(This is a sample cover image for this issue. The actual cover is not yet available at this time.)

This article appeared in a journal published by Elsevier. The attached copy is furnished to the author for internal non-commercial research and education use, including for instruction at the authors institution and sharing with colleagues.

Other uses, including reproduction and distribution, or selling or licensing copies, or posting to personal, institutional or third party websites are prohibited.

In most cases authors are permitted to post their version of the article (e.g. in Word or Tex form) to their personal website or institutional repository. Authors requiring further information regarding Elsevier’s archiving and manuscript policies are encouraged to visit:

http://www.elsevier.com/copyright
PDR: A protocol for dynamic network reconfiguration based on deadlock recovery scheme

Majed ValadBeigi, Farshad Safaei *

Faculty of Electrical and Computer Engineering, Shahid Beheshti University, G.C., Evin 1983963113, Tehran, Iran

Abstract

Dynamic network reconfiguration is described as the process of replacing one routing function with another while the network keeps running. The main challenge is avoiding deadlock anomalies while keeping limitations on packet injection and forwarding minimal. Current approaches which have a high complexity and as a result have a limited practical applicability either require the existence of extra network resources, or they will affect the network performance during the reconfiguration process. In this paper we present a simple, fast and efficient mechanism for dynamic network reconfiguration which is based on regressive deadlock recovery instead of avoiding deadlock. The mechanism which is referred to as PDR guarantees a deadlock-free reconfiguration based on wormhole switching. In PDR, a particular approach is taken to handle both deadlocks and performance degradation. We propose the use of a packet injection restriction mechanism that prevents performance degradation near the saturation by controlling the network traffic. Further, in this approach, to accurately detect deadlocks, the deadlock detection mechanism is implemented and also improved by using only the local information, thereby considerably reducing false deadlock detections. In the rare cases when deadlocks are suspected, we propose a new technique that absorbs the deadlocked packet at the current node instead of dropping deadlocked packets and re-injects it later into the network. The main advantage of this method is its simplicity and also it does not require any additional buffers in intermediate nodes to handle deadlocks. It requires only some buffer space in the local node to temporarily hold the deadlocked packets removed from the network. Evaluating results reveal that the mechanism shows substantial performance improvements over the other methods and it works efficiently in different topologies with various routing algorithms.

1. Introduction

All digital electronic systems types are rapidly becoming communication limited. The cause of restraining costs, performance, size and power in such systems is the movement of data, not arithmetic or control logic. Meanwhile, buses, long the mainstay of system interconnect, are unable to keep up with increasing performance requirements.

Interconnection networks offer an attractive solution to this communication crisis and are becoming pervasive in digital systems. They are used in almost all digital systems that are large enough to have two components to connect. The most common applications of interconnection networks are in computer systems and communication switches. In computer systems, they connect processors to memories and input/output (I/O) devices to I/O controllers. They connect input ports to the...
output ports in communication switches and network routers. They also connect the sensors and actuators to the processors in control systems. Anywhere that bits are transported between two components of a system, an interconnection network is likely to be found [1]. They are also found in high-end servers [2–4] in the form of system area networks [5–8] as well as in multicore processors [9,10] as networks-on-chip (NoC) [11–13] at the other end of the spectrum.

Various switching mechanisms have been described in the literature for interconnection networks including packet switching (PS), virtual cut-through (VCT) and wormhole switching (WS) [14].

WS [15] (also referred as wormhole-routing) has become the most widely used switching mechanism for multicomputers and distributed shared-memory multiprocessors, and it is also being used in networks of workstations [6]. The use of virtual channels can considerably increase the network throughput by dynamically sharing the physical bandwidth among several packets [16]. However, it has been shown that virtual channels are expensive, increasing node delay considerably [17]. Therefore, the number of virtual channels per physical channel should be kept small. Besides, for the sake of transmission and flow control, WS is fragmented into elemental units, called flits [14].

In PS and VCT, packets are completely buffered in a node. As a result, the packets consume network bandwidth proportional to the network load. On the other hand, wormhole-switched packets may block occupying buffers and channels across multiple routers, precluding access to the network bandwidth by other packets [14].

With increasing probability of failure and reliability concerns for interconnection networks, fault-tolerance is quickly becoming an integral part of such systems. It is therefore necessary to provide an efficient fault-tolerant mechanism to keep the system running, even in the presence of faults.

Fault-tolerance is defined as the ability of a system to continue an operation despite the presence of faults [1]. Three most applicable terms in fault-tolerance area are reliability, availability, and dependability [18]. However, due to the large application area, interconnection networks are found in the systems with high requirements for reliability and continued operation.

In case of a component failure, even in a degraded mode, the use of fault-tolerance mechanisms will assure that the system keeps working until the failed component is repaired. Basically, there are three ways to cope with faults in the interconnection networks: component redundancy, fault-tolerant routing algorithms, and reconfiguration techniques [19]. Using component redundancy has been the easiest yet costly way to provide fault-tolerance. Components in the system are replicated and once a failed component is detected, it is simply replaced by its redundant copy.

Fault-tolerant routing algorithms aim at avoiding packets from traversing faulty components by providing some kind of routing path redundancy. To reach this end, packets must be able to be routed through alternative paths to circumvent or avoid faulty regions over the network. Fault-tolerant routing schemes should be designed to tolerate a certain number of faults while still guaranteeing deadlock freedom in the network. However, to fulfill the requirements, fault-tolerant routing strategies often need to use additional network resources such as virtual channels or additional hardware at switches or routers.

Applying reconfiguration [20] on any number of faults can be tolerated, provided that the network is physically connected. Once a fault is detected, the reconfiguration mechanism consists of discovering new topologies, computing a new routing scheme and updating the required components in the network. The main disadvantage of reconfiguration is the high delayed packets that may occur during the reconfiguration process.

Reconfiguration techniques can be either static or dynamic. Static reconfiguration techniques require completely stopping the traffic in the network before changing any routing table, so the network is emptied. The routing algorithm used after the reconfiguration process is different. It implies that all the paths for each source–destination pair need to be computed. Owing to the network down-time, i.e., halting packet injection that may cause strong performance degradation during the reconfiguration process, static reconfiguration has a large impact on packet latency. This issue prevents static reconfiguration techniques from being used in systems with high performance requirements.

Unlike the static reconfiguration, in a dynamic reconfiguration the transition from one routing function to another is performed while the functional parts of the network are fully operational, i.e., we have no network down-time and no halting packet injection. Compared to the static reconfiguration, this typically leads to a reduction in the number of packets that miss their quality of service deadline. The problem in this approach resides in the fact that, in general, two different and individually deadlock-free routing functions may be prone to deadlock if they coexist in the network. It means that, in a dynamic reconfiguration, there will be a transition phase between the old and new routing functions where reconfiguration-induced deadlocks may occur. Another drawback of using dynamic reconfiguration is that it usually requires extra resources.

The main challenge of a dynamic reconfiguration process is to avoid deadlock anomalies. In a global reconfiguration process, the system migrates from a routing algorithm (9Old) to a new one (9New). Although both routing algorithms are deadlock-free when isolated, during the reconfiguration process both algorithms coexist and they may induce cyclic dependencies between packets (old packets and new packets). As a second challenge, dynamic reconfiguration processes must reduce packet ejecting rate while keeping limitations on packet injection and forwarding minimal.

To overcome these challenges, in this paper we present a simple and powerful method for dynamic network reconfiguration. The mechanism, referred as PDR, guarantees a fast and deadlock-free transition from the 9Old to the 9New routing function, but instead of avoiding deadlocks the mechanism is based on regressive deadlock recoveries. In PDR, a particular approach is taken to handle both deadlocks and performance degradation. We propose the use of a packet injection restriction mechanism that prevents performance degradation near the saturation by controlling the traffic and reducing the total number of the traffic lost in fault scenarios. Further, in this approach, to accurately detect deadlocks, the deadlock detection mechanism is implemented and also improved by using only the local information, thereby considerably reducing false
deadlock detections. Instead of measuring the time a packet is blocked, the proposed method measures the time that channels requested by packets are inactive due to the current packet occupying them being blocked.

A packet in our approach is presumed to be a deadlocked packet if all the alternative output virtual channels that are requested by that packet contain blocked packets. It is clear that when the routing algorithm uses all the virtual channels in each physical channel in the same way, it is only necessary to monitor activity in the physical channel. In the rare cases when deadlocks are suspected, we propose a new technique that absorbs the deadlocked packet at the current node and re-injects it later into the network. The main advantage of this routing is its simplicity and also it does not require any additional buffers in intermediate routers to handle deadlocks. It requires only some buffer space in the local node to temporarily hold deadlocked packets removed from the network.

The rest of the paper is organized as follows. Related work is presented in Section 2. In Section 3, we present PDR method and describe implementation details. In Section 4, PDR is evaluated. Finally, in Section 5, conclusions are provided.

2. Related work

Faults in a network take many forms, such as hardware faults, software bugs, malicious sniffing or removal of packets. The first step in dealing with errors is to understand the nature of component failures and then to develop simple models that allow us to reason about the failure and the methods for handling it. Classification of faults by nature is either random or systematic. Random faults are usually hardware faults affecting the system components, which occur with a certain probability, while systematic faults such as software failures are faults which are not random, whether a component has it or not [14]. We assume that such permanent failures are detected and contained on a node or link boundary. Thus, faults are assumed to be fail-stop [21], meaning that we do not consider Byzantine (i.e., malicious) faults [14]. In the contexts of fault-tolerant routing, these are common assumptions [12, 21–23].

Faults also can be classified by their duration as transient and permanent faults [14]. Transient faults persist in the system for only a short duration, while permanent faults remain in the system until it is repaired and may be either dynamic or static. In a dynamic fault model, once a new fault is found, actions are taken in order to appropriately handle the faulty component which allows the system to reconfigure at the hardware level, and preserves the original network topology. A static system provides a fault-tolerant routing algorithm that will bypass any failed node.

In some situations the defined promises of the routing algorithm and/or network topology may break, affecting the network dependability. This may happen when the network topology is changed, either involuntarily due to the faulty components or voluntarily due to the removal or addition of components. This normally requires the network routing algorithm (routing function) to be reconfigured in order to re-establish the connectivity of the entire network [24].

With reconfiguration, once a fault is detected, the process starts discovering the new topology and computing and distributing the new routing tables. However, during the reconfiguration process (in a dynamic fault model) possibly some extra resources will be required by the reconfiguration technique. Reconfiguration is appropriate for diverse networks (Myrinet or InfiniBand). As an example, topologyagnostic routing algorithms like up/down [25], DFS [26], and Segment-Based Routing [27] can be used in order to compute the routing tables for the new (possibly irregular) topology.

Unlike static reconfiguration techniques, dynamic reconfiguration techniques [20] do not require bringing the network traffic to a complete stop. However, some packets must be removed from the network and re-injected later, which could cause a strong degradation in performance during the reconfiguration time. In the last decade several dynamic reconfiguration mechanisms have been proposed. Next we describe some of them.

In [20], a Partial Progressive Reconfiguration (PPR) technique is proposed, allowing arbitrary networks to migrate between two instantiations of up/down routing. The effect of network load and size of the network on PPR performance is evaluated in [28].

Another approach is the NetRec scheme [29] which requires each switch to maintain information about switches in some hops away. Yet another approach is the Double Scheme (DS) [30] that uses two sets of virtual channels in the network which act as two disjoint virtual network layers during the reconfiguration. A methodology for deriving new reconfiguration processes for any given pair of old and new routing function is given in [24]. An orthogonal approach which may be applicable on top of all the above mentioned techniques is described in [31], where, for up/down routing only parts of the network need to be reconfigured on a network change. Solid theoretical support on which dynamic reconfiguration design methodologies and techniques are proved deadlock-free can be found in [32].

Moreover, a mechanism was suggested in [24] which is referred as Simple Reconfiguration (SR). In SR a token is issued to separate packets routed with the old routing function from packets routed with the new routing function. Tokens advance through an output port in a switch once there are no more old packets passing through the output port (based on input and output dependencies generated from the old routing function). By doing this, there are no cycles in the network since there will be no old packets behind new ones.

The above mentioned mechanisms lack at least one of the identified goals in this paper. In particular, PPR only works with routing functions that adhere to the up/down scheme. NetRec [29] is specially tailored for re-routing packets around a faulty node. It basically provides a protocol for generating a tree which connects all the neighbors of a fault, and drops packets to avoid deadlocks in the reconfiguration phase. DS is more flexible, in the sense that it can handle any topology and transition between any pair of deadlock-free routing functions. However, it requires the presence of two sets of virtual channels.
The methodology in [24] requires complex computation in order to derive a safe reconfiguration process once the new routing function has been decided. It consumes time and thus limits the applicability of the methodology. Finally, SR mechanism requires a token to be distributed over the entire network. Although it separates the old and the new traffics, it has two major drawbacks. The first one is that its implementation is not straightforward. The token distribution is based on the dependencies of the old routing function. The second, packets suffer from some extra blocking since new packets must wait for the tokens to advance.

PDR can exhibit superior performance characteristics over the following goals. First, packets are not getting blocked by any cause. They are routed as soon as possible. Therefore, the packet latency is minimized. Second, nodes close to the failed link or node are updated in a quick way. Third, the packet injection restriction mechanism is presented which uses only the local information, detects all the deadlocks, and considerably reduces the probability of false deadlock detection. Fifth, our technique absorbs the deadlocked packet at the current node instead of dropping deadlocked packets and re-injects it later into the network. Sixth, nodes react quickly in the presence of a reconfiguration process. Seventh, the mechanism is simple and also it does not require any additional buffers in intermediate routers to handle deadlocks. The mechanism has been performed based on WS and works with any routing algorithm implemented in the form of routing tables at nodes.

### 3. Idea description

We consider the use of two routing algorithms referred to as $R_{\text{old}}$ and $R_{\text{new}}$. Routing information is distributed along the nodes by using routing tables. The mechanism is based on the fact that deadlock does not happen frequently, so recovery may be preferred to prevention. Indeed, the probability of deadlock is proportional to the traffic injection rate and inversely proportional to the availability of resources. Assuming that the network works on a non-congested scenario, the probability of deadlock is minimum.

PDR mechanism will allow cycles to be formed (see Fig. 1). A deadlock is potentially formed when a packet cannot be transmitted due to a cycle occurrence. Such a cycle can be only formed when different nodes route packets with different routing algorithms ($R_{\text{old}}$ and $R_{\text{new}}$) at the same time. For example, a cycle formed in a reconfiguration process is showed in Fig. 1. It is formed since there is a cyclic dependency among all the packets/queues. Notice that a packet routed with the new routing function is waiting (in the cycle) behind a packet routed with the old routing function, and in the same cycle a packet routed with the old routing function is waiting behind a packet routed with the new routing function.

However, notice that if queues involved in the cycle have enough space for the packets to advance, a deadlock will not happened in the network. Indeed, once a packet can be forwarded, the cycle disappears. In the case that queues are full, packets cannot advance and, thus, a deadlock occurs. To solve this condition, the mechanism will eject some of the packets in order to remove the deadlock. Note that PDR is proposed as a reconfiguration mechanism for tolerating faults and reducing the traffic lost caused by the faults during the process, and although PDR may eject some packets results in reducing the total number of lost packets in the fault scenarios by introducing packet injection restriction mechanism which is described in Section 3.1. Also notice the retransmission mechanisms are still needed in networks with fault scenarios, thus ejected

![Fig. 1. Cycle in a dynamic reconfiguration not introducing a deadlock.](image-url)
packets by PDR (if there are any) will be retransmitted. Fig. 2 summarizes all the steps following the reconfiguration process at the different network components. The following sections describe every one of the steps.

### 3.1. Packet injection restriction

Assume that in the situation that the network reaches saturation, the cyclic channel dependencies are formed by the packets and the resources which lead buffers to become full along the cycle and thus a deadlock occurs. The problem can be stated as follows: the number of ejected packets would be high and Latency increases with network traffic until it reaches a certain situation (deadlock situation) is reached, at that time the latency value increases considerably while throughput (accepted traffic) tails off. Performance degradation within the network occurs when a cyclic dependency between channels is formed while the network reaches saturation. When the traffic goes high, the packets block cyclically faster than they are drained by the escape paths, so it leads to increasing in latency and decreasing in throughput. Provided that there are some escape resources to drain packets blocking cyclically, deadlock cannot occur, but packets wait for a long time in the network.

The solution for the problem is to control network traffic to ensure that it is always under the performance degradation point. But traffic is often global in nature. Thus, it is not feasible to easily measure it at any node. As an approximation, traffic can be estimated locally by counting the number of busy virtual output channels at each node [33]. The average number of busy virtual output channels at each router monotonically increases with the network load. Hence, we can establish that there is a useful correlation between the number of busy virtual channels and the network tending to saturation. With using this definition global traffic rate is simply approximated by monitoring the number of busy virtual output channels local to a router. This technique is used in implementing our packet injection restriction mechanism to avert network saturation, therefore the deadlock is prevented.

When the number of busy virtual output channels exceed a threshold value \( t \), the router prevents the injection of new packets, keeping them at the source node. If we select the suitable \( t \), there is a high probability that the network will never reach saturation and performance degradation can be relieved. A simple implementation of this mechanism requires only a register which holds the \( t \) and also a comparator counter \( D \) associated with each router. \( D \) is incremented each time a successful route is established (another output virtual channel becomes occupied) and is decremented when the tail of a packet leaves the router.

### 3.2. Status information distribution

In some situations the defined promises of the routing algorithm and/or network topology may break and naturally affect the network dependability. This may happen when the network topology is changed, either involuntarily due to the faulty components or voluntarily due to the removal or addition of the component. This normally requires the network routing algorithm to be reconfigured in order to network connectivity to be re-established. PDR is applied whenever a new routing algorithm is needed for the network. In some conditions the change in the topology does not necessarily lead to a change in the routing algorithm. This is the case of adding/removing nodes to/from the system. As the routing algorithm remains unchanged, there is no probability of deadlock, thus no demanding for a global reconfiguration process. On the other hand, a new node or link might be added, changing the topology. In that case, a change of the routing algorithm could lead to achieving higher performance and thus activating a reconfiguration process. The routing algorithm might also need to be changed even if no topological changes occur. In cases which the topology makes no changes in, by modifying the routing algorithm higher performance might be achieved. However, the most considerable topological change is when a node or link (or a group of them) fails. Once some parts of the network are being disconnected, altering the routing algorithm is required. To alleviate this problem, we will introduce an efficient mechanism in the situation that a failure occurs.

![Fig. 2. The steps of PDR mechanism.](image-url)
In all the above mentioned cases, PDR reconfiguration mechanism is activated. To do so, a selected node runs a Components Manager (CM) to be responsible for detecting any topological change. To achieve this, CM periodically sends control signals to all nodes. Nodes reply with the current status of their links and neighboring nodes. With this provided information, the CM builds the current topology. On detecting the topological change, the new paths are computed. At the moment that the nodes are notified, they use the alternate paths to bypass the failed node. However, employing the alternate paths is an additional mechanism which has nothing to do with the reconfiguration process. Therefore, we only focus on the reconfiguration process (i.e., updating tables).

Once the paths are computed, they must be distributed by sending all the new routing tables to all nodes through the network. Fig. 3 shows an example of the sequence used in updating the nodes. Once a node finds out the new paths, it updates its routing table removing the old one.

Additionally, in order to reduce the traffic control overhead, only the differences between the old and the new routing tables might be sent to each node. Depending on the similarity of the routing algorithms the percentage of traffic control reduction may be significant. Indeed, we will see in the evaluation that this improvement will affect the effectiveness of the mechanism.

3.3. Deadlock detection mechanism

A deadlock detection mechanism should have two important features. First, it should be simple; it should not add unnecessary complexity to the network that could decrease performance and/or increase the costs. Second, it should be implemented as a distributed mechanism, working only with the local information available at each router.

Instead of measuring time duration for a packet to be blocked, the proposed mechanism measures the time that channels, requested by packets, are inactive due to the current packets occupying them while remaining blocked. Transmission activity is monitored in all the virtual output channels which can be used by a given blocked packet. A packet is only presumed to be deadlocked if all the alternative virtual output channels, requested by that packet, contain blocked packets. It is worth mentioning that when the routing algorithm uses all the virtual channels in each physical channel, it is only necessary to monitor the activity in the physical channels.

This mechanism can be implemented as follows. A counter \( \mathcal{A} \) is associated with each output physical channel. \( \mathcal{A} \) will increment once for every clock cycle and is reset when a flit is transmitted across the physical channel. Thus, \( \mathcal{A} \) contains the number of cycles that show this channel is inactive. Note that \( \mathcal{A} \) also indicates the number of cycles since the last flit transmission across any of the virtual channels in that physical channel. This time is continuously compared with a given threshold \( \mathcal{T} \). If it is greater than the \( \mathcal{T} \), a one-bit flag (inactivity flag which is denoted by \( \mathcal{F} \)) is set illustrating that the physical output channel is inactive. \( \mathcal{F} \) is reset when a flit is transmitted across the physical channel.

The routing control unit is assigned to the packet headers in a round-robin fashion. Blocked headers are also routed in order to determine whether some of the output channels requested by them became free. Each time a packet is routed, the \( \mathcal{F} \)'s associated with the corresponding physical output channels are checked if all the feasible virtual output channels are busy. If all of these \( \mathcal{F} \)'s are set, there is no activity through any of the feasible physical output channels and the packet is presumed to be involved in a deadlock. It is important to note that \( \mathcal{F} \)'s and \( \mathcal{A} \)'s are associated with physical output channels instead of virtual channels. In order to implement the mechanism, the only required hardware is a counter, a

![Fig. 3. Sequence updating of routing table while the failure occurs at the node F. The numbers indicate both the sequence and the distance to the failed component.](image-url)
comparator and a single bit latch associated with each physical output channel. Obviously, the mechanism will detect all possible deadlocks. Yet there might exist a limited number of false deadlocks depending on the used $T$. Thus, the mechanism must be properly tuned, choosing the appropriate $T$.

3.4. Deadlock recovery process

From this study, we know that deadlocks are very rare. Moreover, the packet injection restriction mechanism described in Section 3.1 can be used to further reduce the probability of reaching a deadlock. Status Information Distribution approach is presented in Section 3.2. Also, the technique depicted in Section 3.3 is used for deadlock detection. Although deadlocks are highly improbable, they could still be detected. So, a recovery mechanism is required.

It is easy to see that in a cyclic dependency and thus a deadlock, at least one of the packets involved in it will have its header at the head of an input buffer, waiting for an output channel. Also, the proposed deadlock detection mechanism only presumes that a packet is deadlocked if its header is being routed (it is at the head of an input buffer). Thus, all we have to do in order to recover from deadlock is to remove that packet from the network by ejecting it at the current node. This can be easily accomplished by the router when it detects a possibly deadlocked packet. The router selects the internal memory channel at the current node for this packet, as if this node were its destination. A control bit is required to distinguish between normal and deadlocked packets. Finally, removed packets must be re-injected into the network at a later time. This is also easy to accomplish.

The proposed mechanism is a low-cost progressive deadlock recovery technique. Instead of dropping the deadlocked packets, it absorbs them at the current node, allowing them to make progress at a later time. The main advantage of this technique is its simplicity. The proposed recovery mechanism does not even require dedicated buffers in the router to recover from deadlock. It is enough to have some buffer space in the local node.

4. Evaluation

In this section we evaluate the proposed reconfiguration mechanism. To do so, we first present the evaluation methodology and traffic patterns. Then, we briefly describe the reconfiguration mechanisms used for comparison purposes. Finally, results and analysis are presented.

4.1. Evaluation methodology

We developed a detailed simulator that allows modeling the network at the cycle level. An event-driven simulator as Xmulator [35] was used for evaluating the performance of the proposed methodology. The Orion power library [36] was integrated to our simulator to calculate the power consumption of the networks.

In order to determine the fault-tolerance of the variations of our methodology, we have performed dynamic reconfiguration analysis for an $8 \times 8$ torus topology. WS is considered as the flow control mechanism. When wormhole flow control is applied, a packet is not allowed to re-enter the adaptive channel after having been routed in the escape channel. When adaptive routing is used, it is possible to have cycles in the dependency graph and for the routing function still to be deadlock-free. It is shown in [1,14] that a routing function is deadlock-free if there exists a routing sub function which is connected and has no cycles in its extended dependency graph. The channels belonging to the deadlock-free routing sub function are referred to as escape channels, and serve to enable packets to escape from cycles. We refer to the remaining channels as adaptive channels. The simulations have been performed using a base packet size of $16$ flits and also the width of $128$ bits for each flit. Moreover, each physical link was split into two virtual channels (VCs). We calculated the power consumption for each router’s links in $70$ nm technology library. In this technology, the clock frequency was set to $250$ MHz, and the length of links between two adjacent routers was set to $1$ mm for the torus topology. In order to make the performance results independent of the relative positions of the faults, a large number of simulations have been performed.

4.2. Traffic patterns

We consider two different traffic patterns when evaluating the network behavior: synthetic patterns and traces [1].

Synthetic patterns are widely used because they allow evaluating the network in the most generic way. When we use them, every node has the same traffic injection rate. We evaluate the complete range of traffic injection rate, from low levels up to the saturation point. The used synthetic traffic patterns are uniform and Hotspot [1].

- For uniform traffic, each source node sends packets to all the destinations with the same probability.
- For hotspot, 10% of the sources (selected randomly) inject traffic to the same destination (selected randomly), the rest of end nodes inject traffic to random destinations. This traffic pattern allows to model the situation when one or more end nodes are frequently accessed by the remaining end nodes (a disk server, for instance).
On the other hand, traces are based on capturing the traffic when running real applications. Traces contain the source, destination, injection time, and the size of each sent packet. They allow obtaining results in more realistic scenarios and let us compare them with the results obtained when using synthetic patterns. In this paper, some results obtained from this type of traffic pattern are shown.

The traces used were extracted under the execution of the FFT, LU, BARNES, RADIX, WATER-Nsquared, and WATER-Spatial applications from SPLASH-2 suite in shared-memory multiprocessors. These types of applications are widely used when simulating multiprocessor systems on engineering and scientific computations.

4.3. Evaluation mechanism

PDR mechanism is evaluated when sending all the routing tables. We compare it with SR and DS mechanisms.

In all the reconfiguration mechanisms, once a change in a topology is detected, the new routing tables along with a control packet are sent to all the nodes through the control virtual channel. In the case of DS, during the distribution of paths one virtual channel is drained. Once drained, control packets are sent to restore normal operation (the reconfiguration has finished). In the case of SR, the tokens that separate the old and the new traffic simultaneously are sent to the nodes. More details of DS and SR can be found in [34].

4.4. Results and analysis

We evaluate all the reconfiguration mechanisms for a random node failure in an $8 \times 8$ torus network during 10,000,000 cycles of execution. In this case, the old and the new routing algorithms are the up/down. For the packet ejection and re-injection at intermediate nodes, we assumed a delay of 200 cycles. The threshold for deadlock detection was four times the size of the longest packet.

Fig. 4 shows the average packet latency for the $8 \times 8$ torus for each reconfiguration mechanism and for different injection rates. The figure reveals that the latency of SR scheme increases significantly. The reason is that SR experiences higher latency due to the blocking introduced by the tokens. This effect is rapidly extended and thus the packets experience higher latencies that this phenomenon carries on even though the reconfiguration process has finished. Further, According to Fig. 4 the average packet latency of DS and PDR grows steadily with the increase of traffic injection rate. The reason for the little difference between DS and PDR in higher rates is that though both have two VCs, the operations deadlock recovery and deadlock detection, have caused more average packet latency in PDR compared to DS. Besides, due to the packet injection mechanism limitation for PDR in higher rates, it grows with the same increasing rate and never saturates, while DS and SR have very high average packet latency. Note that in traffic injection rates 0.025 packets/node/cycle the average packet latency of PDR, DS and SR is 79, 218, and 300 cycles, respectively.

Moreover, each mechanism requires the same reconfiguration time regardless of the traffic rate. This is because in all the cases the distribution of routing tables is the process which takes most of the time (tables are sent sequentially) and the amount of information to distribute is the same despite the traffic injection rate. Also, it is apparent that control packets use the reserved control virtual channel and they have a higher priority than data packets.

Fig. 5 shows the average packet latency plotted against the generation time for the different reconfiguration schemes under the uniform traffic. As can be seen in Fig. 5a, figure moves almost steadily with different cycles. Note that this stability in the figure shows that our approach is able to adopt with every topological changes and there is no sudden changes in PDR latency. Also very little changes in increasing latency compared to other cycles are seen in some of the cycles and it is caused by some operations such as a deadlock detection and recovery. We have also evaluated the mechanisms with a variety of traffic patterns. Fig. 6 shows the average packet latency for an $8 \times 8$ torus network for different synthetic traffic patterns.

Table 1 shows the number of packets detected as possibly deadlocked for different values of threshold, measured in clock cycles. A packet is counted if it has been detected as possibly deadlocked in any checkpoints. The table shows the values for our detection mechanism proposed in this paper (ODM) and previously proposed mechanisms based on time-outs (Touts)

![Fig. 4. Average packet latency vs. the generation traffic for an $8 \times 8$ torus network under the uniform traffic pattern.](image)
However, since all the packets arrived at their destination, none of them is deadlocked. As we can see in Table 1, the routing algorithm with two virtual channels per physical channel and short packets requires a threshold not lower than 64 cycles in order to avoid false deadlock detections using the ODM. Therefore, the optimal value for the threshold depends on the packet length. However, the proposed mechanism considerably reduces the number of false deadlock detections over crude time-outs. Moreover, no deadlocks are detected even when the network reaches the saturation point, provided that the threshold is properly tuned. This is not the case for detection mechanisms based on crude time-outs. These mechanisms detect a much higher number of false deadlocks for the same timeout value. Additionally, the number of detected deadlocks increases very quickly when the network approaches saturation. Hence, the proposed deadlock detection mechanism considerably improves over previously proposed ones.

Fig. 5. Average packet latency in an $8 \times 8$ torus network during the reconfiguration process.
Fig. 7 compares the latency by PDR, SR, and DS across six SPLASH-2 traces, on a 7 × 7 torus. The results are normalized to the results given by PDR. On average, PDR outperforms DS by 10% and SR by 24% when considering the packet latency. For the programs like WATER-Nsq, and WATER-Spatial, where the traffic is distributed rather evenly across the nodes, DS provides a better improvement than PDR over the SR. PDR, on the other hand, outperforms DS when a large portion of the packets are transmitted along several (preferably small number of) high-volume traffic flows. This traffic pattern, as shown before, can be efficiently handled by PDR. The RADIX traffic is an example of such traffic pattern for which the PDR gives the largest improvement, with 26% reduction in latency over DS and 35% over the SR.

Fig. 8 compares the Power obtained by PDR, SR, and DS across six SPLASH-2 traces, on a 7 × 7 torus. The results are normalized to the results given by PDR. On average, considering the power consumption, PDR underperforms DS by 5% and SR by 2% when considering the power consumption.

As an overall conclusion it can be seen that in terms of latency and power consumption, PDR has outperformed DS and SR. Besides, SR blocks the traffic in order to avoid packets from being routed through the failed link. As we have seen, it has a great impact on increasing the packet latency.
5. Conclusions

In this paper we have presented a novel and efficient mechanism called PDR which has been also compared with two prominent mechanisms namely Double Scheme (DS) and Simple Reconfiguration (SR). The suggested approach is a dynamic reconfiguration mechanism that uses a regressive deadlock recovery scheme in order to guarantee the deadlock-freedom condition in the network. Our method works for variety of topologies and operates with any pair of routing functions. PDR guarantees a deadlock-free reconfiguration based on wormhole switching. In this scheme, a particular technique is taken to handle both deadlocks and performance degradation. In this article, we introduced a set of mechanisms that minimize the hardware requirements to handle the deadlocks. In particular, we proposed a packet injection restriction mechanism that reduces the probability of deadlock to negligible values, and eliminates performance degradation at the saturation point. We also proposed an improved deadlock detection mechanism that considerably reduces the probability of false deadlock detection. Besides, we proposed a progressive deadlock recovery mechanism that requires no buffers to recover from deadlocks. It is based on the absorption of messages detected as being deadlocked at the current node and their re-injection into the network. Evaluating results reveal that PDR can provide superior performance over a simple reconfiguration under various traffics, especially, when the same amount of hardware is required. Further, as an overall conclusion it can be seen that in terms of latency and power consumption, PDR has outperformed DS and SR. Moreover, in conclusion we proposed a set of mechanisms to handle the deadlocks requiring a very small amount of hardware, eliminate the performance degradation at saturation point, reduce the frequency of deadlock to negligible values, and considerably reduce the probability of false deadlock detection. To the best of our knowledge, this is the first feasible deadlock handling technique for wormhole-switched networks that requires no dedicated buffer resources to handle deadlocks. As for future work, we are planning to apply our approach on on-chip interconnection networks such as NoC to improve the performance and decrease the power consumption.

Acknowledgement

The authors would like to thank the vice-presidency of Shahid Beheshti University for financial support of this work under Grant # 600/7589.

References
