An Efficient and Deadlock-Free Network Reconfiguration Protocol

Olav Lysne, *Member*, *IEEE*, José Miguel Montañana, José Flich, *Member*, *IEEE Computer Society*, José Duato, *Member*, *IEEE*, Timothy Mark Pinkston, *Senior Member*, *IEEE*, and Tor Skeie

Abstract—Component failures and planned component replacements cause changes in the topology and routing paths supplied by the interconnection network of a parallel processor system over time. Such changes may require the network to be reconfigured such that the existing routing function is replaced by one that enables packets to reach their intended destinations amid the changes. Efficient reconfiguration methods are desired which allow the network to function uninterruptedly over the course of the reconfiguration process while remaining free from deadlocking behavior. In this paper, we propose, evaluate, and prove the deadlock freedom of a new network reconfiguration protocol that overlaps various phases of "static" reconfiguration processes traditionally used in commercial and research systems to provide performance efficiency on par with that of recently proposed "dynamic" reconfiguration processes but without their complexity. Simulation results show that the proposed *Overlapping Static Reconfiguration* protocol can reduce reconfiguration time by up to 50 percent, reduce packet latency by several orders of magnitude, reduce packet dropping by an order of magnitude, and provide unhalted packet injection as compared to traditional static reconfiguration while allowing network throughput similar to dynamic reconfiguration.

Index Terms—Interconnection network, network reconfiguration protocol, deadlock freedom, routing algorithm, reliability and fault tolerance.

1 Introduction

As the number of components in a system increases, maintaining the desired system dependability and availability requirements becomes increasingly challenging. Interconnection networks serve as the communication infrastructure of parallel processing systems, enabling the various processing, memory, storage, and I/O components of the system to communicate. They are found in high-end servers [1], [2], [3], [4] in the form of system area networks [5], [6], [7], [8], [9], [10], [11] as well as in multicore processors [12], [13], [14], [15] as networks-on-chip (NoCs) [16], [17], [18] at the other end of the spectrum. The network plays a critical role in determining system performance and dependability as the interaction and cooperation of other system components ultimately depend on its ability to establish communication paths between those components.

Many techniques have been developed over the years to improve interconnection networks [19], [20], [21], [22], [23], [24]. A central problem that must be addressed in the design of a network is how deadlock can be handled. Deadlock in an interconnection network is the anomalous situation in which packets transported within the network hold onto a set of fully occupied resources in a cyclically knotted manner while waiting endlessly for some resource within that set to become available [25]. Unless somehow avoided or resolved quickly after beginning to form, deadlock can cause the entire system to fail as none of the packets in the set can move first, thus eventually stalling forward progress of other (possibly all) packets in the network. Network deadlock is generally avoided through careful design of the routing algorithm, i.e., the routing function. In some cases, however, the premises upon which the design is based may break. This can happen in unexpected and unplanned ways when network switches or links suddenly fail; alternatively, this can occur in a controlled and planned way by the user/system operator to replace or service switches and links. A change in network configuration often requires a replacement of the deadlock-free routing function with another one that takes the changes into account. The routing function that prevails during the reconfiguration process in transitioning from the old to the new should maintain the deadlock-free properties of the individual routing functions, i.e., it should also be deadlock-free. This is trivially done when the change consists only of adding new network resources. When removing network resources (i.e., due to failure) or changing the routing paths over existing resources, the process is less trivially defined.

Deadlock-free network reconfiguration schemes developed thus far to confront this problem are based on two approaches. In *static reconfiguration* (SR) [1], [26], [27], no packets routed according to the new routing function are allowed to be injected into the network while there still are packets in the network routed according to the old routing function. In *dynamic reconfiguration* [28], [29], [30], [31], [32], [33], [34], [35], both can coexist in the network, with packets

- *O. Lysne and T. Skeie are with the Department of Informatics, University of Oslo, and the Simula Research Laboratory, PO Box 134, NO-1325 Lysaker, Norway. E-mail: {olav.lysne, tskeie}@simula.no.*
- *J. Flich, J.M. Montañana, and J. Duato are with the Departamento DISCA, Universidad Politécnica de Valencia, Camino de Vera, s/n, 46020 Valencia, Spain. E-mail: jflich@disca.upv.es, {jmontana, jduato}@gap.upv.es.*
- *T.M. Pinkston is with the Ming Hsieh Department of Electrical Engineering, University of Southern California, 3740 McClintock Ave., EEB-208, Los Angeles, CA 90089-2562. E-mail: tpink@usc.edu.*

Manuscript received 20 Mar. 2007; revised 11 Oct. 2007; accepted 12 Dec. 2007; published online 12 Feb. 2008.
Recommended for acceptance by Y. Yang.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2007-03-0097.
Digital Object Identifier no. 10.1109/TC.2008.31.

being routed according to the new routing function, possibly arriving at destinations before packets routed according to the old routing function. Because of its serialization in executing reconfiguration phases, SR leads to high packet latencies, high packet dropping rates in case of link/switch failures, and low throughput, which may be unacceptable for many applications targeted for computational data centers, highly available Web servers, video servers, and process control servers. Dynamic reconfiguration can lead to fewer packets that miss their quality-of-service (QoS) deadline, a dramatically reduced number of dropped packets, and no network downtime. Some proposed techniques, however, are applicable only to a limited set of routing functions [29], rely on dropping packets to avoid deadlocks [30], appear to be more complex than the straightforward static approach [33], or have requirements on the minimal set of hardware resources implemented [31]. The Double Scheme (DS) [31], for example, is simple and general, i.e., applicable to any network topology and routing function, and it does not rely on dropping packets to avoid reconfiguration-induced deadlocks, but it requires the network to implement two sets of data virtual channels.

In this paper, we present a very simple method for reconfiguring a network efficiently in a deadlock-free manner in the event of faults or other circumstances requiring routing paths to change over existing resources. The proposed new network reconfiguration protocol is generally applicable to any network topology and routing function and it does not require multiple sets of data virtual channels or packet dropping to maintain deadlock freedom.[1] In addition, it guarantees in-order delivery of packets during reconfiguration when the old and new routing functions are deterministic and, for that reason, can offload much of the fault-handling burden from higher level network protocols. Simulation results reveal that the proposed network reconfiguration protocol can reduce the reconfiguration time by up to 50 percent, reduce the packet latency by several orders of magnitude, reduce packet dropping by up to nearly an order of magnitude, and provide unhalted packet injection as compared to traditional SR while allowing network throughput similar to the DS dynamic reconfiguration.

2 Idea Description

Network reconfiguration is invoked after the need for it is detected and notified across the network, as described in [1], [29], [31]. Static network reconfiguration is traditionally based on the notion of global barrier synchronization [1], [26], [27]. It consists of three phases separated essentially by two global synchronization operations, as described below:

Phase 1: Halt packet injection globally at all network source end nodes to allow the network to drain itself of all previously injected packets. Simultaneous to this, the new routing function can be calculated and distributed globally across the network but cannot yet be activated. Note that the start of network drainage could have occurred asynchronously at any time prior to this, but full network drainage completes globally by the end of this phase.

Barrier synchronization 1: Packets must be completely drained from the network globally across the entire network. As some links and end nodes could be drained far in advance of others, this barrier imposes an unnecessary constraint that can cause significant inefficiencies in the reconfiguration process.

Phase 2: Activate the new routing function globally across all input ports of all switches. Note that uploading of the new routing function could have occurred asynchronously at any time prior to this (either in Phase 1 or at the beginning of Phase 2), but its activation occurs anytime after the start of this phase and completes globally by the end of this phase.

Barrier synchronization 2: The new routing function must be activated globally across the entire network. Again, as some switches and end nodes could be activated far in advance of others, this barrier imposes an unnecessary constraint that can cause significant inefficiencies in the reconfiguration process.

Phase 3: Packet injection at the network end nodes is allowed to resume. Note that the resumption of packet injection may occur asynchronously across the network at any time after the start of this phase and it completes globally across the network by the end of this phase.

Synchronizing globally across the network prevents overlap between any part of the different reconfiguration phases by any of the switches and end nodes throughout the network. Although this can be inefficient (and, thus, is the target of our optimizations), it at least ensures two important invariants:

Invariant 1: Packets use either only the new routing function (these are referred to as new packets) or only the old routing function (these are referred to as old packets). This property ensures in-order delivery of packets when the routing functions are deterministic as no new packet can overtake any old packet.

Invariant 2: There are no cyclic-wait dependencies between packets due to reconfiguration. This property ensures that the network cannot deadlock on account of the reconfiguration process itself.

While these invariants are very useful properties for any reconfiguration process, we propose a much more efficient way in which we provide them without the need for global synchronization. The basic idea is to synchronize between processes *locally*, only when needed, by using reconfiguration tokens in order to allow operations within different phases to be overlapped in time across the network. An informal description of how this can be done is given in the following and serves as the basis for our proposed *Overlapping Static Reconfiguration* (OSR) protocol.

We can begin to understand how the phases of traditional static network reconfiguration can be overlapped by first analyzing the transition between Phases 1 and 2, both locally and globally, across the network. In order to maintain Invariant 1 while overlapping the two phases, input ports of network switches must not activate the new routing function if old packets can still be received at those ports. If this is upheld, Invariant 2 will be trivially fulfilled as long as synchronization between Phases 2 and 3 remains intact (we deal with this in what follows). Thus, Phases 1 and 2 can be overlapped by allowing each input channel in each switch to

1. Any dropped packets arising from this protocol are due to truncation or arriving at a disconnected link or switch arising from a fault.

activate the new routing function locally only after it has forwarded its last old packet, even if this and all other packets have not yet reached their destinations globally across the network. This way, Phase 2 will start locally on each switch input port immediately after the port has finished Phase 1. The mechanism that we propose for implementing this is based on reconfiguration tokens.

Reconfiguration tokens serve as local synchronizers between reconfiguration phases. Each source node injects a reconfiguration token into the network after its last designated old packet has been injected. The token is associated and propagates with the last old packet (or with no packet if none needs to be injected). When an input port of any switch receives and processes a token, it is designated as having locally completed Phase 1 of the reconfiguration process. Alternatively, if an input port is connected to a link or neighboring switch that has been detected as being faulty, it likewise is designated as having completed Phase 1 and locally generates a token. *Local synchronization occurs by multicasting the token to all output ports to which the input port could transmit packets according to the old routing function.* This notifies output ports that this channel is now completely drained of all old packets. The input port is then allowed to proceed locally to Phase 2, switching from the old routing function to the new one simply by activating the new routing function for the channel.

In transitioning from Phases 1 to 2 globally throughout the network, each output port in each switch also synchronizes locally, transmitting a token to the next neighboring switch immediately after all tokens have been received and processed from all input ports from which the output port can receive packets according to the old routing function. This way, tokens propagate within switches and between switches throughout the network in the order of the channel dependency graph[2] of the old routing function. If this channel dependency graph is acyclic, each input port of each switch will receive a token exactly once. Phase 2 completes globally once all reconfiguration tokens have been processed at all input and output ports of all switches and end nodes. This signifies that old packets no longer exist in the network.

We now turn our attention to the transition between Phases 2 and 3 locally and globally across the network. In order to maintain Invariant 1 while overlapping the two phases, all input ports of network switches encountered by new packets must activate the new routing function first before routing those packets.[3] Thus, Phases 2 and 3 can be overlapped by resuming the injection of packets into switch input ports from source nodes and output ports of neighboring switches only after those input ports have activated the new routing function locally, even if the new routing function is not yet activated at all switch input ports globally across the network. This way, Phase 3 can start locally on each switch input port immediately after the switch input port has finished Phase 2. Again, our proposed reconfiguration token mechanism can be used for implementing this as well.

Each source node can resume injecting packets (i.e., new packets) locally into the network (i.e., into the input ports of attached switches) immediately after having transmitted a reconfiguration token associated with the last designated old packet, if any. Likewise, each switch output port can resume transporting packets (i.e., new packets) to the input port of its neighboring switch immediately after having transmitted a reconfiguration token to that channel. *Local synchronization occurs by not allowing any new packets to be forwarded from any input port to a target output port within switches before a token has been transmitted from that output port.* Those new packets will remain buffered, as normal, at the input ports. Output ports of switches are blocked from accepting new packets, even from input ports that have already processed their token, until the new routing function is activated locally at the switch for *all* input ports from which the output port was able to receive old packets.

The transition from Phases 2 to 3 globally throughout the network occurs similarly to the transition from Phases 1 to 2: The activation of the new routing function for each input port in each switch synchronizes locally with the output port of the neighboring switch or source node via the transmission of a reconfiguration token. This occurs throughout the network in the order of the channel dependency graph of the old routing function until all input ports receive a token. This way, channels will transmit, in channel dependency order, first only old packets (if any), then a reconfiguration token, and, finally, only new packets (if any). Similarly to Phase 2, Phase 3 completes globally once all reconfiguration tokens have been processed at all input and output ports of all network switches and end nodes.

One remaining issue is whether localizing the synchronization between Phases 2 and 3 violates Invariant 2 globally across the network. As the following reasoning shows, it does not. Any cyclic-wait dependencies will have to contain both old and new packets; otherwise, either the old or the new routing function would be prone to deadlocking by themselves, which is assumed to not occur (i.e., both are independently deadlock-free by design). Hence, for there to be cyclic-wait dependencies containing both old and new packets, there would have to be an old packet waiting behind a new one and vice versa. This would, in turn, mean that an old packet would have to enter into a channel that has already accepted a new packet. But then, this old packet would have arrived on the channel after a reconfiguration token, which is contradictory to what is described above to maintain Invariant 1. Thus, Invariant 2 is also maintained.

An illustration of how this works is given in Fig. 1, where we have depicted a 2×2 mesh network at various stages of the reconfiguration process. At time T1, the routing algorithm is first vertical and is then horizontal, giving dependencies from vertical channels to horizontal channels, as the arrows indicate. The reconfiguration process ends at time Tn, where the routing is first horizontal and then is vertical. It is clear that, if the dependencies at time T1 and those at time Tn coexisted at any point in time, we would have a cycle of dependencies and, thus, danger of deadlock. At time T1, the old routing function is intact. Assume now that reconfiguration starts and that switch 1 receives the reconfiguration token on its vertical link. This means that it

2. The term channel dependency graph is defined in Section 3.2. If the channel dependency graph contains cycles (i.e., due to an adaptive routing function), a preprocessing stage could be used to remove some adaptive routing choices, resulting in a restricted (but still connected) routing function with an acyclic channel dependency graph through which tokens propagate. Previous results on deadlock-free routing imply this is always possible for any deadlock-free adaptive routing function [22], [23], [37].

3. The domain of the routing function is defined on input channels and not on switches, as described in Section 3.

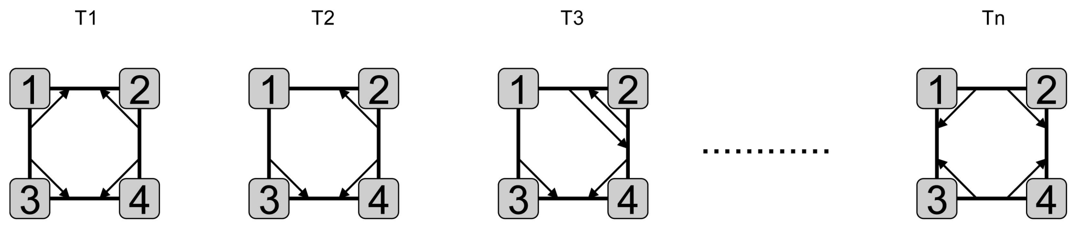


Fig. 1. The development in channel dependencies as Overlapping Static Reconfiguration proceeds.

can no longer receive old packets on this link. Since no new packet will make a vertical-to-horizontal turn, the dependency between the links connected to switch 1 disappears at time T2. Now, switch 1 can start sending new packets on its horizontal link; thus, switch 2 may receive packets that need to do a horizontal-to-vertical turn. This gives rise to a new dependency between the links of switch 2 at time T3. The important observation is that the introduction of the new dependency around switch 2 can only happen after the dependency around switch 1 has been removed. This way, our reconfiguration method, in a very simple way, guarantees that old dependencies are removed and new ones are introduced in a controlled order, without ever having a cycle of coexisting dependencies.

Finally, the protocol described in this paper assumes that one network reconfiguration occurs at a time. Although this is a relatively natural and practical assumption, it can easily be relaxed without significantly changing the base protocol. The main requirement is that the sequence of routing functions active on each input channel must be the same for all input channels on all switches, which means that there is a global sequence of routing functions that all components agree upon. Thus, the base protocol can be extended in an obvious way: No packet that is routed according to a routing function will have to wait behind a packet that is routed according to a routing function that occurs later in the sequence. Tokens must also carry information on which reconfiguration they belong to, i.e., between which pair of routing functions a token demarks in the transition.

We wrap up this informal description with some additional remarks on the motivation for overlapping the phases of traditional static reconfiguration. We stated previously that there are two generic reasons for optimizing operations this way: either to shorten the time from the start to the end of the operation or to keep components from lingering in nonoptimal states. In the case of network reconfiguration, reconfiguration time is less important than network components lingering in nonoptimal states due to packets being prevented from being injected and routed in the network. The overall execution time of applications running on the system is impacted more by the latter than the former [1], [27]. Thus, the main benefit of the OSR protocol is its dramatic reduction in the period of time in which the network is prevented from injecting and forwarding packets as compared to traditional non-OSR schemes. The simulation results presented in Section 4 quantitatively confirm this to be the case.

3 Detailed Protocol Description and Proof of Deadlock Freedom

A detailed description of our proposed Overlapping Static Reconfiguration protocol and a comprehensive formal proof of its deadlock-free properties are presented in Sections 3.1 and 3.2.

3.1 Overlapping Static Reconfiguration

The following definitions and notation are used in describing the proposed network reconfiguration protocol.

Definition 1. *An* interconnection network *or, simply, a* network, *I, is represented by a strongly connected directed graph, $I = G(N, C)$. The vertices of I are the set of switches and end nodes N, whereas the edges of I are the set of communication channels (possibly virtual), C. Channels are unidirectional: Channel c_i interconnects the two nodes $src(c_i)$ and $dst(c_i) \in N$, i.e., the source and destination of the channel, respectively. A* link *is comprised of a set of channels $c_1, c_2, \dots, c_n$ such that, for all i and j, either $src(c_i) = src(c_j)$ and $dst(c_i) = dst(c_j)$ or $src(c_i) = dst(c_j)$ and $dst(c_i) = src(c_j)$. This means that a bidirectional link will consist of at least one channel in each direction. All channels in C are assumed to be part of exactly one link.*

In each network, a subset of the nodes are *end nodes*, which generate and consume network traffic (packets); all other nodes in the network are *switches*, which connect to other switches or end nodes. End nodes have one or more ingress and egress links to/from network switches and the channels of these links are called *injection* and *ejection* channels, respectively, depending on whether they inject packets into the network or eject packets out of it. Links connect a switch to other switches and end nodes through switch *input ports* and *output ports*.

Definition 2. *Assume a network, $I = G(N, C)$. A* routing function *$R : C \times N \longrightarrow \mathcal{P}(C)$, where $\mathcal{P}(C)$ is the power set of C, takes a channel c and a destination node d as parameters and returns a set of next-hop channels that can be chosen from channel c for packets whose destination is d. A routing function R is* deterministic *if, for all (c, d) combinations, $R(c, d)$ is singleton; otherwise, it is* adaptive.

This definition of a routing function allows a node to select different output channels, depending on into which input channel a packet arrives. In the following description, the old routing function that is active before network reconfiguration is referred to as R_{old} and the new routing function to which the network is to be reconfigured is referred to as R_{new}. The term *old channel* is used to signify a channel that has not yet processed a reconfiguration token

Overlapping Static Reconfiguration Protocol

- At each switch input port connected to a link or neighboring switch detected as being faulty and also at each source end node, do the following at the start of network reconfiguration:
    1) Generate a reconfiguration token associated to the last injected or buffered packet. This token indicates that no more packets routed according to R_{old} will be injected into those channels anymore. The token serves as a local synchronization point (i.e., "fence"), demarcating old packets from new packets transmitted on the channel.
- At each input port of every switch in the network, do the following:
    1) Continue using R_{old} until a packet associated to a reconfiguration token arrives at the head of the queue and is routed or until the input queue is empty and a reconfiguration token is received. The completion of this signifies that the token has been processed.
    2) Once the reconfiguration token has been processed, disable intra-switch forwarding of packets to output ports and multicast the token to all output ports supplied by R_{old} for all possible destinations from this input port.
    3) Atomically remove R_{old} routing options for the channel and atomically add R_{new} routing options for the channel, i.e., activate the new routing function for the channel. Note that although the new routing function is activated for the channel, intra-switch forwarding of packets from the input port continues to be disabled. This will persist as long as the output port supplied by R_{new} for the packet at the head of the queue has *not* yet transmitted a reconfiguration token (next bullet).
- At each output port of every switch in the network, do the following:
    1) Wait until all input ports from which the output port could receive traffic according to R_{old} have processed and transmitted a reconfiguration token to this port.
    2) Transmit the token from this output port to the input port of the neighboring switch or destination end node across the channel.
    3) Enable intra-switch forwarding of packets from all input ports which have packets at the head of the queue that are supplied this output port according to R_{new}.
- At each destination end node, do the following:
    1) Wait for the arrival of a token.
    2) Once a token is processed at all destination end nodes, reconfiguration will have completed globally across the network.

Fig. 2. Proposed Overlapping Static Reconfiguration protocol implemented at each network end node and switch.

and, for this reason, still uses R_{old}. Similarly, *new channels* are those that have processed the token and, hence, have started to use R_{new}.

The proposed *OSR* protocol is described in Fig. 2 and Fig. 3 illustrates the dynamics of the proposed protocol. The state of a switch at three different points in time during reconfiguration is shown. Each part (Figs. 3a, 3b, and 3c) shows a switch with four input ports and four output ports, where each port serves one channel. The ports are depicted with a smaller standing rectangle within the switch and input and output ports are shown separately for clarity. Lightly shaded input ports are those that have received a token but the associated packet has not yet reached the head of the queue, whereas darkly shaded ones are those that have processed the token. Likewise, darkly shaded output ports are those that have transmitted a token. The output port number to which an input port is allowed to transmit packets for all possible destinations according to the currently active routing function is shown to the right of each input port. For ports that have processed a token, this will be the new routing function. For ports that have not yet processed a token, this will be the old routing function. The arrows from input to output ports indicate to where packets might be forwarded at this stage of the reconfiguration process. Input ports that have already processed the token and activated the new routing function are allowed to forward packets only to output ports that have already transmitted a token. Conversely, input ports that have not

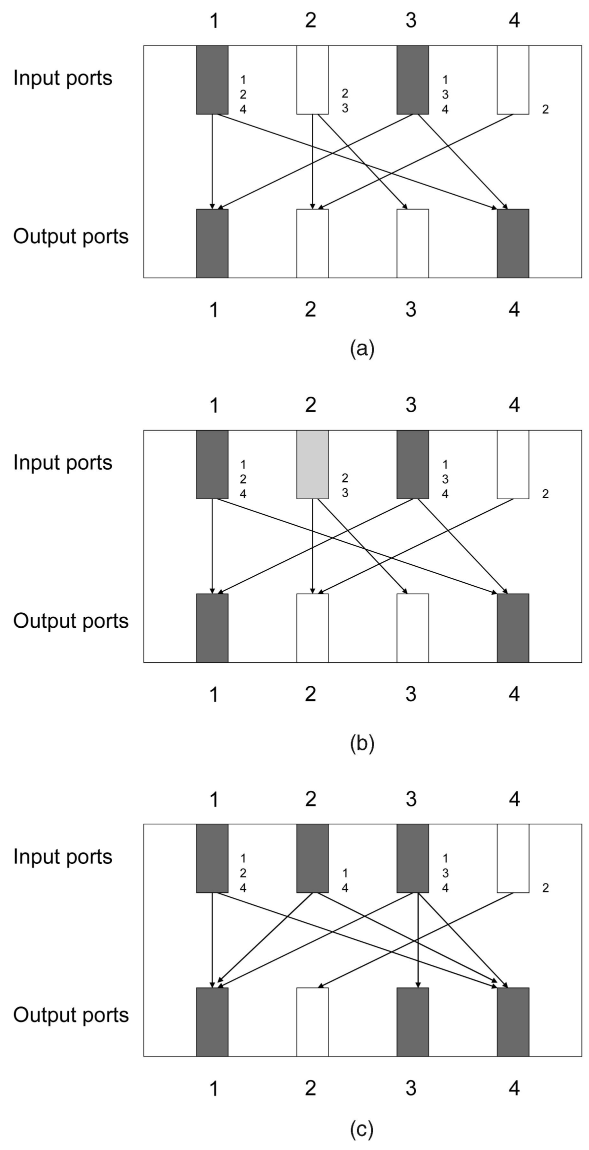


Fig. 3. Illustration of the dynamics of the OSR protocol over time within one switch.

yet processed a token may forward packets only to output ports that have not yet transmitted a token.

In Fig. 3a, a token has been processed on input ports 1 and 3 and tokens have been transmitted through output ports 1 and 4. Even if input port 1 is ready to forward packets to output port 2 and input port 3 is ready to forward packets to output port 3 according to the new routing function, they are not allowed to do so as those output ports have not yet transmitted tokens themselves and can still receive more old packets from other input ports. Fig. 3b illustrates a token having arrived on input port 2 for the last old packet but that packet not yet arriving at the head of the queue. Fig. 3c illustrates what must have happened in the switch as a consequence of routing the last old packet and processing the token at input port 2. Output port 3 has transmitted a token as all input ports that were able to forward old packets to it have already processed a token. Input port 3 is no longer disabled from forwarding packets to output port 3 now that output port 3 has transmitted a token. However, input port 1 remains disabled from forwarding packets to output port 2 as that output port may still receive old packets from input port 4. Finally, input port 2 has activated the new routing function for its packets. This is illustrated in the figure by the change in output ports supplied by the routing function shown to the right of the input port.

3.2 Proof of Deadlock Freedom of OSR

The formal proof of correctness of the OSR is based on theory developed in [32], which, unlike previous theories (e.g., [20], [22], [23], [37]), encompasses the deadlock freedom of networks that undergo reconfiguration. That theory is based on describing network states as sets of *CND-tuples*,[4] where each tuple describes the occupancy of a network channel and possible dependency to some other channel for a packet destined for a particular end node [32]. For routing function R, $R(c, d)$ is the set of channels that a packet header residing in channel c can take for a destination given by d. With this, CND-tuple $\langle c, c', d\rangle$ signifies that channel c is occupied by part of a packet destined for end node d, whose header was routed to channel c'. A CND-tuple $\langle c, c', d\rangle$ is *legal* for R only if $c' \in R(c, d)$. Alternatively, CND-tuple $\langle c, c, d\rangle$ signifies that channel c is occupied by part of a packet destined for end node d whose header has not yet been forwarded further to another channel.

According to [32], the runtime state during reconfiguration is represented as a *CND-relation*, which is a set of CND-tuples. The set of all possible runtime states at a given stage of a reconfiguration process is represented by a superset of CND-tuples. That set is legal if it is the union of all of the CND-relations that represent all legal runtime states at a given point in the reconfiguration process. If Θ is a set of legal CND-tuples that represents a set of legal runtime states, any one of these given runtime states is represented by a CND-relation formed from a consistent subset of Θ.

Before developing the deadlock freedom proof for OSR, we give some additional definitions and the main results from [32], upon which the proof is based. To support readability of the proof, we have simplified the theorem into a less powerful one. The original theorem supports adaptive routing functions with cyclic dependency graphs, where deadlock freedom is ensured by the existence of a set of cycle-free escape resources. Since our reconfiguration protocol requires cycle-free channel dependency graphs, we do not need to distinguish the escape resources from the rest of the resources. Our reformulations of the theorem and associated corollary from [32] are thus based on the assumption that all channels are escape channels. The interested reader is referred to [32] for further explanation.

Definition 3. *For a network I and routing function R, there exists a* dependency *from channel c to c' if and only if $c' = R(c, d)$ for some destination d. That is, packets destined for d may use c' immediately after c.*

Definition 4. *For any prevailing routing function R and set of CND-tuples Θ, there is a* channel dependency *from channels c to c'' if there exists a channel c' and a destination d such that $c' = Header(c)$ in some consistent subset θ of Θ*

4. Several CND-tuples are needed to represent a blocked wormhole packet that spans multiple channels, whereas a single tuple is needed to represent a blocked virtual cut-through packet, given that channels are able to store entire packets.

and $c'' \in R(c', d)$, where $Header(c)$ refers to the channel containing the header of the packet that occupies channel c. Let $DEP(R, \Theta)$ be the set of channel dependencies corresponding to routing function R and a set of legal CND-tuples Θ. That is, $DEP(R, \Theta) = \{< c, c'' > \mid$ there exists a channel c' and destination d for which $c' = Header(c)$ for some consistent subset θ of Θ and $c'' \in R(c', d)\}$.

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

In developing the proof, we use the following notation to simplify our description of network state at any point during reconfiguration. The set of legal CND-tuples for routing function R_{old} is given by Θ_{old}, i.e., Θ_{old} is the union of all of the sets of CND-tuples that describe legal runtime states of the network according to R_{old}. Likewise, the set of legal CND-tuples for routing function R_{new} is given by Θ_{new}. A CND-relation that represents a deadlocked packet configuration is called a *deadlock CND-relation*.

The following theorem and corollary are developed and proven in [32].

Theorem 1. *For any legal set of CND-tuples Θ and prevailing routing function R, any consistent CND-relation $\theta \subseteq \Theta$ is not a deadlock CND-relation under R if $DEP(R, \Theta)$ is acyclic.*

Corollary 1.1. *For any stage of reconfiguration process RP with a corresponding prevailing routing function R and a set of legal CND-tuples Θ, there is no deadlocked runtime state at that stage of reconfiguration if $DEP(R, \Theta)$ is acyclic.*

With the above understanding given by the theory in [32], we are now able to develop our deadlock freedom proof for OSR. We start by stating a working assumption and then proceed by presenting two simple lemmas on the dynamics of our proposed reconfiguration protocol. Based on those two lemmas, we develop two other lemmas that describe deadlock-free properties for the sequence of states that the proposed reconfiguration process undergoes. The last of these lemmas, i.e., Lemma 4, together with Corollary 1.1 proves deadlock freedom of the OSR protocol.

Assumption 1. *Both R_{old} and R_{new} independently are assumed to have acyclic channel dependency graphs by design. That is, $DEP(R_{old}, \Theta_{old})$ and $DEP(R_{new}, \Theta_{new})$ are free of cycles.*

Lemma 1. *OSR ensures that all packets are routed either solely according to R_{old} or solely according to R_{new}. That is, Invariant 1 is maintained.*

Proof. We prove by contradiction. Consider a packet that experiences routing according to both routing functions. On its path from the source to the destination, there will be two consecutive switches, S_1 and S_2, where this packet is routed according to the two routing functions. There are two possible cases:

Case 1: The packet is first routed according to R_{old} in S_1 and then routed according to R_{new} in S_2. This packet must arrive at switch S_2 on an input channel that has already processed a token. In S_1, it was routed according to R_{old}, so it arrived there on an input channel *before* a token was processed on that channel. Therefore, if S_2 received a token before the packet, S_1 must have sent a token from the output channel going to S_2 before S_1 had processed a token on one of the input channels from which this output channel could receive traffic according to R_{old}. According to the procedure for output channels in the reconfiguration protocol, this cannot happen.

Case 2: The packet is first routed according to R_{new} in S_1 and then routed according to R_{old} in S_2. This packet must arrive at S_2 on an input channel that has not yet processed a token. In S_1, it was routed according to R_{new}, so it arrived there after a token was processed. Therefore, S_1 must have forwarded packets from an input channel that had processed a token to an output channel that had not yet transmitted a token. According to the procedure for output channels in the reconfiguration protocol, this cannot happen. □

Lemma 2. *The order in which packets and tokens are processed by channels for the OSR protocol is the following: first only old packets (if any), then a reconfiguration token, and, finally, only new packets (if any).*

Proof. Again, we prove by contradiction. Assume a channel in a network reconfigured according to the proposed protocol for which Lemma 2 does not hold. This would require either a new packet to traverse the channel before a reconfiguration token is processed or an old packet to traverse the channel after a token is processed. This results in the new packet being routed according to R_{old} or the old packet being routed according to R_{new} at the next switch. Both cases contradict Lemma 1. □

Lemma 3. *Let Θ_T be the set of legal CND-tuples at some point in time T in the OSR protocol and let R_T be the prevailing routing function at that time. Then, all dependencies in $DEP(R_T, \Theta_T)$ that start from an old channel are also in $DEP(R_{old}, \Theta_{old})$ and all dependencies in $DEP(R_T, \Theta_T)$ that start from a new channel are also in $DEP(R_{new}, \Theta_{new})$.*

Proof. We prove by induction. First consider the channel dependencies caused by packets routed according to the old routing function. Assume any channel dependency from c to c'' in $DEP(R_T, \Theta_T)$, where c is an old channel. This would require a CND-tuple $\langle c', c', d\rangle \in \Theta_T$ such that $c' = Header(c)$ and $R_T(d, c') = c''$. Since c is an old channel, we know that the packet with header in c' was routed according to R_{old} when it was in c; thus, it is an old packet. According to Lemma 1, we know that it must be routed according to R_{old} all the way up to channel c' and also at channel c'. Thus, the prevailing routing function for channel c' is R_{old}. But, this means that $c'' \in R_T(c', d) = R_{old}(c', d)$; thus, the dependency from c to c'' must be in $DEP(R_{old}, \Theta_{old})$ as well.

The proof for channel dependencies caused by packets routed according to the new routing function is similar. Assume any dependency from c to c'' in $DEP(R_T, \Theta_T)$, where c is a new channel. This would require a CND-tuple $\langle c', c', d\rangle \in \Theta_T$ such that $c' = Header(c)$ and $R_T(d, c') = c''$. Since c is a new channel, we know that the packet with header in c' was routed according to R_{new} when it was in c; thus, it is a new packet. According to Lemma 1, we know that it must be routed according to R_{new} all the way up to and including channel c'; thus, the prevailing routing function for channel c' is R_{new}. But, this means that $c'' \in R_T(c', d) = R_{new}(c', d)$; thus, the dependency from c to c'' must be in $DEP(R_{new}, \Theta_{new})$ as well. □

A simple observation from Lemma 3 is that any dependencies in $DEP(R_T, \Theta_T)$ are in $DEP(R_{old}, \Theta_{old})$, in $DEP(R_{new}, \Theta_{new})$, or in both.

Lemma 4. *$DEP(R_T, \Theta_T)$ is acyclic at any point in time T, where R_T is the prevailing routing function at that time for a network undergoing OSR.*

Proof. We again prove by contradiction. Assume that $DEP(R_T, \Theta_T)$ contains a cycle. We begin by trivially showing that there are no cycles of channel dependencies containing only new channels or only old channels. According to Lemma 3, all channel dependencies starting in new channels are part of $DEP(R_{new}, \Theta_{new})$. Thus, such a cycle would imply a cycle of dependencies also in $DEP(R_{new}, \Theta_{new})$, which contradicts Assumption 1. With the exact same reasoning, there cannot be a cycle of channel dependencies using old channels only as $DEP(R_{old}, \Theta_{old})$ also is cycle free according to Assumption 1. Therefore, any cycle in $DEP(R_T, \Theta_T)$ must contain both old channels and new channels and, in particular, a dependency from an old channel to a new channel. This would require a CND-tuple $\langle c, c', d\rangle \in \Theta_T$ such that c is an old channel and $R_T(d, c')$ is a new channel.

Since c is an old channel, we know that the packet with header in c' was routed according to R_{old} when it was in c. According to Lemma 1, we know that it must also be routed according to R_{old} in channel c'; thus, the prevailing routing function for channel c' is R_{old}. Since $R_T(d, c') = R_{old}(d, c')$, we know that channel $R_T(d, c')$ will receive an old packet; thus, it cannot yet have received a token according to Lemma 2. Therefore, it must be an old channel, which leads to a contradiction. □

4 Evaluation

In this section, we evaluate the proposed reconfiguration protocol. First, we describe the methodology, traffic patterns, and topologies that are used in the evaluation. Then, we describe the metrics used and the implementation details of the various alternative schemes used to compare against the proposed protocol. Finally, we present the results and analysis.

4.1 Evaluation Methodology

In conducting the evaluation, we developed a detailed simulator that allows us to model the network at the cycle-accurate level. The simulator has been used and validated in different recent works [38], [39]. The simulator models network end nodes, switches, and links.

Network end nodes are attached to switches through regular links and generate packets at a given rate, whether or not the traffic is accepted by the network. It is reasonable to think that, in such situations, the generation of packets slows down as there are dependencies among packet sources in applications. In order to model this behavior, the end nodes are modeled as having finite queues, each having a depth of 64 packets. If a queue overflows, packets are simply dropped at the source, which is measured in the evaluation.

Links are assumed to be bit serial, with a data rate of 2.5 gigabits per second (Gbps). We model an 8 bytes/10 bytes encoding [40]; thus, a new byte is transmitted every 4 ns. The physical medium of the link is modeled as 15 m copper cables, with a propagation delay of 5 ns/m. Switches are modeled as having virtual channels and a nonmultiplexed crossbar, thus allowing parallel forwarding of packets from different virtual channels from the same input port (i.e., an input speedup of 2). Input and output buffering is assumed and, in our plotted experiments, they both have a depth of 1 Kbyte. This size has been chosen based on the knowledge of buffer sizes for some existing InfiniBand components. We have, however, also conducted experiments with 512 byte buffers and the impact on the results from halving the buffer size was negligible. Packets (both data and control) have a maximum size of 58 bytes, divided into 32 bytes of payload and 20 bytes of header. Flow control is implemented with special control packets of only six bytes. Flow control and other control packets (mentioned in the following) are taken into account in the simulations. Data packets are routed at each switch by accessing a routing table. Control packets are routed using source (directed) routing, e.g., as in [11], [40]. Routing tables contain the output port to be used at the switch for each possible destination for packets at given input ports. The routing delay at each switch is assumed to be 100 ns. This time includes the lookup table, crossbar allocation, and connection setup through the switch.

All reconfiguration schemes assume the use of two data virtual channels and one control virtual channel multiplexed onto the physical links. The data virtual channels are used by data packets, whereas the control virtual channel is used only by control packets for monitoring, notifying, and issuing commands. The reason for this is that many schemes, including the OSR protocol, require notifications and routing table updates through a separate channel. Also, one scheme used in the comparisons (i.e., the DS) requires the use of two virtual channels. Credit-based flow control and virtual cut-through switching are used to regulate the virtual channels: A packet is transmitted over the link only if there is enough buffer space to store the entire packet at the receiving end of a scheduled virtual channel.

4.1.1 Reconfiguration Schemes Used in the Comparisons

We evaluate the OSR protocol and the traditional non-OSR process. In order to see how close our improvements to SR bring us to achieving the performance efficiency of previously proposed dynamic reconfiguration schemes, we include comparisons against one of the best known dynamic reconfiguration processes—the Double Scheme (DS) [31]. In all cases, there exists a network manager (NM) located on an arbitrary end node that monitors the network for changes and orchestrates the reconfiguration. This is achieved by sending control packets to all switches through the control virtual channel. Upon the receipt of a "configuration" control packet, each switch informs the NM about any changes in its attached links in order to discover the new network topology, as in [31].

We model two different implementations of OSR which differ in the order of events triggered once a topology change is detected and the new routing tables are calculated. In OSR Packet Drop Aware (OSR-PDA), tables are sent in parallel with a "reconfiguration" control packet that is broadcast by the NM to all end nodes and switches, signaling nodes to generate reconfiguration tokens. In OSR Latency Aware (OSR-LA), the NM first distributes and stores the new routing tables into a secondary location in the end nodes and switches before it broadcasts the reconfiguration control packet. The former minimizes the

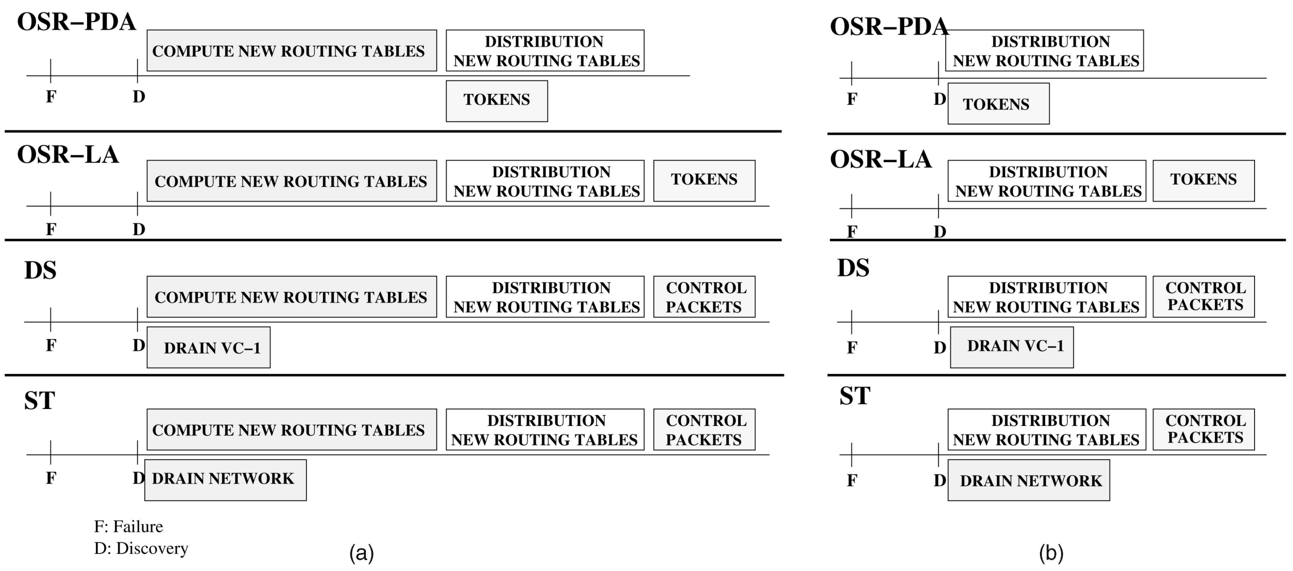


Fig. 4. Reconfiguration schemes evaluated and events that distinguish them.

time that R_{old} is in use, thus minimizing the number of packets that could be dropped due to routing across failed links/switches, but at the expense of possibly longer average packet latency. The latter potentially reduces the average packet latency but at the expense of possibly longer reconfiguration time. This trade-off in the way OSR is implemented is explored further in the following.

In DS, once a topology change is detected, the NM computes the new routing tables and, afterward, it distributes them to all end nodes and switches. At the same time, it sends a "virtual channel drain" control packet to all switches and end nodes, instructing them to drain one of the two data virtual channels (i.e., VC-1). Packets residing in that data virtual channel are transported through the other data virtual channel at each switch. Drainage thus occurs in parallel across all of the network switches. Once the designated data virtual channel is empty network-wide,[5] the NM is notified. The NM then signals the end nodes and switches (via control packets) to start using both data virtual channels with the new routing function. The drained virtual channel is used as the escape path for any old packets in the other data virtual channel. Reconfiguration completes once all end nodes and switches are able to use both data virtual channels again.

In SR, once the topology change is detected, the NM broadcasts a "network drain" control packet to all end nodes, instructing them to halt packet injection. At the same time, the NM starts computing the new routing tables. Once the tables are computed and distributed to all nodes and after the network is completely drained of all data packets,[6] the NM sends control packets, instructing the end nodes to resume packet injection. Reconfiguration completes once all nodes can inject packets again.

5. A distributed protocol is used for detecting that the designated data virtual channel is empty network-wide [31]. Every switch sends a notification to the NM once the buffers assigned to the data virtual channel are empty and there are no data packets injected through the output links sent by the switch.

6. In our simulations, it takes 26.4 μs, on average, for the baseline network to drain itself completely of data packets under near-saturation network loads.

Fig. 4a summarizes the events that distinguish each of the reconfiguration schemes used in this evaluation. As can be noticed, the change detection and routing table computation must be performed for all of the schemes. The latter operation has been shown to dominate reconfiguration time, depending on the routing algorithm used and the topology [1], [41]. Its impact on network performance and the number of dropped packets is the same, regardless of the reconfiguration scheme, as the old routing function remains in place for all schemes until a change is detected and the new routing function is computed. As this paper focuses on the reconfiguration process, detection/notification is accounted for in the simulations, but the routing table's computation time is assumed to be zero, which would be the case if tables were precomputed before the reconfiguration event (i.e., for user-directed or planned reconfiguration). Fig. 4b shows the behavior modeled for each of the simulated reconfiguration schemes.

4.1.2 *Traffic Patterns, Baseline Network Topology, and Routing Algorithm*

For each simulation run, we assume a constant packet generation rate for all end nodes. To control the effect of start-up instabilities, we ran an experiment with maximum traffic injection on a 10×10 torus network and measured the development in throughput over time. The results are plotted in Fig. 5 and they show that the network stabilizes well before 10,000 packets have been sent. In order to ensure that start-up instabilities do not affect our evaluation results, reconfiguration is not invoked until 80,000 packets have been transmitted.

Three different synthetic traffic patterns are modeled: uniform, bit reversal, and hot spot. For uniform traffic, each source sends packets to all destinations with equal probability. For bit reversal, each source sends traffic only to one end node computed by reversing all bits of the source ID. For hot spot, 10 percent of the sources selected randomly inject traffic to the same destination, also selected randomly, and the rest of the end nodes inject traffic to all other destinations with equal probability.

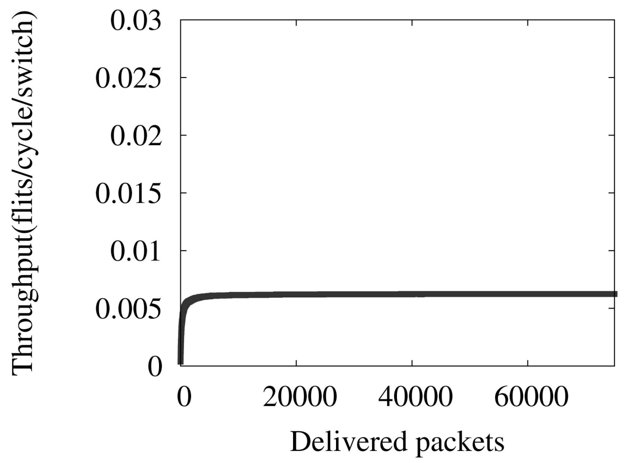


Fig. 5. Stabilization of the network as a function of time. Throughput is measured as a function of all of the messages received until a given point.

The reconfiguration schemes are evaluated on an 8×8 torus baseline network. Two end nodes are attached to each switch; thus, 128 end nodes exist in the system. The schemes are evaluated on how well they handle link failure. Before a random link failure occurs, the up*/down* [26] routing algorithm—which is deadlock-free by design—is used as the old routing function, with the root at node (0, 0). After the link failure, the up*/down* algorithm is recomputed, but, this time, with the root node set at node (3, 3) for the new routing function.

4.1.3 Performance Metrics

One of the metrics used to measure the performance of the reconfiguration scheme is average packet latency as it varies over time. Packet latency is the time that a packet experiences from the point at which it is generated by its source until the point at which it is delivered to its destination. Traditionally, the average packet latency is plotted in relation to when packets are delivered, as illustrated in Fig. 6a, as opposed to when packets are generated, as illustrated in Fig. 6b. The plot in Fig. 6a indicates that packets that **arrived** at time x exhibit an average latency of y cycles. The weakness of this approach is that the effect of reconfiguration on packet latency can be averaged out. Latency measured immediately after reconfiguration starts will be heavily influenced by packets that have completed most of their journey through the network that has not yet undergone reconfiguration. Likewise, immediately after reconfiguration, the results will be partly influenced by packets that have not experienced any reconfiguration effects and partly by packets that have been delayed due to reconfiguration.

In this paper, we provide a different more accurate view of packet latency. As shown in Fig. 6b, we plot the average packet latency in relation to the time at which packets are generated. With this, packets **generated** at time x are shown to exhibit an average latency of y. By plotting the results this way, the impact of reconfiguration on packet latency will be more clearly exposed and latency comparisons can be made between the schemes both during and after network reconfiguration. In the following latency plots, the average packet latency is broken down into the following components: queue latency (QL), network latency (NL), and maximum token latency (MTL). All of these components sum to equal the total latency experienced by packets on average. QL is the time that packets spend at the source end node waiting to be injected. NL is the time that packets spend within the network. MTL is the maximum time that packets are blocked at the head of a queue waiting for its supplied switch output port to transmit a reconfiguration token. This time is relevant only for OSR.

The other metrics used in the evaluation are fairly straightforward. The total reconfiguration time for the schemes is measured, which is the time elapsed from the detection of the topological change to the end of the reconfiguration process. Network throughput is also measured, which is the average number of bytes per unit of time which are delivered by switches plotted for each virtual channel (two data and one control). Likewise, network injection bandwidth is measured for each virtual channel, which is the average number of bytes per unit of time which is injected at network sources. Finally, the number of dropped packets due to reconfiguration is also measured for each scheme.

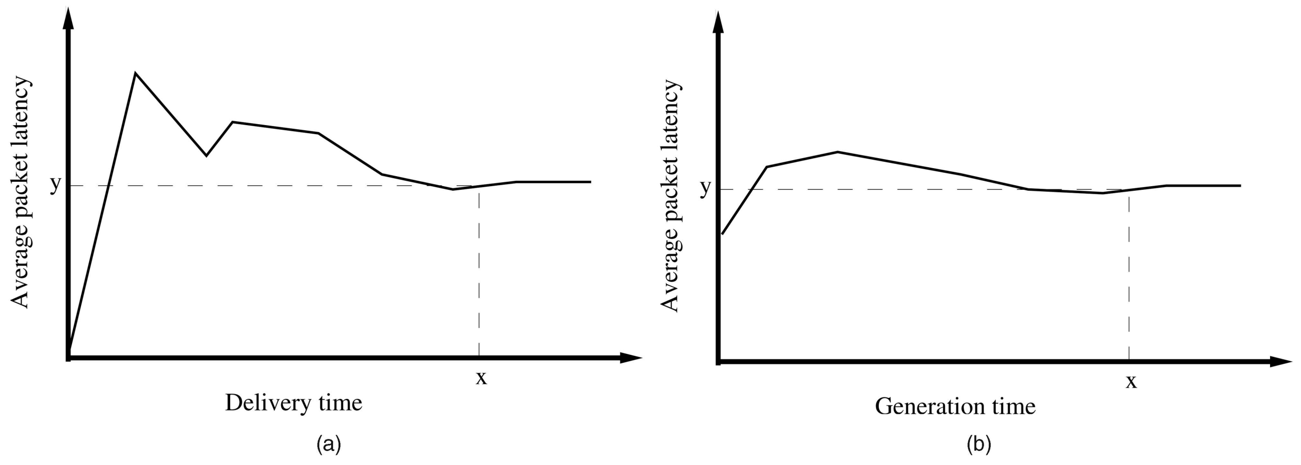


Fig. 6. Average packet latency plotted in relation to (a) when packets are delivered (i.e., delivery time) versus (b) when packets are generated (i.e., generation time).

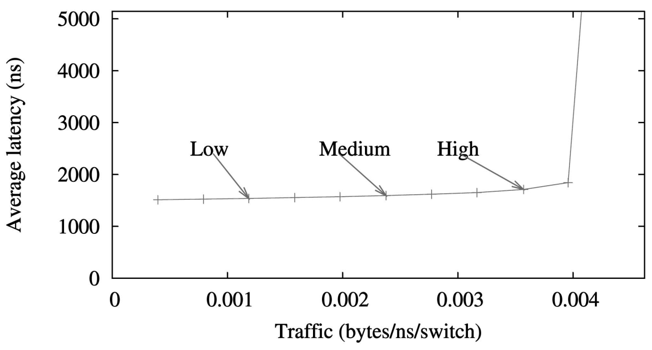


Fig. 7. Average packet latency versus data throughput for up*/down* routing on 8 $\times$ 8 torus.

4.2 Results and Analysis

The OSR protocol is evaluated and compared against the other reconfiguration schemes across various network load rates and traffic patterns. Fig. 7 shows the performance obtained in an 8×8 torus network under uniform traffic when no reconfiguration process is triggered. The figure indicates the three network load rates that are used in the simulations and their intensities relative to the network saturation point. In what follows, the three ranges of load rates are referred to as LOW, MEDIUM, and HIGH, respectively.

Fig. 8 shows the average packet latency plotted against the generation time for the different reconfiguration schemes for uniform traffic under LOW, MEDIUM, and HIGH traffic load rates. For all of the schemes, the start and end times for reconfiguration are indicated with vertical lines. For OSR-LA, an extra vertical line is added which indicates when the routing tables have been uploaded and the reconfiguration tokens are being injected. The first observation is that the reconfiguration times for OSR (both OSR-LA and OSR-PDA) and DS are 30 percent to 50 percent less than SR. The reason for this is that, in both schemes, old and new packets can be transported simultaneously in the network, whereas, in SR, the network must be completely drained before any new packets can be injected. On average, OSR-PDA and OSR-LA reduce the SR time by approximately 47.5 percent and 33.5 percent, respectively, under LOW, MEDIUM, and HIGH network loads. The reconfiguration times for OSR-PDA and DS are practically the same. The slightly increased reconfiguration times of OSR-LA, relative to OSR-PDA, are due to the serialization of table distribution and token distribution. Table 1 summarizes the reconfiguration times and percentages of improvement for each under the simulated network loads.

Another important result based on the figures is that OSR completely removes the need to queue packets at the end nodes during reconfiguration, as done by SR. Indeed, the average QL for OSR is zero for all simulated network loads. This means that, with respect to QL, OSR performs as well as the DS dynamic reconfiguration. In contrast, SR experiences very high queuing latency as the injection of traffic is completely halted. In Fig. 8, we can observe that SR packets generated at the start of the reconfiguration period experience QLs of up to 70 μs.

With regard to latency within the network, we observe that the average packet latency of OSR-OPA due to the reconfiguration process is significantly higher than that of OSR-LA. Indeed, OSR-LA experiences a very small packet latency increase during network reconfiguration, i.e., only in the short period of time in which there are tokens in the network. The reason for this difference is that, in OSR-PDA, packets spend a lot of time waiting for the new routing table to arrive at switches. In the plots, this is captured as an increased MTL, even though the average token latency is close to zero, since very few packets wait for a token. Almost independent of the network load, there are some packets in OSR-PDA that are blocked for a significant amount of time (40 μs). These packets block the packets coming from behind, thus introducing contention. This effect rapidly spreads and, thus, packets in the network experience higher latencies.

Fig. 8 also shows that SR has an increased NL relative to all other schemes.[7] The average packet latency is nearly two orders of magnitude higher than OSR-LA and is factors higher than OSR-PDA. This situation persists for a prolonged period of time, even after reconfiguration has completed. Latency is increased because, when traffic is resumed, all of the end nodes inject their queued packets at the maximum injection rate. This forces the network to experience temporary congestion.

Let us consider injected traffic, as shown in Fig. 9, covering the entire reconfiguration period for HIGH traffic. For LOW and MEDIUM traffic, similar results were obtained. For OSR-PDA, when the failure is detected, the traffic in both data virtual channels is increased, as shown by a traffic spike. This is due to the injection of the tokens from each end node at roughly the same time. After that, traffic injection is not blocked in any data virtual channel. With regard to the control virtual channels, the plots show that they are busy sending new routing tables to every switch. For OSR-LA, similar behavior is observed; however, the traffic spike experienced through data virtual channels due to token injection occurs at the end of the reconfiguration period since OSR-LA first distributes the routing tables and then generates the tokens. For the DS, there is a spike in injected traffic in the control virtual channel. This is due to control packets sent through the virtual channel in order to start the drainage process for one of the data virtual channels (VC-1). Indeed, after this traffic spike, the data virtual channel is no longer used until reconfiguration completes. The other data virtual channel, however, doubles its traffic injection since it now has to cope with all the injected data traffic. The control virtual channel is used throughout the reconfiguration period for distributing routing tables and sending other notifications. Finally, for SR, we observed that the injection of data packets is halted during almost the entire network reconfiguration period. At the end of reconfiguration, there is a data packet injection spike due to most packets being queued at injection end nodes, as noted earlier.

Let us now consider delivered traffic, shown in Fig. 10. Again, only results for HIGH traffic are shown as LOW and MEDIUM traffic give similar results. For OSR-PDA, the delivered data traffic reaches zero during almost the entire reconfiguration period. The network is quickly drained of old packets as they reach their destination or a failed link, but new packets remain blocked in the network for a while,

7. Although, in SR, traffic is stopped during reconfiguration, the plots show traffic within the start and end of reconfiguration lines. This is because latency is plotted at generation time and not at delivery time.

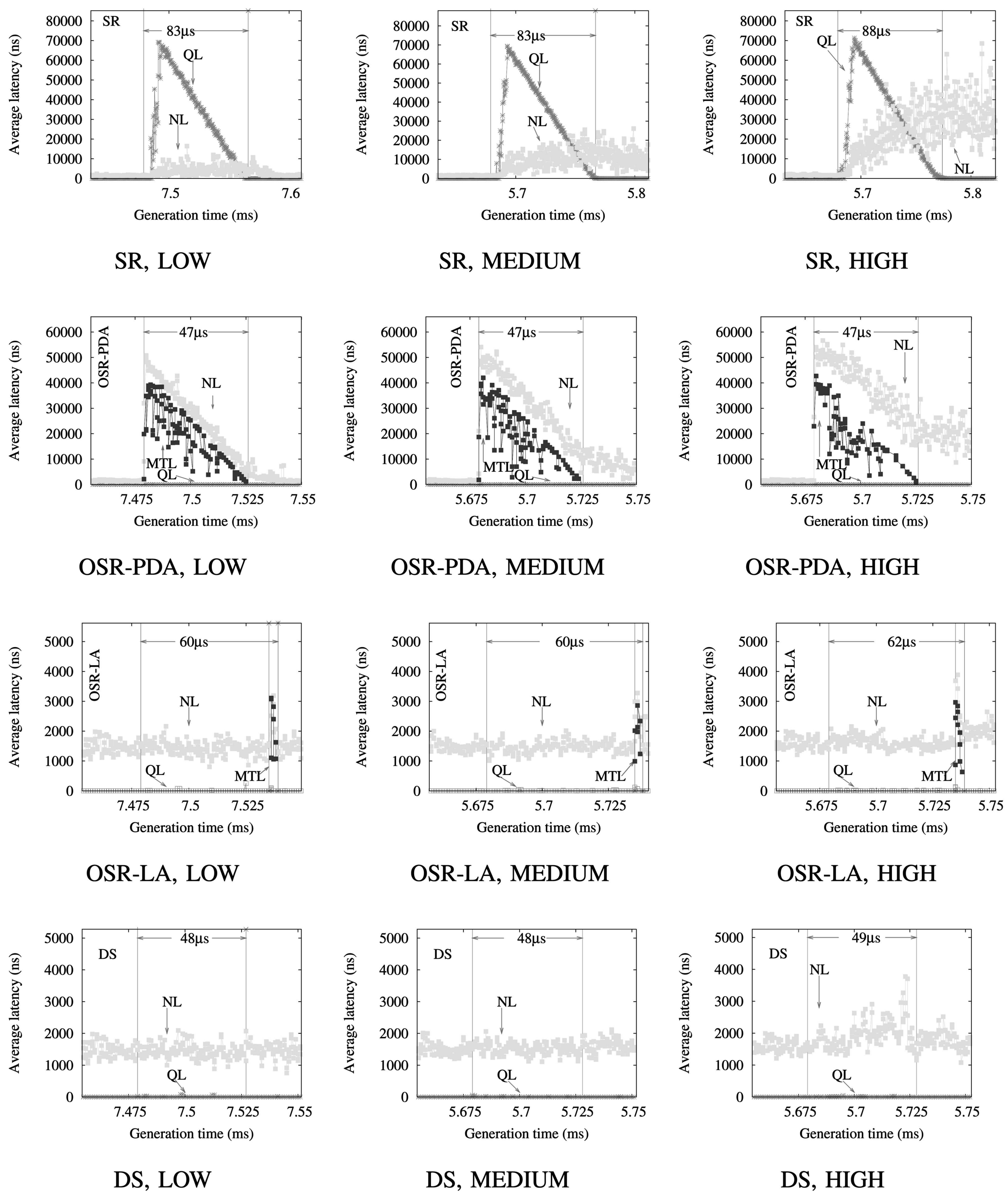


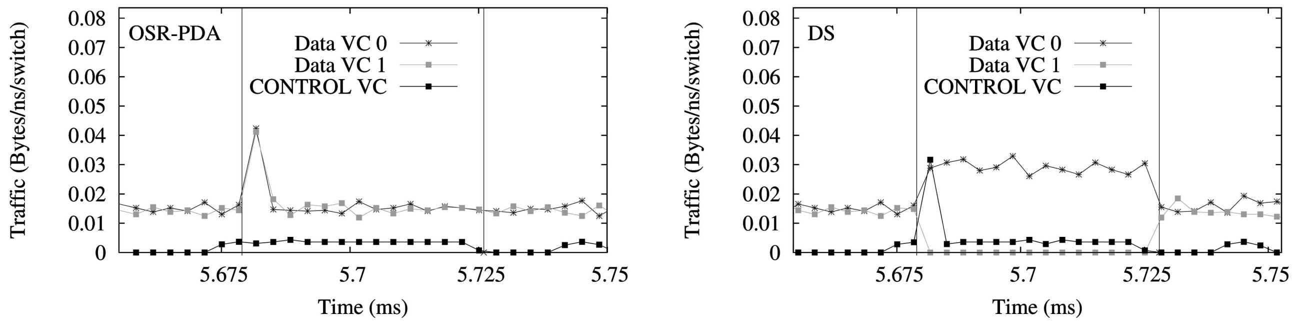
Fig. 8. Average packet latency at various generation times for uniform traffic, where various components of latency are broken down as follows: QL, NL, and MTL.

waiting for the routing tables to be distributed. At the end of reconfiguration, there is a spike of delivered traffic corresponding to the data temporally queued within the network. However, for OSR-LA, delivered traffic keeps constant during almost the entire reconfiguration process. At the end of reconfiguration, delivered data traffic is reduced during a small interval over which old packets are drained and tokens are propagated through the network.

For SR, the delivered traffic reaches zero during almost the entire reconfiguration period and, at the end, a spike of delivered traffic is observed, correlated with the spike exhibited for injection. For the DS, the delivered traffic for one data virtual channel drops to zero, whereas the other data virtual channel doubles its delivered rate. This observation for the DS is interesting for the reason that

TABLE 1
Reconfiguration Times and Percentages of Improvement for the Different Reconfiguration Schemes and Network Loads under Uniform Traffic

	Reconfiguration Time				% Improvement of Reconfiguration Time				
Traffic load	OSR-PDA	OSR-LA	DS	SR	OSR-PDA vs SR	OSR-LA vs SR	DS vs SR	OSR-PDA vs DS	DS vs OSR-LA
LOW	47.51μs	60.77μs	48.52μs	87.70μs	45.83%	30.70%	44.68%	2.08%	20.17%
MEDIUM	47.46μs	59.54μs	49.11μs	88.52μs	46.38%	32.74%	44.52%	3.36%	17.51%
HIGH	47.49μs	60.29μs	49.29μs	95.70μs	50.37%	37.00%	48.50%	3.65%	18.25%



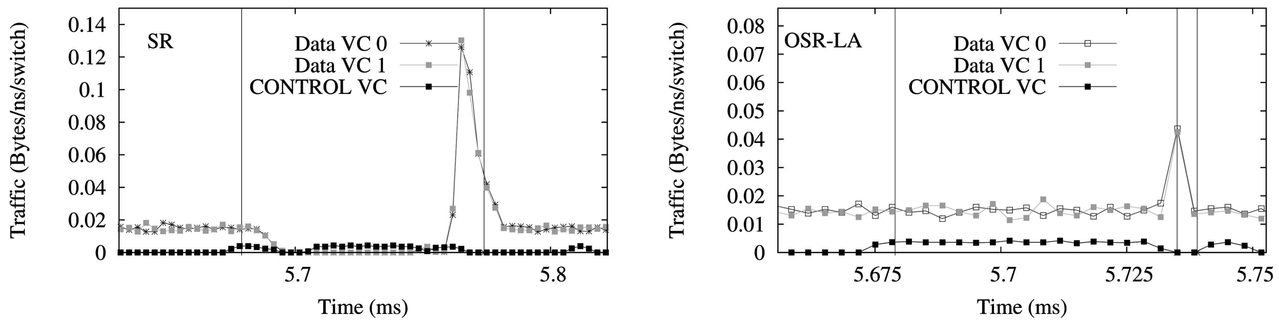
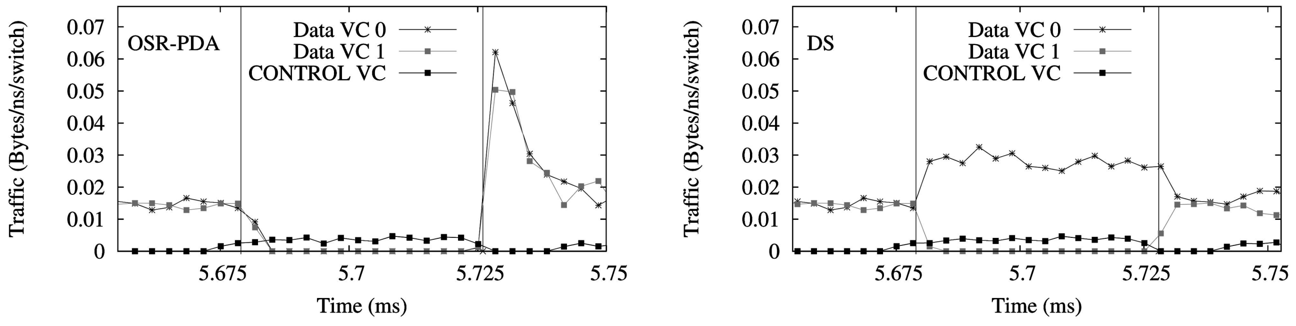


Fig. 9. Injected traffic per virtual channel for uniform traffic under a HIGH network load rate.



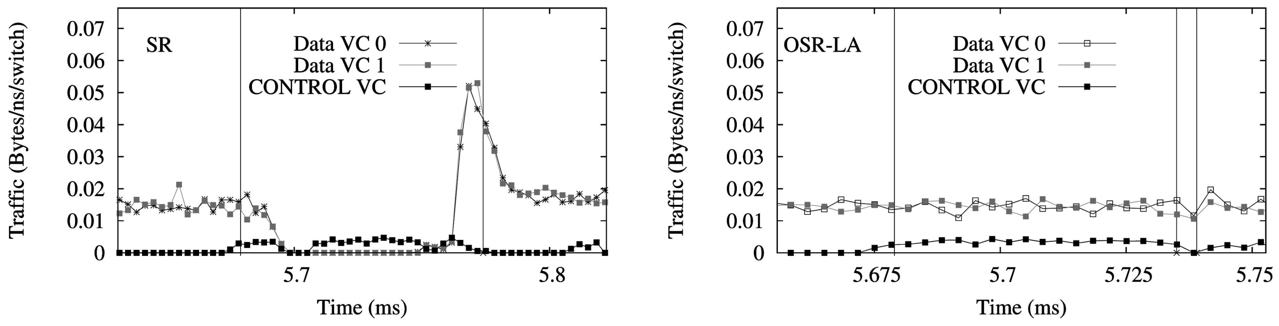


Fig. 10. Delivered traffic per virtual channel for uniform traffic under a HIGH network load rate.

TABLE 2
Dropped Traffic for the Different Schemes and Network Loads under Uniform Traffic

	Dropped packets due to the failed link				Dropped packets at source end nodes			
Traffic load	OSR-PDA	OSR-LA	DS	SR	OSR-PDA	OSR-LA	DS	SR
LOW	0	13	10	0	0	0	0	0
MEDIUM	0	23	12	5	0	0	0	0
HIGH	0	25	24	7	0	0	0	231

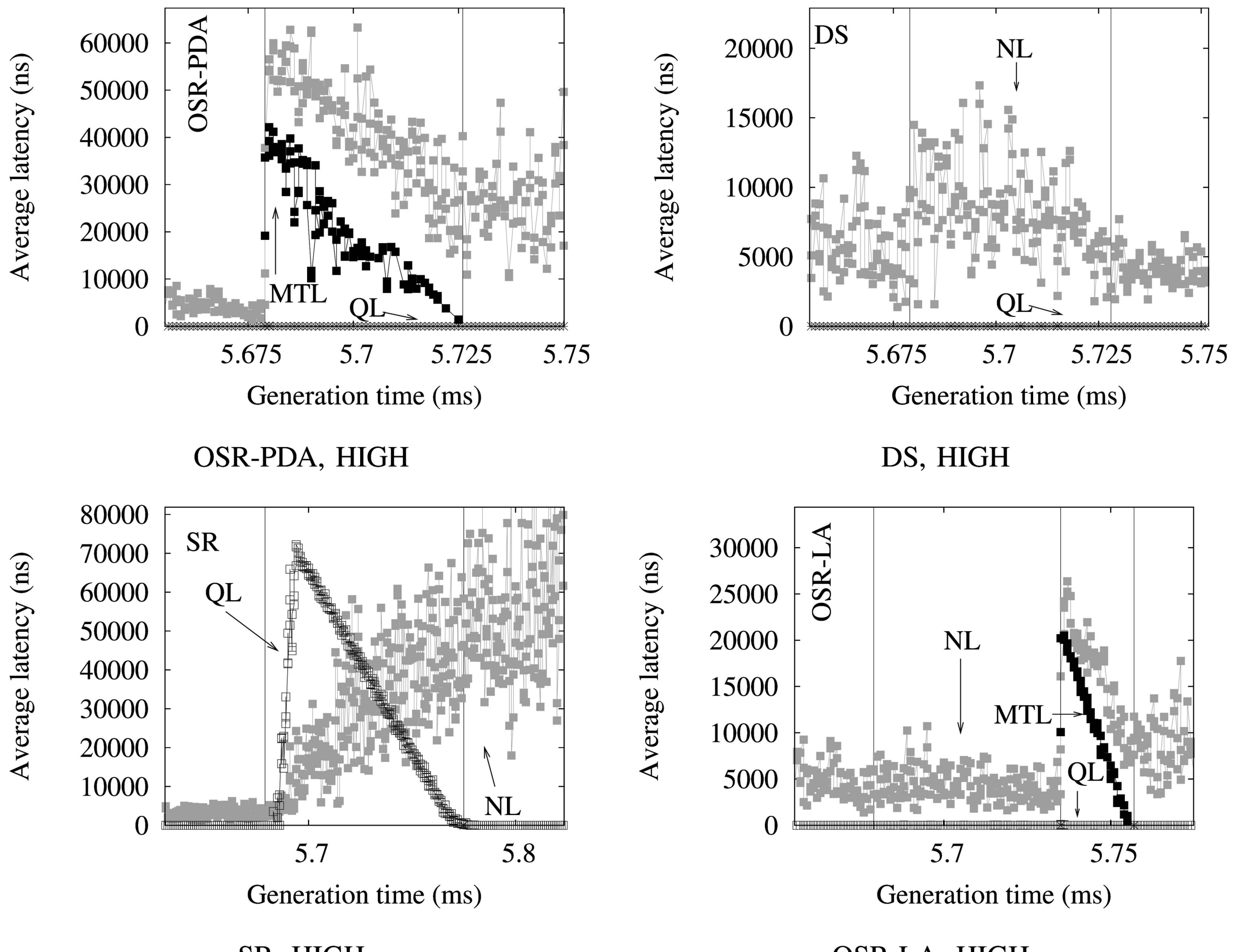


Fig. 11. Average packet latency at the generation time under hot-spot traffic (Q.L.: Queue Latency, N.L. : Network Latency, MTL: Maximum Token Latency).

virtual channels are provided in many technologies to support different service classes. It is clear that the only reconfiguration scheme that sustains throughput over all virtual channels during reconfiguration (thus not harming any of the service classes) is OSR-LA. In this respect, it actually outperforms the best known dynamic reconfiguration technique, namely, the DS.

We now turn our attention to packets dropped due to the reconfiguration scheme. Table 2 shows the number of dropped packets by each reconfiguration scheme. The table distinguishes between dropped packets through the failed link and dropped packets at end nodes due to filling of the source queue. As shown, OSR-PDA causes the fewest number of dropped packets, i.e., zero. The reconfiguration control packet that is broadcast by the NM to start the reconfiguration process is sent over the control virtual channel that has a higher priority than the data virtual channels to allow it to arrive at all destinations quickly. The end nodes can therefore send the reconfiguration token before too many packets get queued. Packets that get queued after the reconfiguration token is generated will use the new routing function, thus avoiding the failed link.

TABLE 3
Comparison of Reconfiguration Times for the Different Schemes under Hot-Spot Traffic

	Reconfiguration Time			
Traffic load	OSR-PDA	OSR-LA	DS	SR
LOW	48.36μs	60.77μs	48.52μs	87.70μs
MEDIUM	48.68μs	59.54μs	49.11μs	88.52μs
HIGH	47 μs	78 μs	49 μs	96 μs

OSR-LA, on the other hand, waits until routing tables are distributed before generating reconfiguration tokens, causing more packets to be dropped due to routing across the failed link according to the old routing function.

The DS scheme similarly loses packets during the process of distributing the routing tables and also during the period over which switches connected to the failed link wait to receive the control packet that indicates that the new routing tables can be used. The amount of dropped packets for SR is higher than for OSR-PDA. For SR, the instruction to stop traffic injection is sent serially to all end nodes. This is an implementation detail. It is possible to broadcast the command to stop traffic injection in SR in order to achieve a packet loss similar to OSR-PDA. This, however, would come at the cost of a concomitant increase in packet dropping at the source end nodes. By comparing OSR-LA and DS, we see that DS causes fewer dropped packets, but the difference is not overwhelming. Note that the effects of packet drops during the change detection and routing table computation are not simulated. As discussed previously, this time dominates the total reconfiguration time; thus, this time will also dominate the number of packets dropped due to the failing link. It could be argued, therefore, that the difference in dropped packets between OSR-LA and DS is not necessarily significant.

The results presented thus far are derived from simulations driven by uniform traffic patterns. The reconfiguration schemes are also evaluated using hot-spot and bit-reversal traffic patterns. In what follows, only a selection of the results is presented in which OSR-LA performs differently, relative to what is observed for uniform traffic. For hot-spot traffic, Fig. 11 shows packet latency for a HIGH network load rate. Here, the base latency is around 5 μs as compared to 1.5 μs for uniform traffic. It is observed that SR enters saturation when launching the reconfiguration, as indicated by the average NL increasing significantly. The most interesting observation, however, is that OSR-LA experiences significant token latency for a short period of time due to the presence of token dependencies. Because of the hot spot, old packets are drained more slowly, which affects token propagation. Table 3 summarizes the reconfiguration times for hot-spot traffic. As can be observed, for HIGH traffic rates, OSR-LA experiences a higher reconfiguration time (78 μs) as compared to uniform traffic, which is due to the slower drainage of old packets and the induced higher token latency.

The number of dropped packets for hot-spot traffic is shown in Table 4. The number of dropped packets depends on the required time for reconfiguration; thus, results are mainly similar to the ones obtained for uniform traffic. The number of packets dropped at the source end nodes is only approximately 10 percent of what we see for uniform traffic. This is because only 10 percent of the sources send packets to the congested destination and thus experience packet loss. Finally, in Figs. 12 and 13, we show the evolution of latency after the reconfiguration processes have completed. A clear observation is that OSR-PDA and SR need a long relaxation time before the normal latency is reached. However, DS and OSR-LA require very little recovery time. This is due to the nonblocking nature of these schemes. However, for SR and OSR-PDA, queued packets must be delivered after reconfiguration and, as reconfiguration-induced latency increases, the number of queued packets (at end nodes or at switches) by the end of the reconfiguration process also increases.

5 Conclusion

Although several techniques for dynamic network reconfiguration have been proposed recently, none, to our knowledge, has found its way into commercial use. Static network reconfiguration is the dominating approach used, despite its apparent shortcomings. Improving the efficiency of SR is of practical importance, thus motivating this research.

This paper describes how the various phases of SR can be overlapped in order to increase parallelism. Packets routed according to the new routing function can be injected and routed in a network that undergoes reconfiguration simultaneously to packets routed and delivered to end nodes according to the old routing function. The proposed OSR protocol benefits from reduced reconfiguration time, packet latency, network downtime, and packet dropping close to dynamic reconfiguration techniques (better for OSR-PDA but worse for OSR-LA), without requiring additional data virtual channels, while, at the same time,

TABLE 4
Dropped Traffic for the Different Schemes under Various Loads and Hot-Spot Traffic

	Dropped packets due to the failed link				Dropped packets at source end nodes			
Traffic load	OSR-PDA	OSR-LA	DS	SR	OSR-PDA	OSR-LA	DS	SR
LOW	0	9	5	0	0	0	0	0
MEDIUM	1	15	14	1	0	0	0	0
HIGH	1	29	16	6	0	0	0	25

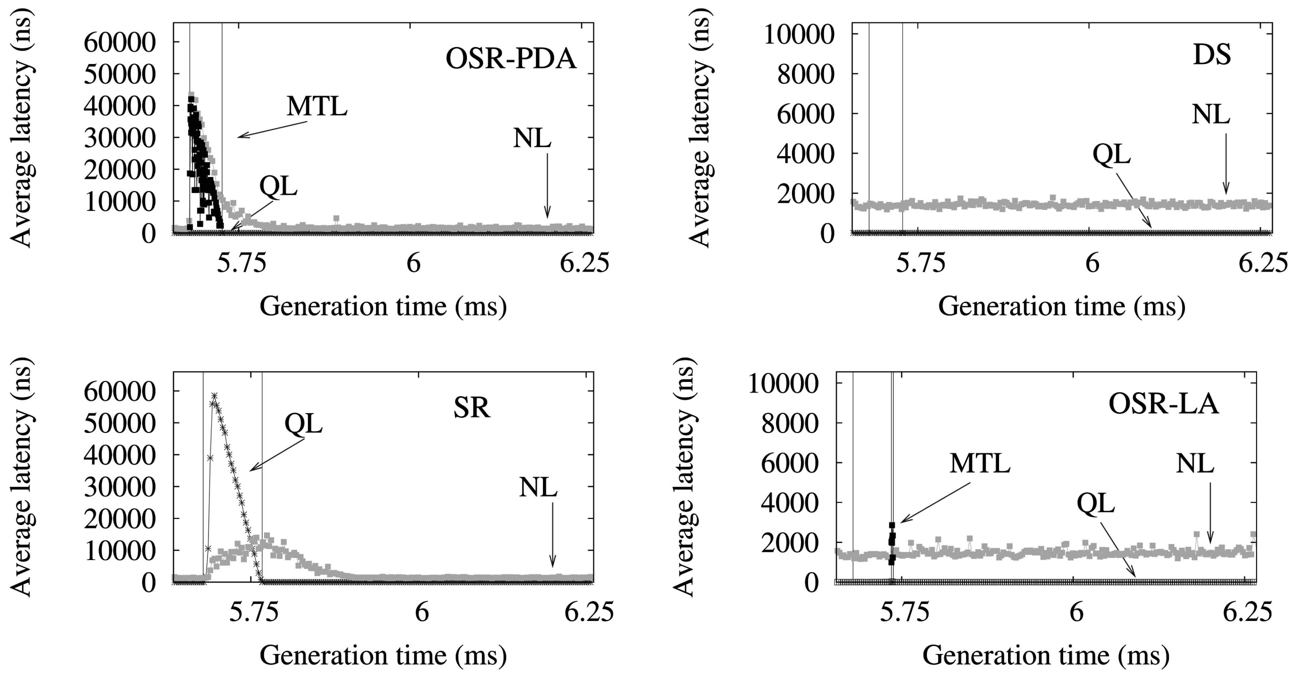


Fig. 12. Average packet latency at generation time under uniform traffic and MEDIUM network load.

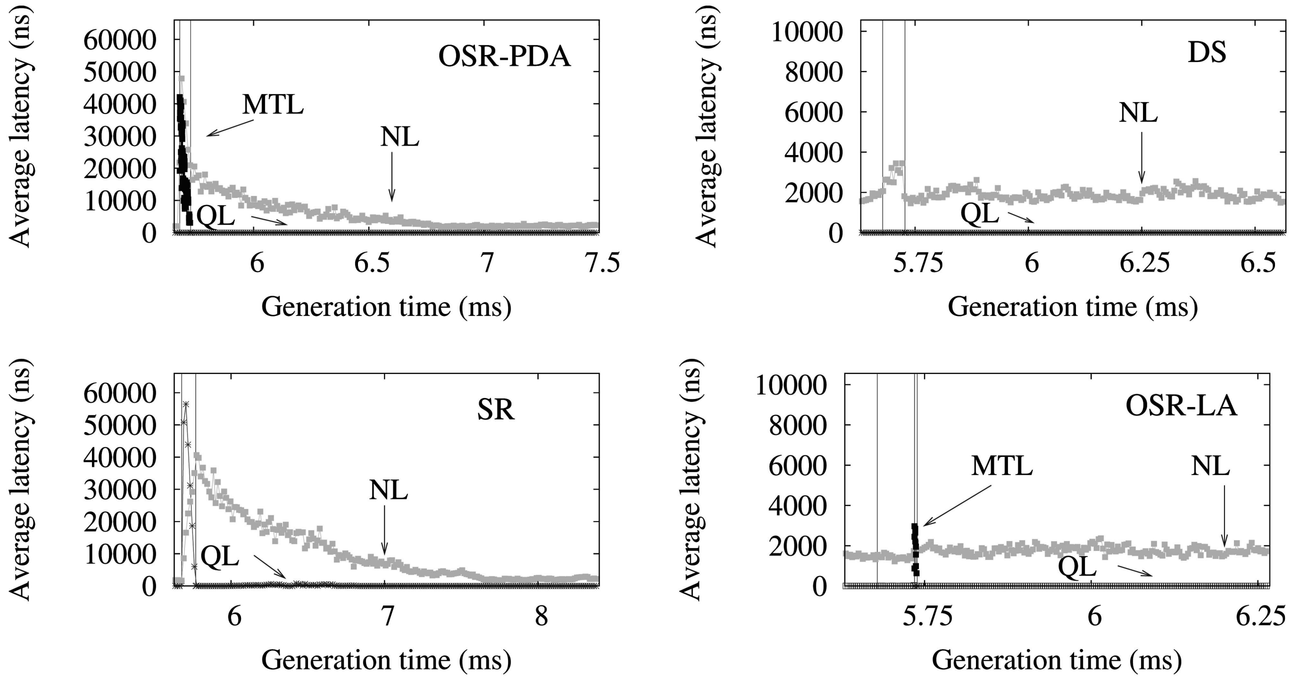


Fig. 13. Average packet latency at generation time for uniform traffic pattern and HIGH network load.

maintaining the generality and simplicity offered by traditional SR. It also guarantees in-order delivery of packets during network reconfiguration in the event that the old and new routing functions are deterministic. A detailed formal proof is developed which shows that the OSR protocol is, indeed, deadlock-free.

In the evaluation of the proposed protocol, simulation measurements and experiments were carefully designed to allow fair comparison to other competing techniques. Simulations were carried out, assuming finite buffers at network injection points, and packets dropped due to buffer overflow were measured in addition to packets dropped due to encountering link failure. To better capture the temporal effects of the reconfiguration processes, packet latency is plotted in relation to when packets are generated rather than when packets are delivered. Results show that the OSR-LA provides significant improvement over traditional SR and performs similarly to the DS dynamic reconfiguration technique. Given that the DS used in the comparisons does not guarantee in-order delivery (even for deterministic routing functions) and requires two sets of data virtual channels, even more benefits become apparent.

With regard to future work on network reconfiguration, we believe that the most important challenges lie in its application areas. In Section 4 of this paper, we focused on topology changes induced by faults, but other important application areas are adaptations to varying traffic loads and planned component replacement. A particularly interesting example of an emerging application area comes from the NoCs. We will, in the relatively near future, see processors with tens or even hundreds of cores on them and, most likely, with an on-chip interconnection network connecting the cores [42]. This will most likely mean that future programs running on these chips will be parallel [43] and that the hardware must accommodate several independent parallel processes. The separation of coexisting processes on the same hardware has been done by time-sharing, scheduling, and context switches on single-core CPUs and with virtualization of addresses on memory. Mechanisms for ensuring that different processes use separate resources in the network must therefore be developed and fast network reconfiguration will be an important ingredient in such a set of mechanisms.

ACKNOWLEDGMENTS

This work was supported in part by US National Science Foundation (NSF) Grant CCF-0541417. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF. The work was also supported by the Spanish MEC under Grants CONSOLIDER-INGENIO CSD2006-46 and TIN2006-15516-C04-04.

REFERENCES

[1] D. Teodosiu, J. Baxter, K. Govil, J. Chapin, M. Rosenblum, and M. Horowitz, "Hardware Fault Containment in Scalable Shared-Memory Multiprocessors," *Proc. 24th Ann. Int'l Symp. Computer Architecture, Computer Architecture News* vol. 25, pp. 73-84, 1997.

[2] K. Gharachorloo, M. Sharma, S. Steely, and S. Van Doren, "Architecture and Design of AlphaServer GS320," *ACM SIGPLAN Notices*, vol. 35, no. 11, pp. 13-24, Nov. 2000.

[3] "The AlphaServer SC45 Supercomputer: Facts and Figures," HP SC45 Team, 2002.

[4] W. Barrett et al., "An Overview of the BlueGene/L Supercomputer," *Proc. ACM/IEEE Conf. Supercomputing*, Nov. 2002.

[5] 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.

[6] 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, 1995.

[7] D. Garcia and W. Watson, "ServerNet™ II," *Lecture Notes in Computer Science*, vol. 1417, pp. 119-135, 1998.

[8] O. Feuser and A. Wenzel, "On the Effects of the IEEE 802.3x Flow Control in Full-Duplex Ethernet LANs," *Proc. 24th IEEE Conf. Local Computer Networks*, pp. 160-161, Oct. 1999.

[9] "Guide to Myrinet 2000 Switches and Switch Networks" Myrinet, www.myri.com, Aug. 2001.

[10] *InfiniBand Architecture Specification Volume 1 Release 1.0a.* InfiniBand Trade Assoc., http://www.infinibandta.com, 2001.

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

[12] M. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J.-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal, "The Raw Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs," *IEEE Micro*, vol. 22, no. 2, pp. 25-35, Mar./Apr. 2002.

[13] K. Krewell, "Sun's Niagara Pours on the Cores," *Microprocessor Report*, pp. 1-3, Sept. 2004.

[14] J.A. Kahle, M.N. Day, H.P. Hofstee, C.R. Johns, T.R. Maeurer, and D. Shippy, "Introduction to the Cell Multiprocessor," *IBM J. Research and Development*, vol. 49, nos. 4/5, 2005.

[15] D. Berger et al., "TRIPS Tutorial: Design and Implementation of the TRIPS EDGE Architecture," *Proc. 32nd Int'l Symp. Computer Architecture*, pp. 1-239, June 2005.

[16] M.B. Taylor, W. Lee, S.P. Amarasinghe, and A. Agarwal, "Scalar Operand Networks," *IEEE Trans. Parallel and Distributed Systems*, vol. 16, no. 2, pp. 1-18, Feb. 2005.

[17] D. Krolak, "Unleashing the Cell Broadband Engine Processor: The Element Interconnect Bus," *Proc. Fall Processor Forum*, http://www-128.ibm.com/developers/power/library/pa-fpfeib/, Nov. 2005.

[18] P. Gratz, K. Sankaralingam, H. Hanson, P. Shivakumar, R. McDonald, S.W. Keckler, and D. Burger, "Implementation and Evaluation of a Dynamically Routed Processor Operand Network," *Proc. First Network-on-Chip Symp.*, 2007.

[19] P. Kermani and L. Kleinrock, "Virtual Cut-Through: A New Computer Communication Switching Technique," *Computer Networks*, vol. 3, pp. 267-286, 1979.

[20] 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.

[21] W.J. Dally, "Virtual-Channel Flow Control," *IEEE Trans. Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194-205, Mar. 1992.

[22] 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.

[23] 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.

[24] W.J. Dally and B.P. Towles, *Principles and Practices of Interconnection Networks.* Morgan Kaufmann, 2004.

[25] S. Warnakulasuriya and T.M. Pinkston, "A Formal Model of Message Blocking and Deadlock Resolution in Interconnection Networks," *IEEE Trans. Parallel and Distributed Systems*, vol. 11, no. 2, pp. 212-229, Feb. 2000.

[26] M.D. Schroeder, A.D. Birrell, M. Burrows, H. Murray, R.M. Needham, T.L. Rodeheffer, E.H. Satterthwaite, and C.P. Thacker, "Autonet: A High-Speed, Self-Configuring Local Area Network Using Point-to-Point Links," SRC Research Report 59, Digital Equipment Corp., 1990.

[27] T.L. Rodeheffer and M.D. Schroeder, "Automatic Reconfiguration in Autonet," *Proc. 13th ACM Symp. Operating Systems Principles*, pp. 183-197, Oct. 1991.

[28] O. Lysne and J. Duato, "Fast Dynamic Reconfiguration in Irregular Networks," *Proc. 29th Int'l Conf. Parallel Processing*, pp. 449-458, 2000.

[29] R. Casado, A. Bermúdez, J. Duato, F.J. Quiles, and J.L. Sánchez, "A Protocol for Deadlock-Free Dynamic Reconfiguration in High-Speed Local Area Networks," *IEEE Trans. Parallel and Distributed Systems*, vol. 12, no. 2, pp. 115-132, Feb. 2001.

[30] N. Natchev, D. Avresky, and V. Shurbanov, "Dynamic Reconfiguration in High-Speed Computer Clusters," *Proc. Third IEEE Int'l Conf. Cluster Computing*, pp. 380-387, 2001.

[31] T. Pinkston, R. Pang, and J. Duato, "Deadlock-Free Dynamic Reconfiguration Schemes for Increased Network Dependability," *IEEE Trans. Parallel and Distributed Systems*, vol. 14, no. 8, pp. 780-794, Aug. 2003.

[32] J. Duato, O. Lysne, R. Pang, and T.M. Pinkston, "Part I: A Theory for Deadlock-Free Dynamic Network Reconfiguration," *IEEE Trans. Parallel and Distributed Systems*, vol. 16, no. 5, pp. 412-427, May 2005.

[33] O. Lysne, T.M. Pinkston, and J. Duato, "Part II: A Methodology for Developing Deadlock-Free Dynamic Network Reconfiguration Processes," *IEEE Trans. Parallel and Distributed Systems*, vol. 16, no. 5, pp. 428-443, May 2005.

[34] D. Avresky and N. Natchev, "Dynamic Reconfiguration in Computer Clusters with Irregular Topologies in the Presence of Multiple Node and Link Failures," *IEEE Trans. Computers*, vol. 54, no. 5, May 2005.

[35] J.R. Acosta and D.R. Avresky, "Intelligent Dynamic Network Reconfiguration," *Proc. 21st Int'l Parallel and Distributed Processing Symp.*, pp. 1-9, 2007.

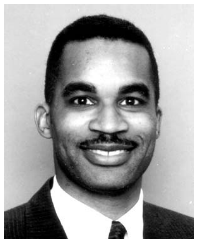
[36] J.M. Mellor-Crummey and M.L. Scott, "Algorithms for Scalable Synchronization on Shared-Memory Multiprocessors," *ACM Trans. Computer Systems,* vol. 9, no. 1, pp. 21-65, 1991.
[37] J. Duato and T.M. Pinkston, "A General Theory for Deadlock-Free Adaptive Routing Using a Mixed Set of Resources," *IEEE Trans. Parallel and Distributed Systems,* vol. 12, no. 12, pp. 1219-1235, Dec. 2001.
[38] J.M. Montañana, J. Flich, A. Robles, and J. Duato, "A Scalable Methodology for Computing Fault-Free Paths in Infiniband Torus Networks," *Proc. Sixth Int'l Symp. High-Performance Computing,* 2005.
[39] J.M. Montañana, J. Flich, A. Robles, and J. Duato, "Reachability-Based Fault-Tolerant Routing," *Proc. 12th Int'l Conf. Parallel and Distributed Systems,* pp. 515-524, 2006.
[40] InfiniBand Architecture Specification, InfiniBand Trade Assoc., 2000.
[41] A. Bermúdez, R. Casado, F.J. Quiles, and T.M. Pinkston, "Evaluation of a Subnet Management Mechanism for InfiniBand Networks," *Proc. 32nd Int'l Conf. Parallel Processing,* pp. 117-124, 2003.
[42] F. Angiolini, P. Meloni, S.M. Carta, L. Raffo, and L. Benini, "Layout-Aware Analysis of Networks-on-Chip and Traditional Interconnects for MPSoCs," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 26, no. 3, pp. 421-434, 2007.
[43] F. Poletti, A. Poggiali, D. Bertozzi, L. Benini, P. Marchal, M. Loghi, and M. Poncino, "Energy-Efficient Multiprocessor Systems-on-Chip for Embedded Computing: Exploring Programming Models and Their Architectural Support," *IEEE Trans. Computers,* vol. 56, no. 5, pp. 606-621, May 2007.

Olav Lysne received the MS and DrSci degrees from the University of Oslo in 1988 and 1992, respectively. He is currently the director of basic research at the Simula Research Laboratory and a professor of computer science at the University of Oslo. He has been a member of the program committees of several of the most renowned conferences, including HPCA, IPDPS, ICPP, EuroPar, and HiPC. He was a general cochair of ICPP '05. He has participated in a series of European projects. His research interests include algebraic specification and term rewriting. Over the last decade, he has been working on interconnects, in particular on effective routing, fault tolerance, and quality of service. He has published around 70 academic papers. He is a member of the IEEE.

José Miguel Montañana received the BS degree in physics and the MS degree in electronics engineering from the University of Valencia in 1997 and 2002, respectively. He is currently working toward the PhD degree in the Department of Computer Engineering at the Technical University of Valencia. His research interests include fault tolerance and reconfiguration in interconnection networks.

José Flich received the MS and PhD degrees in computer science from the Universidad Politécnica de Valencia, Spain, in 1994 and 2001, respectively. In 1998, he joined the Department of Computer Engineering (DISCA) at the Universidad Politécnica de Valencia, where he is currently an associate professor of computer architecture and technology. He has been a member of the program committee in different conferences, including ICPP, IPDPS, HiPC, CAC, ICPADS, and ISCC. He is also a cochair of CAC and INA-OCMC and the vice-chair of EuroPar. His research interests include high-performance interconnection networks for multiprocessor systems, cluster of workstations, and networks on chip. He is a member of the IEEE Computer Society.

José Duato received the MS and PhD degrees in electrical engineering from the Polytechnic University of Valencia, Spain, in 1981 and 1985, respectively. He was an adjunct professor in the Department of Computer and Information Science at The Ohio State University. He is currently a professor in the Department of Computer Engineering (DISCA) at the Polytechnic University of Valencia. His research interests include interconnection networks and multiprocessor architectures. He has published more than 380 refereed papers. He proposed a powerful theory of deadlock-free adaptive routing for wormhole networks. Versions of this theory have been used in the design of the routing algorithms for the MIT Reliable Router, the Cray T3E supercomputer, the internal router of the Alpha 21364 microprocessor, and the IBM BlueGene/L supercomputer. He is the first author of *Interconnection Networks: An Engineering Approach.* He was a member of the editorial boards of the *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, and *IEEE Computer Architecture Letters.* He was a general cochair of ICPP '01, the program committee chair of HPCA '04, and a program cochair of ICPP '05. He was also a cochair, a member of the steering committee, the vice-chair, or a member of the program committee for more than 55 conferences, including the most prestigious conferences in his area: HPCA, ISCA, IPPS/SPDP, IPDPS, ICPP, ICDCS, EuroPar, and HiPC. He is a member of the IEEE.



Timothy Mark Pinkston received the BSEE degree from The Ohio State University in 1985 and the MSEE and PhD degrees from Stanford University in 1986 and 1993, respectively. He is currently a professor in the Ming Hsieh Department of Electrical Engineering in the Viterbi School of Engineering at the University of Southern California. Since January 2006, he has been serving a three-year term as the program director of the Computer and Information Science and Engineering Directorate of the US National Science Foundation (NSF) for the computer systems architecture area and, more recently, the Expeditions in Computing Program. His research interests include interconnection networks and communication architectures for parallel processing systems, in particular multicore and multiprocessor computers. He has served on the editorial board of the *IEEE Transactions on Parallel and Distributed Systems* and has been a part of the organizing committee of several prestigious conferences, including ISCA, HPCA, IPDPS, ICPP, and HiPC. He was the program chair of ICPADS '06 and the general chair of IPDPS '07. He will serve as the program chair of HPCA-15. He is a senior member of the IEEE.

Tor Skeie received the MS and PhD degrees in computer science from the University of Oslo in 1993 and 1998, respectively. He is currently an associate professor at the Simula Research Laboratory and the University of Oslo. He has several years of experience as a researcher in the interconnect domain. His work is mainly focused on scalability, effective routing, fault tolerance, and quality of service in switched network topologies. He is also a researcher in the industrial Ethernet area. The key research topics here have been the road to deterministic Ethernet end-to-end and how precise time synchronization can be achieved across switched Ethernet.