RecTOR: A New and Efficient Method for Dynamic Network Reconfiguration

Åshild Grønstad Solheim, Olav Lysne, and Tor Skeie

Networks and Distributed Systems Group, Simula Research Laboratory
Lysaker, Norway
Department of Informatics
University of Oslo
Oslo, Norway

Abstract. Reconfiguration of an interconnection network is fundamental for the provisioning of a reliable service. Current reconfiguration methods either include deadlock-avoidance mechanisms that impose performance penalties during the reconfiguration, or are tied to the Up*/Down* routing algorithm which achieves relatively low performance. In addition, some of the methods require complex network switches, and some are limited to distributed routing systems. This paper presents a new dynamic reconfiguration method, RecTOR, which ensures deadlock-freedom during the reconfiguration without causing performance degradation such as increased latency or decreased throughput. Moreover, it is based on a simple concept, is easy to implement, is applicable for both source and distributed routing systems, and assumes Transition-Oriented Routing which achieves excellent performance. Our simulation results confirm that RecTOR supports a better network service to the applications than Overlapping Reconfiguration does.

1 Introduction

Reliable interconnection networks [1] are essential for the operation of current high-performance computing systems. An important challenge in the effort to support a reliable network service is the ability to efficiently restore a coherent routing function when a change has occurred in the interconnection network's topology. Such a change in topology could be a result of an unplanned fault in one of the network's components, and, as the size of systems grow, the probability of a failing component increases. Furthermore, planned system updates, where network components are removed or added, could also cause changes in the topology. Regardless of the cause of the topology change, the disturbance of the network service provided to the running applications should be minimized.

When a change has occurred, a new routing function must be calculated for the resulting topology, and we refer to the transition from the old routing function to the new one as *reconfiguration*. A main challenge related to reconfiguration is deadlock-avoidance. The transition from one routing function to another may result in deadlock even if each routing function is deadlock-free, as packets that

H. Sips, D. Epema, and H.-X. Lin (Eds.): Euro-Par 2009, LNCS 5704, pp. 1052–1064, 2009.

belong to one of the routing functions may take turns that are not allowed in the other [2].

Most *static reconfiguration* methods [3,4,5,6] do not allow application traffic into the network during the reconfiguration. This has an obvious negative impact on the network service availability, but eliminates the risk of deadlock, as packets routed according to only one of the routing functions are present in the network at a time.

A number of studies [7,8,9,10,2,11,12,13,14,15] have focused on *dynamic reconfiguration*, which allows application traffic into the network during reconfiguration, and thereby aims at supporting a better network service than static reconfiguration do. The studies listed above present general approaches that are not tied to particular network technologies. Other approaches target a specific technology such as InfiniBand [16] (see e.g. [17,18]).

Partial Progressive Reconfiguration (PPR) [8] and Close Graphs (CG) [15] are based on the Up*/Down* [3] routing algorithm, and both restore a correct Up*/Down* graph from a graph that has been rendered incorrect by a topology change. PPR changes the direction of a subset of the network links through a sequence of partial routing table updates, and is only useful in distributed routing systems. CG restricts the new Up*/Down* graph such that packets belonging to the old and new routing functions can coexist in the network without causing deadlocks, and thereby supports an unaffected network service during reconfiguration. Neither PPR nor CG requires virtual channels (VCs) to achieve deadlock-freedom. However, both methods depend on complicated procedures to establish the new routing function. Furthermore, the Up*/Down* routing algorithm achieves relatively low performance (Section 3 gives a brief description of Up*/Down* and its performance issues).

Double Scheme [10] avoids deadlock during reconfiguration by utilizing two sets of VCs in order to separate packets that are routed according to the two routing functions. Each set of VCs accepts application traffic in turn while the other is being drained and reconfigured. Double Scheme can be used between any pair of routing algorithms. However, in order to avoid deadlock, Double Scheme generally requires a number of available VCs that resembles the sum of the number of VCs required by the old and new routing functions.

Overlapping Reconfiguration (OR) is perhaps the most efficient of the proposed methods to reconfigure an interconnection network. It has been categorized both as a dynamic reconfiguration scheme [19] and as a static reconfiguration scheme with overlapping phases [20]. OR can be used between any pair of routing algorithms, ensures in-order packet delivery, and does not depend on the availability of VCs. Originally, OR could only be applied in distributed routing systems, but an adaptation was recently proposed for source routed systems [21]. OR requires relatively complex network switches as each switch must hold and process information regarding the reception and transmission of tokens (special packets used for deadlock-avoidance). The tokens regulate the forwarding of packets that are routed according to the new routing function, and this regulation causes increased latency and reduced throughput during the reconfiguration.

Furthermore, OR demands that two sets of routing tables are kept during the reconfiguration.

This paper presents RecTOR, a new dynamic reconfiguration method which does not impose performance penalties during the change-over from one routing function to another. RecTOR is based on a simple principle that ensures deadlock-freedom while packets that follow either routing function can coexist in the network without restrictions. It does not require complex network switches, does not need more VCs than a routing function does, and is useful for both source and distributed routing systems. RecTOR assumes Transition-Oriented Routing (TOR) [22], a topology agnostic[1] routing algorithm that, given sound path selection, matches the performance of the topology specific Dimension-Order Routing (DOR) in meshes and tori. Our performance evaluation shows that, when compared to OR, RecTOR supports a superior network service during the reconfiguration.

As OR is used in the performance evaluation, the algorithm is detailed in Section 2. RecTOR is based on TOR, which is outlined in Section 3 through a comparison with Up*/Down* (which is also used in the performance evaluation). RecTOR is presented in Section 4, and its performance is evaluated in Sections 5 and 6. Section 7 presents our conclusions.

2 The OR Algorithm

OR uses a special packet called a *token* to prevent that deadlock occurs during the transition from one deadlock-free routing function, R_{old}, to another, R_{new}. A packet must be routed from source to destination according to only one of the routing functions. Let us refer to packets that follow R_{old} and R_{new} as packets$_{old}$ and packets$_{new}$, respectively. OR uses the token to separate packets$_{old}$ and packets$_{new}$ on each (physical or virtual) communication channel, such that each channel first transmits packets$_{old}$, then the token, and finally packets$_{new}$.

The token propagation procedure is as follows:

- Each *injection channel* inserts a token between the last packet$_{old}$ and the first packet$_{new}$.
- A *switch input channel* routes packets according to R_{old} until the token is processed, and thereafter routes packets according to R_{new}. After having processed the token, an input channel must only forward packets to output channels that have transmitted the token.
- *A switch output channel*, c_o, must not transmit the token until all input channels, c_i, for which dependencies[2] exist according to R_{old} from c_i to c_o, have processed the token.

For further details, see [20].

[1] A topology agnostic routing algorithm does not presuppose a particular topology.

[2] If a packet may use a channel c_b immediately after a channel c_a there is a channel dependency from c_a to c_b.

3 TOR versus Up*/Down*

Both TOR and Up*/Down* assign up and down directions to all the links in the network to form a directed acyclic graph (DAG) rooted in one of the switches.[3] The path selection is, however, different for the two algorithms.

According to [23] deadlocks cannot form if cyclic channel dependencies are prevented. Up*/Down* breaks all cycles by prohibiting the turn from a down-link to an up-link. VCs are not required. With Up*/Down* all other switches/endnodes[4] can reach the root (the only switch with no outgoing up-links) following one or more up-links, and the root can reach all other switches/endnodes following one or more down-links.

TOR selects paths without regard to the underlying DAG. For TOR, the purpose of the DAG is solely to identify the *breakpoints* – the turns where the cycles must be broken. As for Up*/Down*, the breakpoints are the down-link to up-link turns. TOR prevents deadlock by requiring that when a packet crosses a breakpoint (traverses from a down-link to an up-link), it makes a transition to the next higher VC. Thus, TOR supports a flexible shortest path routing, and can achieve significantly higher performance than Up*/Down* achieves. A main drawback of Up*/Down* is that the area around the root tends to become a hotspot. In addition, a legal route from one switch/endnode to another may be significantly longer than the shortest path in the physical topology.

For further details on TOR and Up*/Down*, see [22] and [3], respectively.

4 RecTOR

RecTOR is based on the following observations concerning TOR. During the operation of a network, changes in the topology can be reflected in the DAG. TOR selects paths independently of the underlying DAG (which sole purpose is to define the breakpoints that decide VC-transitions). Thus, after a topology change, a set of new paths that restores connectivity can always be found, provided that the topology is still physically connected.

Assume that G_{old} and G_{new} are the DAGs that apply before and after a topology change, respectively. In order to ensure that packets routed according to an old and new routing function can coexist in the network without causing deadlock, RecTOR makes only one assumption on the evolution of the DAG: *No breakpoint must be moved* during the transition from G_{old} to G_{new}. Thus, an edge that persists from G_{old} to G_{new} must keep its (up or down) direction. However, breakpoints (and turns in general) can be removed or added as vertexes and edges are removed or added, respectively.

Assume that TOR is used, and that a deadlock-free routing function, R_{old} (which includes VC-transitions according to G_{old}), applies when an unplanned

[3] Links and switches are represented in the DAG by edges and vertexes, respectively.
[4] An endnode is a compute node that generates and processes packets.

or planned topology change occurs.[5] Then, RecTOR prescribes the following procedure to reconfigure the network:

1. Update the underlying DAG to reflect the change in topology. If a link or switch was removed, simply remove the corresponding edge or vertex (including its connecting edges), respectively. If a link or switch was added, add an edge or vertex (including its connecting edges), respectively. Avoid the introduction of cycles when assigning directions to newly added edges (see the Up*/Down* method).
2. Let TOR calculate a new deadlock-free routing function, R_{new} as follows: First, select a new set of paths which restore connectivity. Then, for each path, insert a transition to the next higher VC wherever the path crosses a breakpoint in G_{new}.
3. R_{new} can be applied instantly – packets routed according to R_{old} and R_{new} (or both) can coexist in the network without causing deadlock.

With regard to step 3 above, there is a risk of a packet looping if the packet are routed according to R_{old} in some of the switches and R_{new} in others (which could happen if the system uses distributed routing). Such looping could cause packet loss, as a packet that has reached the highest available VC and still needs to make another VC-transition must be rejected. A simple approach that prevents this problem implies that each switch holds routing tables for both R_{old} and R_{new} during the reconfiguration, and that each packet is tagged to indicate which of the routing functions should be used. However, a better solution could be adopted from the Internet research community, where several studies (e.g. [24]) have focused on preventing packets from looping during the update of routing tables. Like e.g. Double Scheme, RecTOR cannot guarantee in-order packet delivery during reconfiguration.

As deadlock avoidance is an inherent challenge in dynamic reconfiguration, we include and prove Lemma 1.

Lemma 1. *RecTOR provides deadlock-free reconfiguration.*

Proof. The proof is by contradiction. Assume that a deadlocked set of packets, S_d, is a set of packets where none of the packets can advance before another packet in the set advances. Assume also that a reconfiguration from R_{old} to R_{new}, where RecTOR is applied, results in some non-empty S_d.

Both R_{old} and R_{new} are, by themselves, deadlock-free. Thus, S_d must include at least one packet that is taking a turn which is present in both G_{old} and G_{new}, and which is either a breakpoint in G_{old} and not a breakpoint in G_{new}, or a breakpoint in G_{new} and not a breakpoint in G_{old}. In either case some breakpoint must have been moved during the transition from G_{old} to G_{new}, which contradicts the premise of RecTOR. □

[5] The change detection mechanism is outside the scope of RecTOR.

5 Experiment Setup

In order to compare the performance of RecTOR with the performance of OR, we conducted a number of experiments where a link fault and, subsequently, a switch fault were introduced and handled by reconfiguration. OR can be used between any pair of routing algorithms, and we consider two different alternatives. In the first case (referred to as OR_{TOR}), TOR is used. In the second case (referred to as $OR_{DOR/UD}$), DOR is used initially (for the fault-free topology) whereas Up*/Down* is used after the first network component has failed.

Whereas TOR and Up*/Down* are topology agnostic routing algorithms, DOR only works for fault-free meshes and tori. DOR avoids deadlock by first routing a packet in the X-dimension until the offset in this dimension is zero. Thereafter the packet is routed in the Y-dimension until it reaches its destination. For a torus topology, DOR needs two VCs for deadlock avoidance [23].

TOR can calculate shortest path routes in a number of different ways. In these experiments an out-port in the X-dimension is preferred over an out-port in the Y-dimension in every intermediate switch. For a fault-free mesh or torus, this gives similar paths as DOR, although the VC-use is different for many of the paths. In these experiments, the switch in the upper left corner of the mesh is the root of the Up*/Down* graph.

The simulator model was developed in the J-Sim [25] environment. We consider both mesh and torus topologies[6] of size 8×8 and 16×16. In our experiments one endnode is connected to every switch. The packet size is 256 bytes, and both an ingress and egress buffer of a switch port can hold 6 packets per VC. The model applies virtual cut-through switching and credit-based flow control. A transmission queue in an endnode has space for 12 packets per VC, and overflows when the network cannot deliver packets at the rate they are injected. Packets are immediately removed upon reaching their destination endnode. The number of VCs available is 4. Each routing algorithm, TOR, DOR or Up*/Down*, evenly distributes the paths among the available VCs in order to achieve a balanced load.

A transparent synthetic workload model is applied. For the packet injection rate a normal approximation of the Poisson distribution is used. We study two different traffic patterns: A uniform destination address distribution, and a hotspot traffic pattern where 80% of the packets are destined for a randomly selected hotspot node whereas the remaining 20% of the packets are uniformly distributed.

In order to ensure relevant load levels for the experiments, we initially identified the saturation point (the load level where the transmission queues of the endnodes start to overflow) for each of the three routing algorithms.

For uniform traffic, Up*/Down* has the lowest saturation point (Sat_{min}) of the three routing algorithms, whereas the highest saturation point (Sat_{max}) is achieved for TOR and DOR for the mesh topology, and for DOR for the torus

[6] Among the upper ten supercomputers in the Top500 list [26], both mesh and torus topologies are represented.

topology. We selected three load levels of focus, where the lowest, medium and highest load levels correspond to 90% of Sat_{min}, the center between Sat_{min} and Sat_{max}, and 110% of Sat_{max}, respectively.

For hotspot traffic, the saturation point (Sat) is the same for all three routing algorithms (as the saturation point is mainly decided by the congestion that results from 80% of the traffic being directed towards one of the endnodes). Two load levels were in focus, where the lowest and highest level correspond to 80% and 110%, respectively, of Sat.

In order to remove initial transients, data collection is not started for an experiment until 50000 cycles have been run (time is measured in cycles – an abstract time unit)[7]. Data are collected for 100000 cycles. A random link (port) fault occurs after 6000 cycles and a random switch fault occurs after 28000 cycles. In each case a reconfiguration process is triggered.

Using OR, a traffic source (endnode) injects a token to indicate that no packets routed according to R_{old} will follow. The change-over from one routing function to another could be fully synchronized if all traffic sources performed the change simultaneously. It is well known that, due to such factors as clock skew or reception of routing or control information at different times, such synchronization is hard to achieve. In order to compare RecTOR and OR for different degrees of synchronization, we use source routing and let the endnodes perform the change-over from R_{old} to R_{new} as follows. When all endnodes have been notified to initialize reconfiguration, each endnode draws its change-over time, t_{change}, from a normal distribution with a mean of 500 cycles and where the standard deviation is a simulation parameter, $change_{std}$. An endnode starts a timer according to t_{change} and continues injecting $packets_{old}$ until the timer expires, then, if OR is used, injects the token, and thereafter injects $packets_{new}$. The higher $change_{std}$ is, the more unsynchronized the change of routing function becomes. In these experiments, $change_{std}$ assumes the values 0 and 100, where the former value represents the fully synchronized case.

We consider the metrics Thr_t and Lat_t which result from the division of the data collection period into 500 time intervals, each with a duration of 200 cycles. For a time interval int, Thr_t is the number of packets that are generated by any endnode in int and that subsequently reach their destination endnode. Lat_t for int is the average latency of all packets that are generated by any endnode in int and that subsequently reach their destination endnode. The latency for a single packet is the time that elapses from when the packet is generated and injected into a transmission queue in the source endnode until the packet is received by the destination endnode. For each int the values for Thr_t and Lat_t are plotted in the middle of the interval, whereas in the same plots the start and end times of the reconfiguration period are plotted without regard to interval borders. For OR the reconfiguration starts when the first token is injected and ends when the last token is received by an endnode. For RecTOR the reconfiguration starts when the first endnode starts using the new routing function and ends when the last packet that belonged to the old routing function are removed from the network.

[7] E.g. a link speed of 10 Gbps gives a simulator cycle length of 102.4 ns.

The values presented are the mean values that result from 30 repetitions of each experiment. Each repetition is initialized by a different seed and applies a unique link (port) fault, switch fault, and hotspot node (in the case of hotspot traffic).

6 Results

Due to space limitations we could not display the plots from all our experiments. Therefore, a set of representative plots was selected and included.

For the uniform traffic pattern, Figure 1 compares RecTOR with OR_{TOR} and $OR_{DOR/UD}$ for a 8×8 mesh under low traffic load and a 8×8 torus under medium traffic load. The vertical lines depict the start and end times of reconfiguration for each of the three methods (some of the lines are plotted on top of each other). The reduction of Thr_t seen at times 6000 and 28000 for all three methods is due to packet loss in the faulty link and switch, respectively.

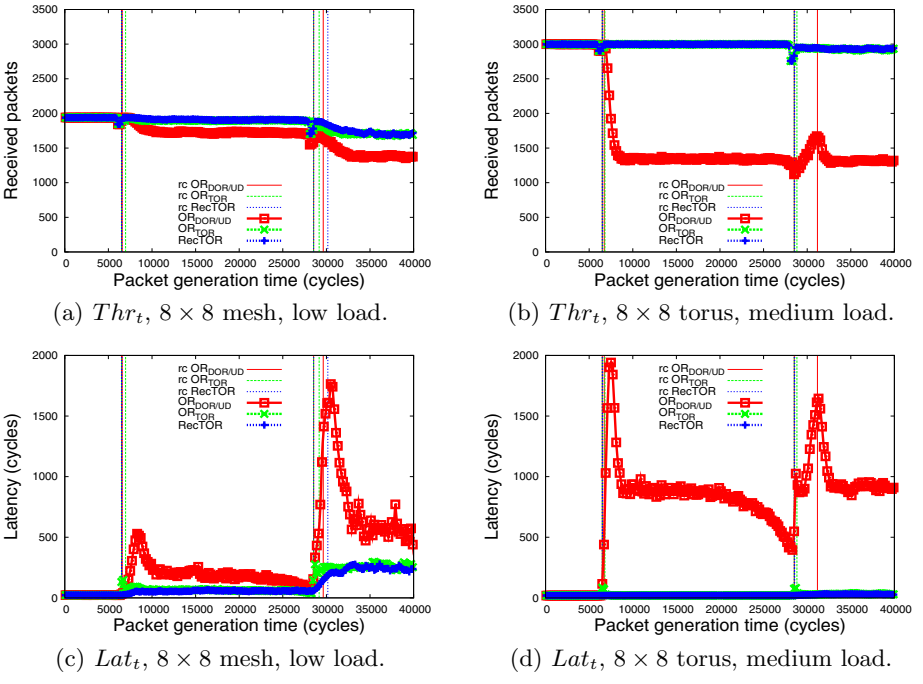
Figures 1(a) and 1(b) show that, after the first reconfiguration, RecTOR and OR_{TOR} achieve a significantly higher Thr_t than $OR_{DOR/UD}$ does. Likewise, Figures 1(c) and 1(d) show that, after the first reconfiguration, RecTOR and OR_{TOR} achieve a significantly lower Lat_t than $OR_{DOR/UD}$ does. The low load applied in Figures 1(a) and 1(c) is below saturation for Up*/Down* in the fault-free case. Nevertheless, the fault of only a single link causes a significant performance degradation for Up*/Down* when compared to TOR.

Using DOR for a fault-free mesh or torus, and then using Up*/Down* to calculate new routes as a fault occurs, is a relatively common approach.[8] However, at least for a traffic pattern that resembles uniform, Figure 1 clearly demonstrates the drawbacks of such an approach. RecTOR assumes TOR, which not only achieves better performance than Up*/Down* after a fault has occurred, but also matches the performance of DOR in the fault-free case. Furthermore, using only one routing algorithm simplifies the implementation.

OR often causes decreased throughput and increased latency during the reconfiguration as a number of $packets_{new}$ are held back by the token propagation procedure. For OR_{TOR} the characteristic troughs of the Thr_t curves and crests of the Lat_t curves were hardly noticeable in Figure 1 due to the large scale of these plots. Figure 2, on the other hand, clearly shows the advantages of RecTOR over OR_{TOR} in the case of uniform traffic. $OR_{DOR/UD}$ is not included as we have already concluded from Figure 1 that its performance is inferior to RecTOR and OR_{TOR}.

For a 16×16 mesh, Figure 2(a) shows Thr_t for medium load and Figure 2(c) shows Lat_t for low load. For a 16×16 torus, Figure 2(b) shows Thr_t for high load and Figure 2(d) shows Lat_t for medium load. All of these plots show that, for OR_{TOR}, the Thr_t and Lat_t curves have significant troughs and crests, respectively. For RecTOR, on the other hand, there are no troughs in the Thr_t curves nor crests in the Lat_t curves, as no restrictions are placed on the forwarding of packets during reconfiguration. The decreased throughput and increased latency observed for RecTOR after a fault are merely due to the reduced capacity of

[8] Remember that DOR only works for fault-free meshes and tori.



(a) Thr_t, 8×8 mesh, low load.

(b) Thr_t, 8×8 torus, medium load.

(c) Lat_t, 8×8 mesh, low load.

(d) Lat_t, 8×8 torus, medium load.

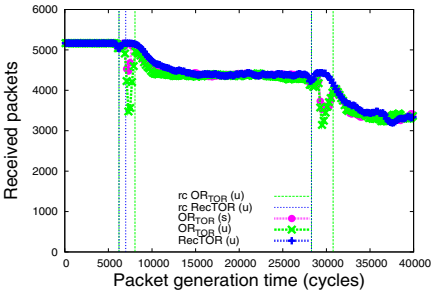
Fig. 1. Thr_t, Lat_t, and start/end times of reconfiguration (rc) for RecTOR, OR_{TOR} and $\mathrm{OR}_{DOR/UD}$ (uniform traffic, synchronized change-over to R_{new})

the network when packets can no longer be forwarded across the faulty link or switch (naturally, this effect is also observed for OR_{TOR}).

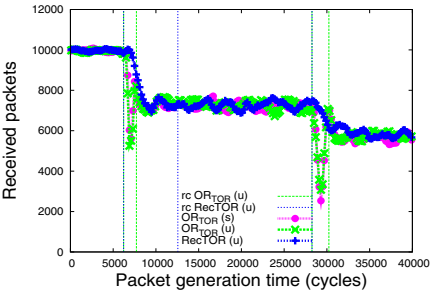
RecTOR performs equally well in the case of a less synchronized change-over ($change_{std} = 100$) from R_{old} to R_{new} as in the case of a fully synchronized change-over ($change_{std} = 0$). Therefore, Figure 2 includes only the less synchronized case for RecTOR, whereas both the fully and less synchronized cases are included for OR_{TOR}. Figure 2(b) shows that, for a traffic load well above saturation, the performance of OR_{TOR} does not depend on how synchronized the change-over from R_{old} to R_{new} is. For low and medium traffic load, on the other hand, Figures 2(a), 2(c) and 2(d) demonstrate that, for OR_{TOR}, the troughs of the Thr_t curves grow deeper, and the crests of the Lat_t curves grow higher as the change-over from R_{old} to R_{new} gets less synchronized.

In summary, Figure 2 shows that RecTOR provides a better network service to an application with a uniform communication pattern than OR does, and that the advantages become even more apparent if the change-over from R_{old} to R_{new} is not fully synchronized.

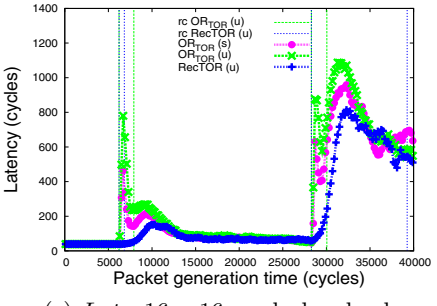
For the hotspot traffic pattern, Figure 3 compares RecTOR with OR_{TOR} and $\mathrm{OR}_{DOR/UD}$. Figures 3(a) and 3(b) show Lat_t for a 16×16 torus under low traffic load. Figure 3(a) shows a fully synchronized change-over from R_{old} to R_{new}, whereas Figure 3(b) shows the less synchronized change-over. For RecTOR, the



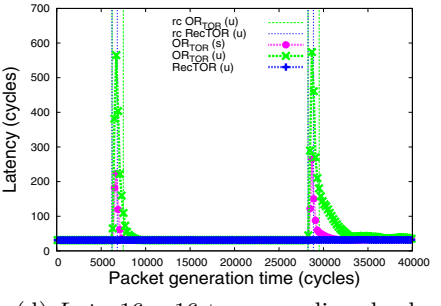
(a) Thr_t, 16×16 mesh, medium load.



(b) Thr_t, 16×16 torus, high load.



(c) Lat_t, 16×16 mesh, low load.



(d) Lat_t, 16×16 torus, medium load.

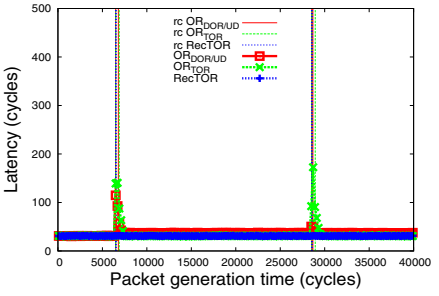
Fig. 2. Thr_t, Lat_t, and start/end times of reconfiguration (rc) for RecTOR and OR$_{TOR}$ under synchronized (s) and unsynchronized (u) change-over to R_{new} (uniform traffic)

Lat_t curves are smooth, without any crests, and, as we already know, the performance of RecTOR is independent of how synchronized the change-over to R_{new} is. For OR$_{TOR}$ and OR$_{DOR/UD}$, on the other hand, the crests of the Lat_t curves are pronounced. For both OR$_{TOR}$ and OR$_{DOR/UD}$ the heights of these crests increase significantly as the change-over to R_{new} gets less synchronized. Thus, as for uniform traffic, the advantages of RecTOR over OR become even more apparent when the change-over from R_{old} to R_{new} is not fully synchronized.

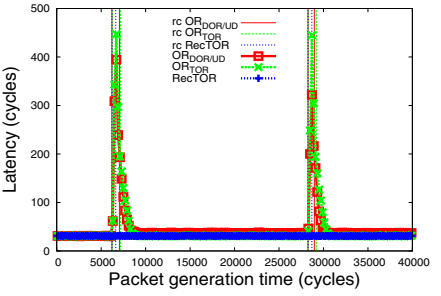
Figures 3(a) and 3(b) also indicate that after the first reconfiguration, Lat_t for OR$_{DOR/UD}$ is higher than for RecTOR and OR$_{TOR}$. This is due to inferior performance of Up*/Down* when compared to TOR, as was also demonstrated for uniform traffic in Figure 1.

For hotspot traffic, the packet throughput is limited due to 80% of the traffic being directed towards one particular node. This explains that in Figure 3(c), which shows Thr_t for a 8×8 mesh under high load, the characteristic troughs in the Thr_t curves for OR are barely visible.[9] We believe that the small increase in Thr_t immediately after the start of each reconfiguration in Figure 3(c) is due to more packets being accepted into the network when the new routes around a

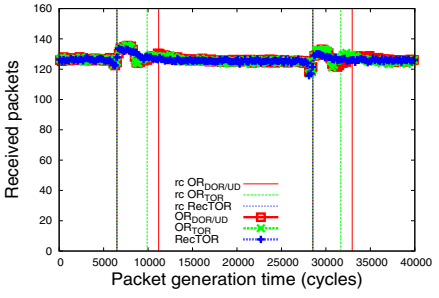
[9] For low traffic load, the Thr_t curves simply resembled horizontal lines, except for the packet loss in a faulty link or switch. Therefore, none of these plots were included.



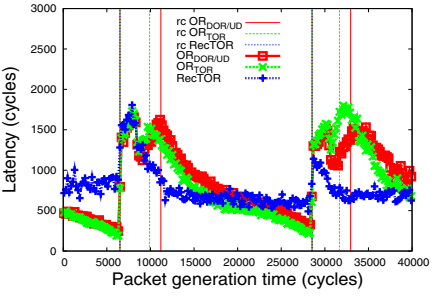
(a) Lat_t, 16×16 torus, low load, synchronized change-over to R_{new}.



(b) Lat_t, 16×16 torus, low load, unsynchronized change-over to R_{new}.



(c) Thr_t, 8×8 mesh, high load, synchronized change-over to R_{new}.



(d) Lat_t, 8×8 mesh, high load, synchronized change-over to R_{new}.

Fig. 3. Thr_t, Lat_t, and start/end times of reconfiguration (rc) for RecTOR, OR$_{TOR}$ and OR$_{DOR/UD}$ (hotspot traffic)

faulty link or switch are taken into use. Figure 3(d) shows that such an increase in Thr_t corresponds to an increase in Lat_t for all three reconfiguration methods. Figure 3(d) demonstrates that, for a load level above saturation, OR experiences more fluctuations in Lat_t than RecTOR does. For OR$_{TOR}$ and OR$_{DOR/UD}$ the Lat_t decreases for packets$_{old}$ injected over a period of time before reconfiguration. This effect is caused by a number of packets$_{new}$ being held back in the switches during the token propagation procedure (which reduces the network load, and packets$_{old}$ thus experience reduced latency). On the other hand, some significant crests in the Lat_t curves for OR$_{TOR}$ and OR$_{DOR/UD}$ are also results of the token propagation procedure. Figures 3(c) and 3(d) represent an extreme scenario (a heavy hotspot pattern in combination with a workload well above saturation). However, even in this case, RecTOR supports a more stable network service than OR does.

7 Conclusion

Existing methods for reconfiguration of an interconnection network have a number of limitations, such as dependence on deadlock-avoidance mechanisms that

impose performance penalties; complicated procedures; or dependence on the Up*/Down* routing algorithm which achieves low performance. Some of the methods require complex network switches, or are only applicable for distributed routing systems.

This paper presents RecTOR, a new dynamic reconfiguration method, which is useful both for source and distributed routing systems, and which does not require complex network switches. RecTOR is based on a simple principle that, while ensuring deadlock-freedom, allows packets routed according to an old and a new routing function to coexist in the network without restrictions. Thus, unlike e.g. OR, RecTOR does not cause degraded performance during the reconfiguration.

Our performance evaluation shows that, during the reconfiguration, RecTOR supports a better network service than OR does, both to applications with uniform and hotspot communication patterns. Complete synchronization of the change-over from an old to a new routing function is hard to achieve (due to such factors as clock skew or reception of routing or control information at different times). The simulation results show that the advantages of RecTOR over OR become even more evident as the change-over gets less synchronized. Furthermore, our results demonstrate the limitations of the common approach that implies using DOR in a fault-free mesh or torus, and then using Up*/Down* routing if a fault occurs. Using RecTOR, only one routing algorithm is needed – TOR, a topology agnostic routing algorithm that not only outperforms Up*/Down*, but also matches the performance of DOR.

Currently, RecTOR appears as the most efficient reconfiguration method for systems that accept out-of-order packet delivery and have virtual channels available for the routing function.

References

1. Duato, J., Yalamanchili, S., Ni, L.: Interconnection Networks: An Engineering Approach. Morgan Kaufmann Publishers, San Francisco (2003)
2. Duato, J., Lysne, O., Pang, R., Pinkston, T.M.: Part I: A theory for deadlock-free dynamic network reconfiguration. IEEE Trans. Parallel and Distributed Systems 16(5), 412–427 (2005)
3. Schroeder, M.D., et al.: Autonet: A high-speed, self-configuring local area network using point-to-point links. SRC Research Report 59, Digital Equipment Corporation (1990)
4. Rodeheffer, T.L., Schroeder, M.D.: Automatic reconfiguration in Autonet. In: 13th ACM Symp. Operating Systems Principles, pp. 183–197 (1991)
5. Boden, N.J., et al.: Myrinet: A gigabit-per-second local area network. IEEE Micro. 15(1), 29–36 (1995)
6. Teodosiu, D., et al.: Hardware fault containment in scalable shared-memory multiprocessors. SIGARCH Computer Architecture News 25(2), 73–84 (1997)
7. Lysne, O., Duato, J.: Fast dynamic reconfiguration in irregular networks. In: Int'l. Conf. Parallel Processing, pp. 449–458 (2000)
8. Casado, R., Bermúdez, A., Duato, J., Quiles, F.J., Sánchez, J.L.: A protocol for deadlock-free dynamic reconfiguration in high-speed local area networks. IEEE Trans. Parallel and Distributed Systems 12(2), 115–132 (2001)

9. Natchev, N., Avresky, D., Shurbanov, V.: Dynamic reconfiguration in high-speed computer clusters. In: 3rd Int'l. Conf. Cluster Computing, pp. 380–387 (2001)
10. Pinkston, T.M., Pang, R., Duato, J.: Deadlock-free dynamic reconfiguration schemes for increased network dependability. IEEE Trans. Parallel and Distributed Systems 14(8), 780–794 (2003)
11. Lysne, O., Pinkston, T.M., Duato, J.: Part II: A methodology for developing deadlock-free dynamic network reconfiguration processes. IEEE Trans. Parallel and Distributed Systems 16(5), 428–443 (2005)
12. Avresky, D., Natchev, N.: Dynamic reconfiguration in computer clusters with irregular topologies in the presence of multiple node and link failures. IEEE Trans. Computers 54(5), 603–615 (2005)
13. Acosta, J.R., Avresky, D.R.: Dynamic network reconfiguration in presence of multiple node and link failures using autonomous agents. In: 2005 Int'l. Conf. Collaborative Computing: Networking, Applications and Worksharing (2005)
14. Acosta, J.R., Avresky, D.R.: Intelligent dynamic network reconfiguration. In: Int'l. Parallel and Distributed Processing Symp. (2007)
15. Robles-Gómez, A., Bermúdez, A., Casado, R., Solheim, Å.G.: Deadlock-free dynamic network reconfiguration based on close Up*/Down* graphs. In: Luque, E., Margalef, T., Benítez, D. (eds.) Euro-Par 2008. LNCS, vol. 5168, pp. 940–949. Springer, Heidelberg (2008)
16. InfiniBand Trade Association: InfiniBand Architecture Specification v. 1.2.1 (2007), `http://www.infinibandta.org/specs`
17. Zafar, B., Pinkston, T.M., Bermúdez, A., Duato, J.: Deadlock-free dynamic reconfiguration over InfiniBand networks. Int'l. Jrnl. Parallel, Emergent and Distributed Systems 19(2), 127–143 (2004)
18. Bermúdez, A., Casado, R., Quiles, F.J., Duato, J.: Handling topology changes in InfiniBand. IEEE Trans. Parallel and Distributed Systems 18(2), 172–185 (2007)
19. Lysne, O., et al.: Simple deadlock-free dynamic network reconfiguration. In: 11th Int'l. Conf. High Performance Computing, pp. 504–515 (2004)
20. Lysne, O., et al.: An efficient and deadlock-free network reconfiguration protocol. IEEE Trans. Computers 57(6), 762–779 (2008)
21. Solheim, Å.G., et al.: Efficient and deadlock-free reconfiguration for source routed networks. In: 9th Worksh. Communication Architecture for Clusters (2009)
22. Sancho, J.C., Robles, A., Flich, J., López, P., Duato, J.: Effective methodology for deadlock-free minimal routing in InfiniBand networks. In: Int'l. Conf. Parallel Processing, pp. 409–418 (2002)
23. Dally, W.J., Seitz, C.L.: Deadlock-free message routing in multiprocessor interconnection networks. IEEE Trans. Computers 36(5), 547–553 (1987)
24. Francois, P., Bonaventure, O.: Avoiding transient loops during IGP convergence in IP networks. In: 24th IEEE INFOCOM, vol. 1, pp. 237–247 (2005)
25. Tyan, H.-Y.: Design, Realization and Evaluation of a Component-Based Compositional Software Architecture for Network Simulation. PhD thesis, Ohio State University (2002)
26. Top 500 project: Top 500 Supercomputer Sites (November 2008), `http://top500.org`