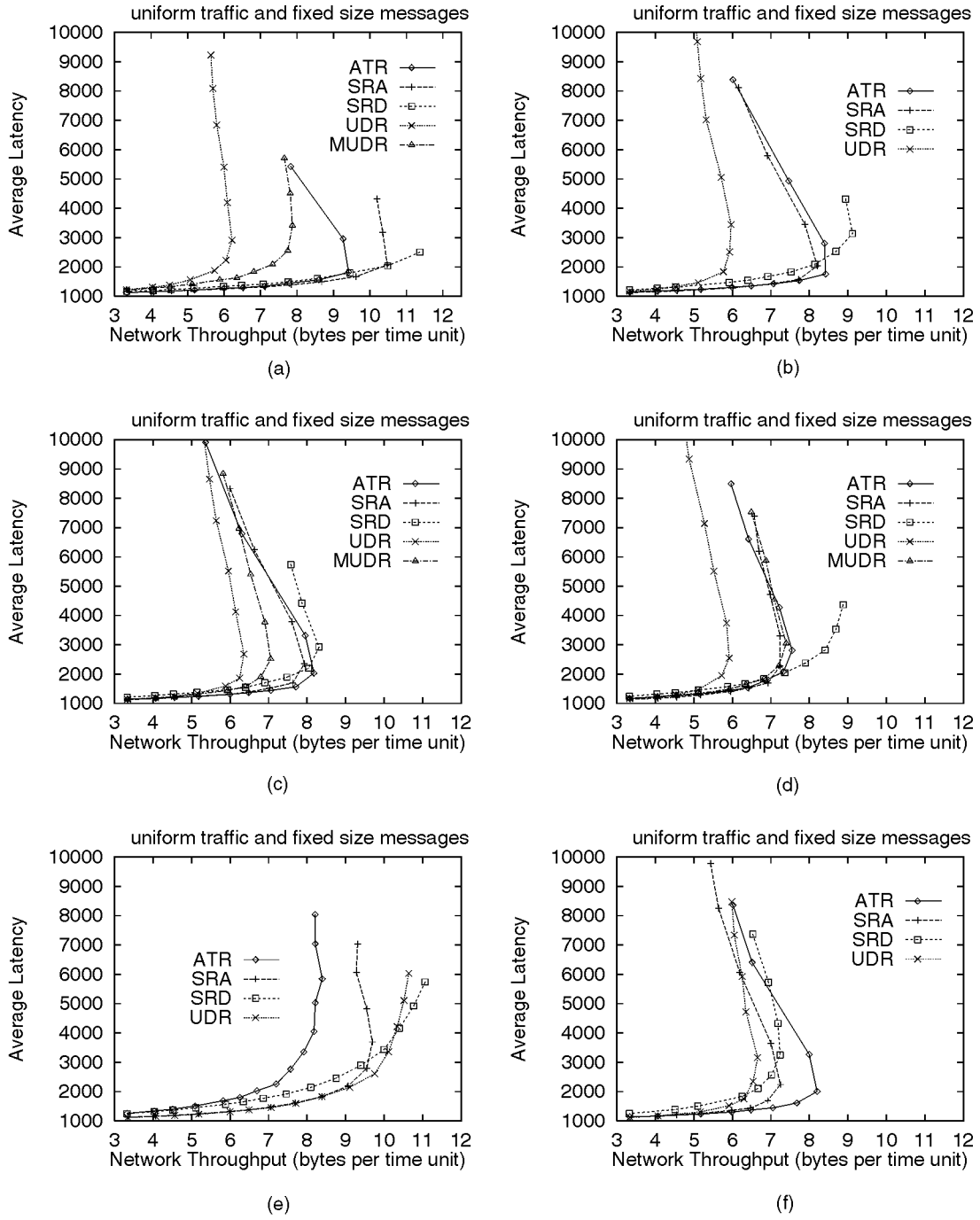


Fig. 10. Performance under uniform traffic with fixed message distribution.

5 Simulation Results and Observations

Section 5 shows some typical simulation results based on the simulation environment described in Section 4. To demonstrate the performance behavior of different routing algorithms, we run the simulations under different workloads by varying the value of x from large to small. The smaller the x is, the heavier the workload is. To make a fair comparison, we use the same x for all algorithms under the same combination of network parameters. Note that the actual effective network workload, which depends on both x and how fast a message leaves its source in each switch, may not be exactly the same for different algorithms. However, this should be enough to fairly and practically evaluate the performance. For each run, at least 30,000 messages have been received and 95 percent confidence interval on the average latency has been reached. The first 1,000 messages are passed to eliminate the start-up transient effect. The latency/throughput curves are used as our *performance curves*.

5.1 Uniform Traffic

In order to fully understand the effect of topologies and routing algorithms to network performance under uniform

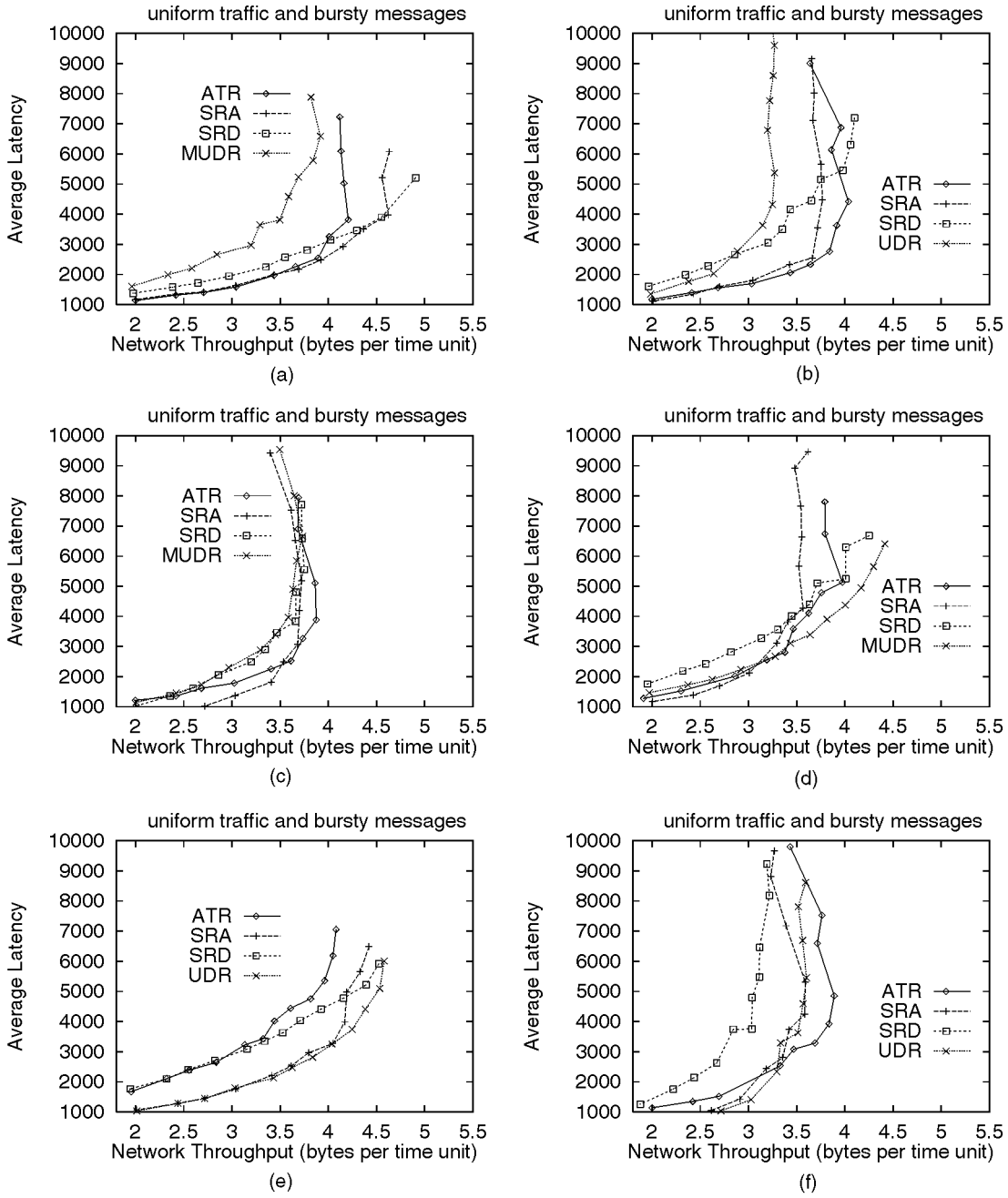


Fig. 11. Performance under uniform traffic with bursty message distribution.

traffic, we have measured the network performance for all six network topologies with fixed message size distribution and bursty message size distribution. Fig. 10 shows the performance curves with fixed message size distribution. It is observed that most performance curves reach their maximum sustained throughput under a certain workload. Then, the throughput will be decreased but the latency is still increased when the workload is further increased. For those curves which have not shown such behavior, they will do so if a heavier workload is given. Because channel contentions are very serious under such over-saturated workloads, many messages have to wait for a long time before they are able to use free channels, which causes higher transmission latency and lower network throughput. In a real network, a good flow control mechanism should be applied to avoid the over-saturated workloads.

5.1.1 The Effect of Routing Algorithms

Although these routing algorithms have different behavior under different topologies, there are still common behaviors for each algorithm. The SRA and SRD achieve good performance in most of uniform cases, but they depend on a linear programming solver [14], which is very time consuming.

Although the ATR is much simpler than the SR, it provides very close (even better in some cases) performance

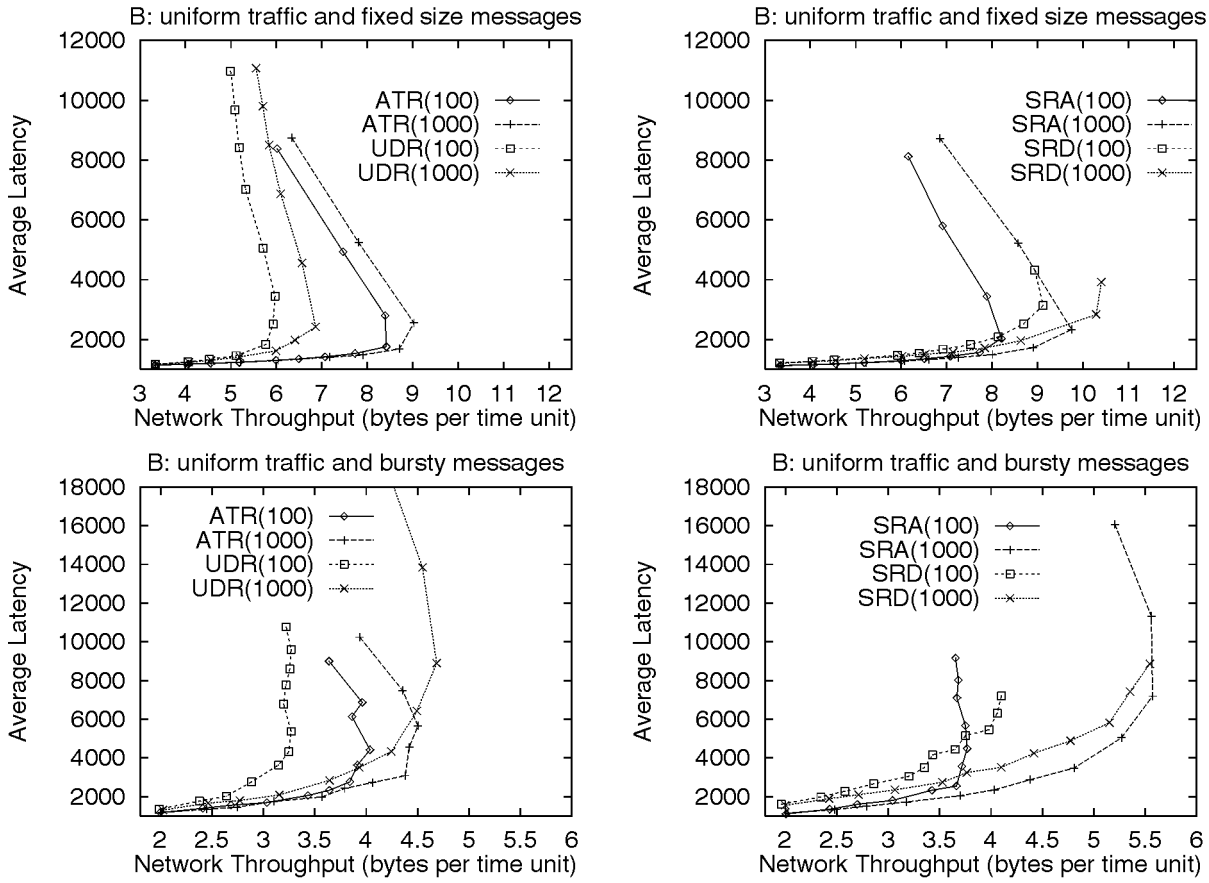


Fig. 12. Effect of buffer size under uniform traffic.

to the SR in all topologies except E. Because ATR uses Eulerian trails and shortcuts to provide reasonable and shorter paths, it is able to deliver a good performance. The ATR uses two opposite Eulerian trails to avoid deadlock. For E, some longer routing paths along the Eulerian trails have to be kept in order to avoid deadlock. Those longer paths consume more channel resources and may cause poor performance.

The UDR provides a worse performance in most of the topologies except E. First, it may concentrate traffic near the root switch. If the root switch does not have enough channels to transfer the traffic, serious contention may cause poor performance, which is the case for $A, B, C,$ and F. Second, the UDR allows a routing path as long as it never uses any "up" channel after using "down" channels. This may allow very long routing paths. For example, in E, $3 \rightarrow 4 \rightarrow 5 \rightarrow 7 \rightarrow 0$ may be a legal path if switch 0 is the root. The reason that the UDR performs very well in E is the avoidance of the above two shortcomings. As shown in Fig. 10, MUDR outperforms UDR because it eliminates very long routing paths. Although the comparison between UDR and MUDR is only made for A, C and D, the same trend holds for B, E, and F. To reduce the number of curves in the figures, we use MUDR curves for topologies A, C, and D and UDR curves for the rest of the topologies.

5.1.2 Adaptive Routing vs. Deterministic Routing

It is interesting to compare the performance of the SRA and SRD. Both of them are developed by the same philosophy and have been made to balance channel utilization under uniform traffic. For all the six topologies, the SRA achieves lower latency under light workload but suffers from worse performance under heavy workload. This is not a surprising observation because adaptive routing can provide an alternative routing path and take advantage of multiple paths between pairs under light workload. However, adaptive routing may take a longer path when shorter paths are not available. The misrouting can cause more channel contentions under heavy workload and result in poor performance. On the other hand, deterministic routing, especially traffic balanced deterministic routing like the SRD, always selects a unique and shorter path for a source-destination pair. Thus, a message has to wait on a busy channel while a free alternative path exists, which results in higher latency under a light workload. Under a heavy workload, such a disadvantage is gone because all channels are busy on transmission and a shorter path is a better choice.

5.1.3 Bursty Messages

As shown in Fig. 11, the performance comparison under the bursty messages is similar to that under the uniform traffic with fixed size messages. This is because they both have the

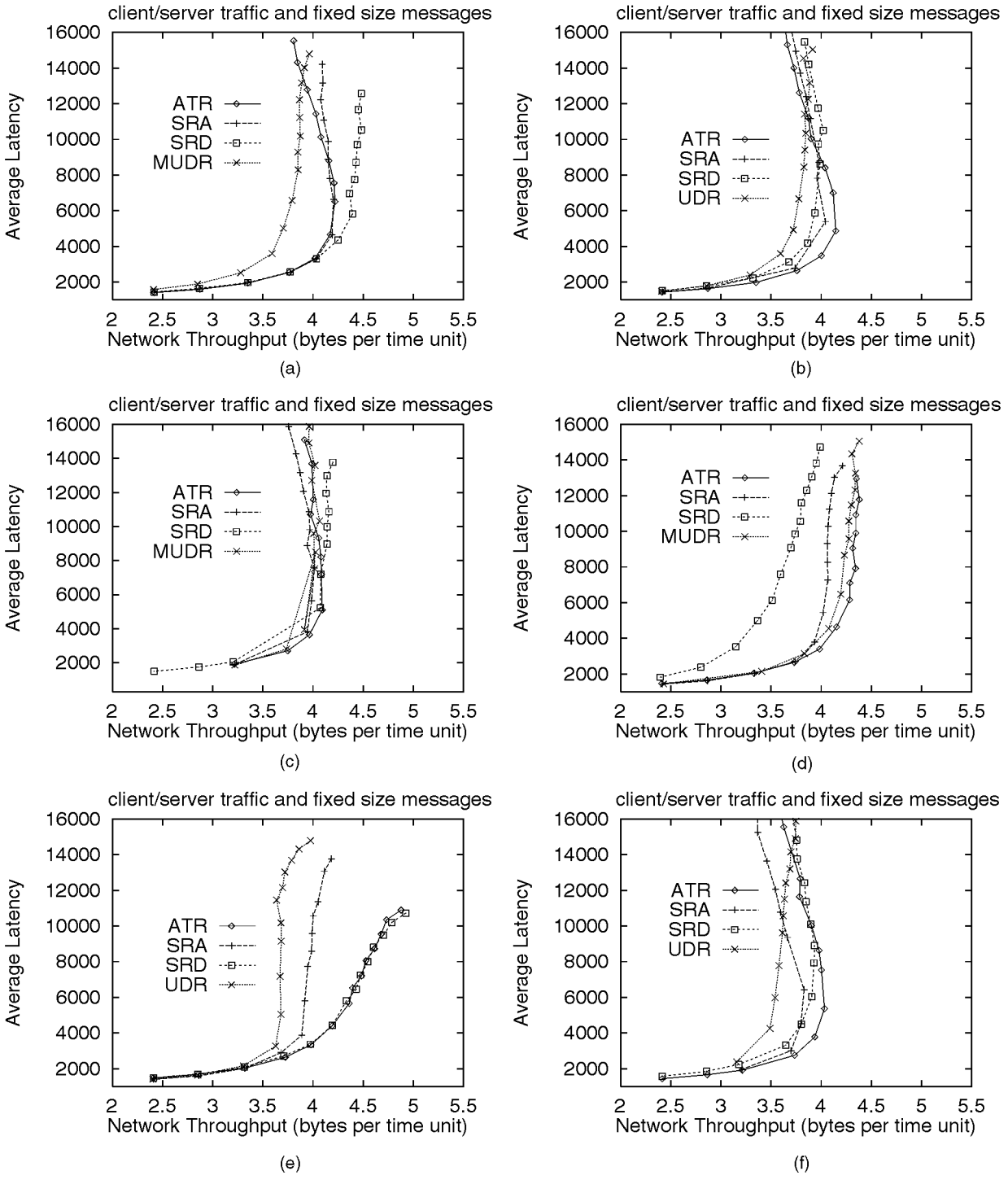


Fig. 13. Performance under client/server traffic.

same destination distribution. When long messages are transmitted in a network, some channels are occupied by these long messages and block many other messages, which causes higher network latency and lower network throughput for all topologies. The SRD causes higher latency because it cannot choose another available path.

5.1.4 Buffer Size

In order to demonstrate the effect of buffer size on performance, we measure the performance by using larger buffer capacity of 1,000 bytes. As shown in Fig. 12, large buffer capacity does improve the performance for all the algorithms under uniform traffic with both fixed size messages and bursty messages. This is because a larger buffer will hold more bytes and reduce the number of occupied channels by a blocked message. Due to the limitation of empty buffer required by the ATR (see Section 4.1), the UDR and SR gain more performance benefit from the larger buffer than the ATR, especially for bursty traffic.

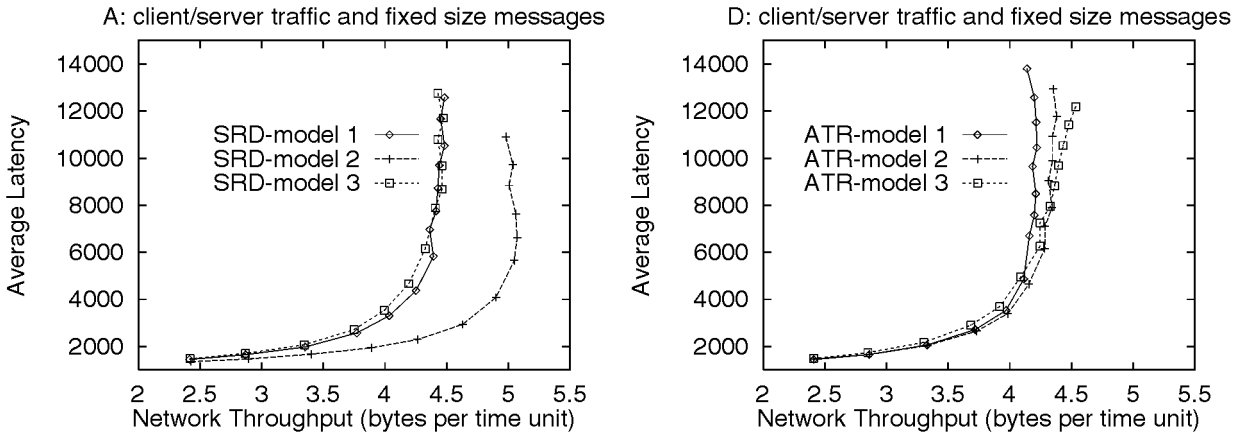


Fig. 14. Effect of server locations and number of servers.

5.2 Client/Server Traffic

The performance under client/server traffic is shown in Fig. 13. Let us look at Fig. 9 again. Four servers are assumed in each network shown in Fig. 9, where two server nodes are located at each of the shadowed switches with wider border and one server node is located at each of the shadowed switches with thinner border. Under client/server traffic, the routing algorithms may have different performance behavior from that of uniform traffic. For example, the ATR has the worst performance under uniform traffic in E, but it does provide a better performance than the UDR and the SRA in the selected client/server model. The reason is that many messages in such a client/server model will go to switches 0 and 9 and the ATR provides reasonable routing paths for these two destinations. Such an observation shows that we have to consider different traffic patterns and workload when we evaluate the performance for a network.

5.2.1 The Effect of Routing Algorithms

Due to the limitation of output contention, the performance difference under client/server traffic is not as significant as that under uniform traffic. An interesting thing is that the SRD delivers the best performance for E, but the worst performance for D. The SRD is designed for uniform traffic and it cannot adjust routing based on the traffic pattern due to its deterministic feature. Thus, it may cause poor performance for some specific traffic pattern. The SRA does not show such extreme performance (although it is also designed based on uniform traffic), but the SRA does have more flexibility to deal with various traffic patterns. A routing algorithm, which is based on uniform traffic and performs well under uniform traffic, may not provide good performance under client/server traffic.

5.2.2 Locations of Servers

The number of servers and the locations of servers both influence the network performance. In Fig. 14, we show their effects on performance in A and D, where three different models shown in Table 4 are used to study the effect for A and D, respectively. We select the SRD and the ATR to show the performance of A and D, respectively. Similar performance is observed in other routing algorithms and other topologies.

A is a network where trunk channels are evenly distributed among switches. Since no switch has extremely high connectivity, it is better to distribute the servers among different switches if possible. In Fig. 14, we compare the performance under three client/server models: model 1 has four servers located at four switches, model 2 has five severs located at five switches, and model 3 has five servers located at two switches. It is not surprising that model 2 achieves the best performance. Model 3 has the worst performance because it concentrates the traffic near two switches and channel bottlenecking becomes serious.

D is a network where switch 0 has full connection to all of the other switches. Servers should be allocated at switch 0 in order to use those channels for it. Again, we compare three models: model 1 has four servers located at four switches with one server at switch 0, model 2 has four servers located at three switches with two servers at switch 0, and model 3 has four servers located at two switches with three servers at switch 0. As shown in Fig. 14, while model 1 shows the worst performance, model 2 shows the best performance under light workload, and model 3 has the best performance under heavy workload. When three servers are located at switch 3, there is no client node in switch 3, which cannot take advantage of short paths to servers. That is why model 3 delivers higher latency with a light workload. When the workload becomes heavier, model 3 gains a little bit better performance, because it

TABLE 4
Three Models to Study the Effect of Server Locations for A and D

Models	Server location switch ids in A					Server location switch ids in D			
Model 1	2	3	8	9	-	0	4	5	7
Model 2	2	3	8	9	1	0	0	4	7
Model 3	2	2	2	8	8	0	0	0	7

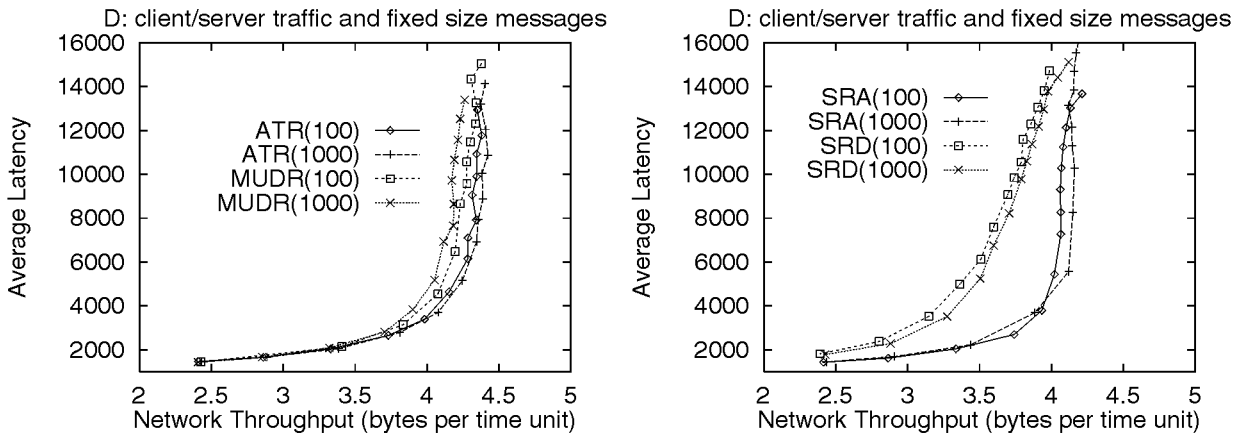


Fig. 15. Effect of buffer size under client/server traffic.

has the highest bandwidth to release traffic near servers. However, because output contention is the major bottleneck in a client/server environment, the performance difference among these three models is little.

5.2.3 Buffer Size

It is observed that buffer capacity does not have a big effect on performance under client/server traffic. Fig. 15 shows the trend in Topology D. This phenomenon is understandable, since the output contention and the channel contention near the servers are main bottlenecks in such an environment. To improve the performance under client/server traffic, output contention should be considered, and multiple ports may be used for a server to reduce output contention.

6 Conclusions and Future Work

Irregular networks using cut-through switches are a promising network platform to build workstation clusters for high performance computing. Such networks provide low network latency and are incrementally scalable. Without topological constraints, channels and switches may be added, removed, or reconfigured to adapt to a particular network traffic pattern of a given environment.

In order to allow irregular interconnection in such a switched network, we have presented a novel adaptive routing algorithm for irregular cut-through switched networks. The routing algorithm is simple to implement, and efficient to lead to reasonable performance for most topologies and traffic workloads. In order to understand the effect of routing algorithms and topologies on network performance, we have demonstrated simulation results comparing the performance of three different routing strategies. The UDR is extremely simple to understand and implement, yet leads to poor throughput for almost all topologies. Our proposed algorithm, the ATR, is just slightly more complicated to compute, yet leads to mostly reasonable throughput for most topologies. Although the SR spends much more time balancing channel utilization under uniform traffic than the ATR, the SR and the ATR actually have very close performance, except that the SRD has better performance under an over-saturated workload.

There are many directions worth further exploration. More simulations are needed to consider irregular network topologies with different shapes, sizes, numbers of nodes, and traffic distributions. In order to understand the effect of the finite input source model, the case of $K > 1$ should be studied.

It should not be difficult to apply our algorithm to many non-Eulerian graphs, if not all of them. For example, we can get an Eulerian trail by removing a small number of channels and add them as shortcuts to the adaptive trails. If virtual channels are allowed and two virtual channels are used for each physical link, all nodes will have even degrees. In this sense, our routing algorithm can be applied to any topology.

One way to apply our algorithm is to implement it on a commercial product such as Myrinet. We may have to modify our algorithm, because the Myrinet switch is using source routing and may not be able to support routing tables.

Appendix

We are going to show that the ATR algorithm is deadlock-free. In order to simplify our proof, we need the following definition of *deadlock-immune* channel which was first defined in [24].

Definition 4. *A channel c is deadlock-immune for a packet M if and only if, once M occupies the channel c, M will eventually leave channel c and release it. For convenience, if a packet M can never use channel c, c is also defined to be deadlock-immune for packet M. A channel is said to be deadlock-immune if and only if it is deadlock-immune for all packets.*

As shown in [24], a routing algorithm is deadlock-free if and only if every channel is deadlock-immune. This is obvious, because the channels involved in a deadlock could not be deadlock-immune.

In the ATR routing algorithm, a packet is transmitted to its destination along $AT1$ and/or $AT2$. Although the

routing tables are created separately based on each adaptive trail, they are finally combined together to become integrated routing tables. Due to the use of a source shortcut, it is possible for a packet to start from one trail and change its route to another trail. Note that this situation will not happen unless there is a channel on its routing path, which is also used as a source shortcut on another trail.

Let ET1 be

$$v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_m$$

and ET2 be

$$u_0 \rightarrow u_1 \rightarrow \cdots \rightarrow u_m,$$

where $u_i = v_{m-i}$. In the following, $c^1_{i,i+1}$ and $s^1_{p,q}$ represent a channel in C(ET1) and a shortcut of AT1, respectively, and $c^2_{i,i+1}$ and $s^2_{p,q}$ represent a channel in C(ET2) and a shortcut of AT2, respectively.

Lemma 1. *If none of the channels $c^1_{i,i+1}$ $(i \geq k)$ in C(ET1) are a source shortcut in AT2, all channels $c^1_{i,i+1}$'s $(i \geq k)$ are deadlock-immune.*

Proof. Let us consider any packet, whether it is originally routed along AT1 or AT2. There are three cases:

1. The packet holds a channel $c^1_{i,i+1}$ $(i \geq k)$ when it is routed along AT1. Then its route can no longer be changed to AT2, because none of $c^1_{i,i+1}$ $(i \geq k)$ is a source shortcut in AT2. If any packet is holding $c^1_{m-1,m}$, the packet will arrive in destination v_m and release $c^1_{m-1,m}$. After $c^1_{m-1,m}$ is released, any packet holding $c^1_{m-2,m-1}$ will go to its destination (either v_m or v_{m-1}) and release $c^1_{m-2,m-1}$. Eventually, $c^1_{m-3,m-2}, \cdots$, and $c^1_{k,k+1}$ will be released by the packet because any packet holding $c^1_{i,i+1}$ $(i \geq k)$ can always go to the destination via path $v_{i+1} \rightarrow v_{i+2} \rightarrow \cdots \rightarrow destination$. Therefore, all channels $c^1_{i,i+1}$ $(i \geq k)$ are deadlock-immune for such a packet.
2. The packet holds a channel $c^1_{i,i+1}$ $(i \geq k)$ when it is routed along AT2. It means that $c^1_{i,i+1}$ is used as a destination shortcut in AT2 (because it is not a source shortcut) for the packet. The packet will go to the destination and release the channel eventually. Therefore, each $c^1_{i,i+1}$ $(i \geq k)$ is deadlock-immune for such a packet.
3. The packet never takes any channel $c^1_{i,i+1}$ $(i \geq k)$. By definition, $c^1_{i,i+1}$ $(i \geq k)$ is deadlock-immune for such a packet.

Therefore, $c_{i,i+1}$ is deadlock-immune. □

Lemma 2. *If none of the channels $c^2_{i,i+1}$ $(i \geq k)$ in C(ET2) are a source shortcut in AT1, then all of the channels $c^2_{i,i+1}$'s $(i \geq k)$ are deadlock-immune.*

The proof is similar to that of Lemma 1.

Lemma 3. *Channel $c^1_{m-1,m}$ is deadlock-immune.*

Proof. If $c^1_{m-1,m}$ is not a source shortcut in AT2, it is deadlock-immune by Lemma 1. Otherwise, let $s^2_{p,q}$ be the source shortcut, which corresponds to $c^1_{m-1,m}$ in AT2. We claim that none of channel $c^2_{j,j+1}$ $(j \geq q)$ can be a source shortcut in AT1. Because otherwise, there will be a dependency cycle as follows, which is the same deadlock configuration shown in Fig. 6 and should not be allowed in our adaptive trails.

- A T 1 : $v_0 \rightarrow \cdots \rightarrow \overrightarrow{v_x \rightarrow \cdots \rightarrow v_y} \cdots v_{m-1} \rightarrow v_m$ (where $s^1_{x,y} = c^2_{j,j+1}$).
- A T 2 : $u_0 \rightarrow \cdots \rightarrow \overrightarrow{u_p \rightarrow \cdots \rightarrow u_q} \cdots u_j \rightarrow u_{j+1} \rightarrow \cdots u_m$ (where $s^2_{p,q} = c^1_{m-1,m}$).

Therefore, all $c^2_{j,j+1}$ is deadlock-immune by Lemma 2. In this case, if any packet holds $s^2_{p,q}$ as a source shortcut, it will go to its destination via $u_q \rightarrow u_{q+1} \rightarrow \cdots \rightarrow destination$ and eventually release $s^2_{p,q}$. So, $c^1_{m-1,m}$ is deadlock-immune for any packet. □

Lemma 4. *Channel $c^2_{m-1,m}$ is deadlock-immune.*

The proof is similar to that of Lemma 1 .

Lemma 5. *For any channel $c^1_{k,k+1}$ in C(ET1), if all channels $c^1_{i,i+1}$ $(i > k)$ are deadlock-immune, $c^1_{k,k+1}$ is deadlock-immune.*

Proof. For any packet, there are the following four cases:

1. The packet never takes channel $c^1_{k,k+1}$. Then by definition, $c^1_{k,k+1}$ is deadlock-immune for the packet.
2. The packet holds channel $c^1_{k,k+1}$ and its destination is switch v_{k+1}. In this case, the packet will finally go to destination and free $c^1_{k,k+1}$. So $c^1_{k,k+1}$ is deadlock-immune for the packet.
3. The packet holds $c^1_{k,k+1}$ and requests $c^1_{k+1,k+2}$ in AT1. Since $c^1_{i,i+1}$'s $(i > k)$ are deadlock-immune, they will be available for the packet. Therefore, the packet will eventually go through $v_{k+1} \rightarrow v_{k+2} \rightarrow \cdots \rightarrow destination$ and reach the destination. In this case, $c^1_{k,k+1}$ is deadlock-immune for the packet.
4. The packet holds $c^1_{k,k+1}$ as a source shortcut in AT2. Suppose the source shortcut is $s^2_{p,q}$. Then the packet requests $c^2_{q,q+1}$. If none of $c^2_{j,j+1}$ $(j \geq q)$ is a source shortcut in AT1, $c^2_{j,j+1}$'s are deadlock-immune by Lemma 2. It means that the packet can reach its destination via $c^2_{j,j+1}$'s. If some $c^2_{j,j+1}$ is a source shortcut in AT1, let $s^1_{x,y}$ be a source shortcut corresponding to $c^2_{j,j+1}$ in AT1 and there must be $y > k$. Otherwise, there will be a dependence cycle as follows, which is the same deadlock configuration shown in Fig. 6 and should not be allowed in our adaptive trails.

$$\begin{aligned} \text{AT1} : v_0 \rightarrow \cdots \rightarrow \overrightarrow{v_x \rightarrow \cdots \rightarrow v_y} \cdots v_k \rightarrow v_{k+1} \\ \rightarrow \cdots v_m (\text{where } s^1_{x,y} = c^2_{j,j+1}) \end{aligned}$$

$$\begin{aligned} \text{AT2} : u_0 \rightarrow \cdots \rightarrow \overrightarrow{u_p \rightarrow \cdots \rightarrow u_q} \cdots u_j \rightarrow u_{j+1} \\ \rightarrow \cdots u_m (\text{where } s^2_{p,q} = c^1_{k,k+1}) \end{aligned}$$

Therefore, we must have $y > k$. Any packet holding $s^1_{x,y}$ $(y > k)$ will go to its destination by path $v_y \rightarrow v_{y+1} \rightarrow \cdots \rightarrow destination$ because all

channels $c^1_{i,i+1}$'s $(i > k)$ are deadlock-immune, which in turn means that $s^1_{x,y}$ (i.e. $c^2_{j,j+1}$) is deadlock-immune. Therefore, the packet holding $s^2_{p,q}$ (i.e., $c^1_{k,k+1}$) can go to its destination via $u_q \rightarrow u_{q+1} \rightarrow \cdots \rightarrow destination$ and release $s^2_{p,q}$. So $c^1_{k,k+1}$ is deadlock-immune for the packet in this case.

Therefore, $c^1_{k,k+1}$ is deadlock-immune. □

Lemma 6. *For any channel $c^2_{k,k+1}$ in C(ET2), if all channels $c^2_{j,j+1}$ $(j > k)$ are deadlock-immune, $c^2_{k,k+1}$ is deadlock-immune.*

The proof is similar to that of Lemma 5.

Theory 1. *The ATR routing algorithm is deadlock-free.*

Proof. Due to Lemmas 3, 4, 5, and 6, channels $c^1_{m-1,m}, c^1_{m-2,m-1}, \cdots, c^1_{1,2}$, channels $c^1_{0,1}$, channels

$$c^2_{m-1,m}, c^2_{m-2,m-1}, \cdots, c^2_{1,2},$$

and channels $c^2_{0,1}$ are proved to be deadlock-immune one by one. Therefore, the routing algorithm is deadlock-free. Otherwise, some channels must not be deadlock-immune, because channels involved in a deadlock cannot be deadlock-immune. □

ACKNOWLEDGMENTS

This work was supported in part by the U.S. National Science Foundation grants MIP-9204066 and MIP-9528903, the U.S. Department of Energy grant DE-FG02-93ER25167, and a grant from Hewlett-Packard Laboratories.

REFERENCES

[1] L.M. Ni, "Issues in Designing Truly Scalable Interconnection Networks," *Proc. 1996 ICPP Workshop on Challenges for Parallel Processing,* Aug. 1996.
[2] R.J. Souza et al., "The GIGA Switch System: A High-Performance Packet Switching Platform," *Digital Technical J.,* vol. 6, Jan. 1994.
[3] Ancor Comm. Inc., http://www.ancor.com/cxt.html.
[4] T.W. Giorgis, "29 Switching Hubs Save the Bandwidth," *BYTE,* pp. 162-169, July 1995.
[5] 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,* vol. 15, pp. 29-36, Feb. 1995.
[6] C.B. Stunkel et al., "The SP1 High-Performance Switch," *Proc. 1994 Scalable High Performance Computing Conf. (SHPCC '94),* pp. 150-157, May 1994.
[7] "Cray T3D System Architecture Overview," technical report, Cray Research Inc., Sept. 1993.
[8] M. Noakes, D.A. Wallach, and W.J. Dally, "The J-Machine Multicomputer: An Architecture Evaluation," *Int'l Symp. Computer Architecture,* pp. 224-236, 1993.
[9] B. Duxett and R. Buck, "An Overview of the Ncube-3 Supercomputer," *Proc. Frontiers of Massively Parallel Computation,* pp. 458-464, 1992.
[10] *Paragon XP/S Product Overview,* technical report, Intel Corp., 1991.
[11] L.M. Ni and P.K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks," *Computer,* vol. 26, pp. 62-76, Feb. 1993.
[12] R. Horst, "ServerNet Deadlock Avoidance and Fractahefral Topologies," *Proc. Int'l Parallel Processing Symp.,* pp. 274-280, Apr. 1996.
[13] S.S. Owicki and A.R. Karlin, "Factors in the Performance of the AN1 Computer Network," *Performance Evaluation Rev.,* vol. 20, pp. 167-180, June 1992.
[14] L. Cherkasova, V. Kotov, and T. Rokicki, "Fibre Channel Fabrics: Evaluation and Design," *Proc. 29th Hawaii Int'l Conf. System Sciences,* Feb. 1995.
[15] W. Qiao and L.M. Ni, "Adaptive Routing in Irregular Networks Using Cut-Throughput Switches," *Proc. 1996 Int'l Conf. Parallel Processing,* vol. 1, pp. 52-60, Aug. 1996. http://www.acs.cps.msu.edu/TechReports/tr-id.html.
[16] M. Yang and L.M. Ni, "Design of Scalable and Multicast Capable Cut-Through Switches for High Speed LANs," *INFOCOM'97,* Technical Report MSU-CPS-ACS-96-08, Dept. Computer Science, Michigan State Univ., E. Lansing, Mich., June 1996.
[17] P. Kermani and L. Kleinrock, "Virtual Cut-Through: A New Computer Communication Switching Technique," *Computer Networks,* vol. 3, no. 4, 1979.
[18] R.O. Onvural, *Asynchronous Transfer Mode Networks: Performance Issues,* 1995.
[19] W. Dally and C. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Trans. Computers* vol. 36, pp. 547-553, May 1987.
[20] C.J. Glass and L.M. Ni, "The Turn Model for Adaptive Routing," *J. ACM,* vol. 41, pp. 874-902, Sept. 1994.
[21] J. Duato, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 4, Nov. 1993.
[22] H. Park and D.P. Agrawal, "Generic Methodologies for Deadlock-Free Routing," *Proc. Int'l Parallel Processing Symp.,* pp. 638-643, Apr. 1996.
[23] A.K. VenKatramani, T.M. Pinkston, and J. Duato, "Generalized Theory for Deadlock-Free Adaptive Wormhole Routing and Its Application to Disha Concurrent," *Proc. Int'l Parallel Processing Symp.,* pp. 815-821, Apr. 1996.
[24] X. Lin, P. McKinley, and L. Ni, "The Message Flow Model for Routing in Wormhole-Routed Networks," *IEEE Trans. Parallel and Distributed Systems,* pp. 755- 760, July 1995.
[25] W. Qiao and L.M. Ni, "Adaptive Routing in Irregular Networks Using Cut-Throughput Switches," Technical Report MSU-CPS-ACS-108, Dept. Computer Science, Michigan State Univ., E. Lansing, Mich., Dec. 1995.
[26] F. Harary, *Graph Theory.* Reading, Mass.: Addison-Wesley, 1972.
[27] A.A. Chien and M. Konstantinidou, "Workload and Performance Metrics for Evaluating Parallel Interconnects," *IEEE Computer Architecture Technical Committee Newsletter,* pp. 23-27, no. 2, 1994.

Wenjian Qiao received the BS and MS degrees in computer science from Peking University, Beijing, China, in 1989 and 1992, respectively. She received the PhD degree in computer science from Michigan State University, Lansing in 1997. She is currently a technical staff member of the IBM Transarc Lab in Pittsburgh. Her research interests include distributed systems, performance analysis, benchmarks of transaction processing, and computer networking. She is a member of the IEEE Computer Society.

Lionel M. Ni earned his BS degree in electrical engineering from the National Taiwan University in 1973, and his PhD degree in electrical engineering from Purdue University in 1980. He is currently a professor for the Computer Science and Engineering Department at Michigan State University, Lansing. He has published more than 160 technical articles in the areas of high speed networks and high-performance computer systems, and has served as an editor of many professional journals. He was the program director of the National Science Foundation Microelectronic Systems Architecture Program from 1995 to 1996. As a fellow of the IEEE, he has served as program chair or general chair of many professional conferences. He has also received a number of awards for his papers and won the Michigan State University Distinguished Faculty Award in 1994.

Tomas Rokicki received his BS in electrical engineering from Texas A&M University in 1985, and his PhD in computer science from Stanford University in 1994. He is presently a member of the technical staff at Hewlett-Packard Laboratories in Palo Alto, California. His interests include real-time verification and computer networks.