Improving the Performance of Real-Time Communication Services on High-Speed LANs under Topology Changes

J. Fernández, J. M. García
Dpto. Ingeniería y Tecnología de Computadores
Universidad de Murcia
30071 Murcia (Spain)
{*peinador,jmgarcia*}*@ditec.um.es*

J. Duato
Dpto. Informática de Sistemas y Computadores
Universidad Politécnica de Valencia
46071 Valencia (Spain)
jduato@gap.upv.es

Abstract

Topology changes, such as switches being turned on/off, hot expansion, hot replacement or link re-mapping, are very likely to occur in NOWs and clusters. Moreover, topology changes are much more frequent than faults. However, their impact on real-time communications has not been considered a major problem up to now, mostly because they are not feasible in traditional environments, such as massive parallel processors (MPPs), which have fixed topologies. They are supported and handled by some current and future interconnects, such as Myrinet or Infiniband. Unfortunately, they do not include support for real-time communications in the presence of topology changes.

In this paper, we propose and evaluate a new protocol that provides topology change- and fault-tolerant real-time communication services on NOWs and clusters. This protocol overcomes the main drawback of our previously proposed protocol, called Dynamically Re-established Real-Time Channels (DRRTC), which is physically limited by the number of virtual channels per port. The new protocol allows different real-time channels to share the same virtual channel. In this way, the new protocol allows to establish a greater number of real-time channels than the previous one. Moreover, its only limitation is the bandwidth devoted to real-time traffic. However, this introduces two new problems that are successfully managed by the new protocol: the existence of cyclic dependencies among different real-time channels and the increased complexity of deadline requirements. We present and analyze the performance evaluation results when a single switch or a single link is deactivated/activated for different topologies and workloads. The new protocol overwhelms the DRRTC protocol while guaranteeing deadline requirements and channel recovery.

Keywords - NOWs, clusters, real-time services, topology change tolerance, dynamic reconfiguration

1. Introduction

In the past few years, networks of workstations (NOWs) and clusters, based on off-the-shelf commodity components like workstations, PCs, and high-speed local-area networks (LANs), have emerged as a serious alternative to massive parallel processors (MPPs) and are the most cost-effective platform for high-performance servers [20]. In fact, they are becoming the main infrastructure for science and engineering distributed processing. Moreover, they are acquiring a prevalent role in providing support for many other applications such as web servers or distributed databases. However, distributed real-time processing on NOWs and clusters is still a pending issue. This is because of the lack of schemes that are able to provide dependable real-time services on NOWs.

Distributed real-time applications impose strict conditions on traffic such as bounded delivery time (deadline) or guaranteed bandwidth [3]. In order to provide real-time communication, real-time channels [16] establish unidirectional connections among source and destination hosts. Once a real-time channel has been established, that is, resource reservation has finished, maximum delivery time and bandwidth are guaranteed.

Topology changes, such as switches being turned on/off or link re-mapping, are very likely to occur in NOWs and clusters. Moreover, topology changes are much more frequent than faults. However, their impact on real-time communications has not been considered a major problem up to now, mostly because they are not feasible in traditional environments, such as massive parallel processors (MPPs), which have fixed topologies. Topology changes may tear down existing real-time channels and degrade network ability to establish new real-time channels because the routing tables are not up-to-date. Therefore, not all the available resources can be fully exploited. To make the best use of resources, every time a topology change or fault occurs,

routing tables must be updated to reflect the new configuration. Finally, topology changes are supported and handled by some current and future interconnects, such as Myrinet [5] or Infiniband [15]. Unfortunately, they do not include support for real-time communications in the presence of topology changes.

We proposed a previous protocol, called DRRTC [11, 12], that provides topology change- and fault-tolerant real-time communication services on NOWs while still providing best-effort ones. The protocol was based on real-time channels [16] with single backup [14] channels and dynamic reconfiguration [1, 6]. Real-time channels provided real-time communications, single backup channels provided single-fault tolerance, and dynamic reconfiguration provided topology change tolerance and tolerance to additional faults. The main drawback of the DRRTC protocol is the small number of real-time channels that can be supported. Each real-time channel reserves a virtual channel per output port in every switch it goes through and the number of virtual channels per physical port is limited [12]. In our experiments, we reached up to four real-time channels per host under certain conditions [12]. The motivation of this work is to improve the previous protocol by eliminating the physical constraint imposed by the number of available virtual channels. To eliminate that physical constraint, the new protocol allows several real-time channels to share a given virtual channel. This introduces two new problems that are successfully managed by the new protocol: the existence of cyclic dependencies among different real-time channels and the increased complexity of deadline requirements.

We present and analyze the performance evaluation results when a single switch or a single link is deactivated/activated for different topologies and workloads. The results show that the new protocol allows up to ten times the original number of supported real-time channels while guaranteeing deadline requirements and channel recovery. Furthermore, with the new protocol, topology change tolerance is only limited by the available bandwidth to establish real-time channels, as well as by the topology connectivity, because the physical constraint imposed by the number of virtual channels is eliminated.

The rest of this paper is organized as follows. The next section describes related work. Sect. 3 outlines the new protocol. In Sect. 4, the design trade-offs for the new protocol are analyzed. The experimental testbed is depicted in Sect. 5. In Sect. 6, the performance evaluation results are shown. Finally, we present our conclusions and feasible ways of future work.

2 Related work

Faults have been traditionally considered a major problem in distributed real-time processing since they may interrupt real-time communications. Thus, several researchers have proposed efficient solutions based on the reservation of additional resources. The *Backup Channel Protocol* (BCP), developed by Shin *et al.* [14] and based on real-time channels [16], performs recovery from faults by means of the reservation of additional resources (backup channels). In this approach, the maximum number of admissible faults depends on the maximum number of alternative paths provided by the routing function to establish the backup channels. Moreover, topology change tolerance is not provided.

Static reconfiguration techniques (Autonet [22] and Myrinet with GM [18]) stop user traffic to update routing tables. Although reconfigurations are not frequent, they can considerably degrade performance [22] and real-time constraints can not be met because of traffic disruption.

Dynamic reconfiguration, recently proposed by Duato *et al.* [1, 6], assimilates topology changes by updating routing tables without stopping traffic. In this way, the negative effects of static reconfiguration are avoided [6]. The protocol guarantees that the global routing algorithm remains deadlock-free at any time. Note that dynamic reconfiguration by itself provides neither quality of service nor real-time services, but it provides support for an additional mechanism designed to meet real-time requirements. Pinkston *et al.* [19] developed a simple but effective strategy for dynamic reconfiguration in networks with virtual channels. Lysne *et al.* [17] aim at reducing the scope of reconfiguration by identifying a usually small part of the network, the *skyline*, as the only part where a full reconfiguration is necessary. Avresky *et al.* [4] recently presented a new dynamic reconfiguration protocol, called *NetRec*, for high-speed LANs using wormhole routing.

The possibility of providing support for QoS in router architectures has been explored by a few researchers. A hybrid approach using two different types of switching mechanisms has been presented in [10, 21]. The Media Worm Router [25] explores the feasibility of providing QoS in wormhole switched routers. It aims at using multiple connections within each virtual channel to achieve soft guarantees.

3 Protocol

This section summarizes our previous protocol and describes the operation of the new protocol in an informal way. For further details about the protocol see [11, 12].

3.1 DRRTC protocol

In order to support real-time communications, we set up a primary channel and a single secondary channel[1] for each real-time channel. Once a real-time channel has been established, real-time messages flow through the primary channel from source host to destination host until the real-time channel is closed or the primary channel is broken down. When either a hot topology change or a fault breaks down a primary channel, real-time messages are redirected through its secondary channel. At the same time, the dynamic reconfiguration algorithm described in [1, 6] is triggered. The dynamic reconfiguration process updates routing tables in such a way that secondary channels could be re-established for the affected real-time channels as long as the network topology still provides an alternative physical path. The dynamic reconfiguration process does not affect the performance of real-time messages flowing through real-time channels because real-time traffic is allowed during reconfiguration, and real-time messages have the highest priority (see Subsection 4.1). After reconfiguration, a new secondary channel is allocated for each affected real-time channel regardless of whether the old secondary channel has become the new primary channel or the old secondary channel was broken down. The procedure for interrupted secondary channels is the same as for primary ones. However, in this case, real-time traffic is not affected. On the other hand, if several topology changes or faults concurrently occur, the dynamic reconfiguration protocol combines them into a single process [6], and thus, the protocol remains valid.

3.2 Real-time channel establishment

A *real-time channel* is a unidirectional connection between a pair of hosts H_1 and H_2. A real-time channel consists of a primary channel and a single secondary one. Each of them is a set $\{H_1, H_2, B, D, T\}$ where H_1 is the source host, H_2 is the destination host, B is the required bandwidth, D is the maximum admissible latency or deadline for real-time messages, and T is the type of channel, that is, primary or secondary. Enough resources are assigned to each channel in each switch along its path to meet its bandwidth and deadline requirements before real-time messages are transmitted. Real-time messages flow through the primary channel from H_1 to H_2 until the real-time channel is closed or the primary channel is broken down. In the meantime, the secondary channel remains idle and does not consume link bandwidth along its path from H_1 to H_2. Before transmitting real-time messages, H_1 must reserve the necessary resources for both the primary and the secondary channels. A best-effort message, called RTC_REQUEST, is sent from H_1 to H_2 to set up the primary channel. In each switch, the request message is processed to check for the availability of resources and to reserve them as we will see later. If there are not enough resources, a best-effort message, called RTC_RESPONSE(False), is returned to H_1. When a request message arrives at H_2, the primary channel is accepted if, and only if, the maximum latency for real-time messages along the channel path is shorter than channel deadline (see Subsection 4.3).

Then, H_2 sends a best-effort message, called RTC_RESPONSE(True), back to H_1. The path followed by the response messages may not be the same used by the request to establish the channel. If H_1 receives a false response message or channel timeout expires, resources are released by means of a RTC_MSG(Release) message that flows through the channel. Channel timeout allows H_1 to release resources when no response message is received. After a few cycles, H_1 will try to establish the primary channel again until it is established or a maximum number of attempts is reached. If H_1 receives a true response message, the secondary channel will be established likewise. A request is sent from H_1 to H_2 to set up the secondary channel. In each switch, the request message is processed to check for the availability of resources, to reserve them, and also to verify that the primary channel does not go through that switch. This is because the primary and the secondary channels must not share resources in order to maximize fault tolerance.[2] Once both the primary and the secondary channels have been successfully established, the real-time channel establishment process has finished. Otherwise, resources for both the primary and the secondary channels are released, and the real-time channel is rejected. Note that several channels could be concurrently established.

Finally, we are going to analyze the processing of requests in detail. First, for each possible output port provided by the routing function for a request message, we check for the availability of resources, that is, enough bandwidth. Output ports without enough resources are no longer considered. If the request corresponds to the secondary channel, it must also be verified that the primary channel does not go through the switch. To do this, each switch has a channel table (see Table 1) that keeps track of all channels that go through it. If there are not enough resources or the primary channel goes through the switch, a false response message is returned to H_1. Once an appropriate output port has been found, the channel is added to the channel table, resources are reserved, and the request is forwarded to the next switch. Note that selection of channel routes is distributed among all switches, that is, global information is not necessary to establish channels.

[1] The terms secondary and backup are used interchangeably throughout this paper.

[2] Initially, two NICs per host are assumed so that the primary and the secondary channels have to share neither switches nor links.

Table 1. Real-time channel table

Field	Meaning
CHANNEL	Channel identifier in the source host
HSSOURCE	Source host identifier
TYPE	Primary or Secondary
LinkIn	Input port
LinkOut	Output port

3.3 Real-time channel operation

After a real-time channel has been successfully established, H_1 begins to inject real-time messages, called RTC_MSG messages, through the primary channel. Real-time messages flow from H_1 to H_2 through the primary channel until the real-time channel is closed or the primary channel is broken down. In the former case, resources are released by means of two release messages (for the primary and secondary channels, respectively). In the latter case, real-time messages are redirected through the secondary channel and a new secondary channel will be allocated if possible. Note that if the secondary channel is broken down, real-time traffic is not affected.

3.4 Real-time channel recovery

Once a real-time channel has been set up and is transmitting real-time messages, we have to deal with the problem of channel recovery while still satisfying real-time requirements. Let us assume that a single link fails or is turned off. Next, the two adjacent switches detect the fault and determine the broken channels looking up their real-time channel tables. For each channel whose output port matches the broken link, a best-effort message, called RTC_REPORT, is sent to its corresponding source host. The switch remains in the releasing state until the corresponding release message is received. If report timeout expires, a report message is sent again. For each channel whose input port matches the broken link, a release message is sent to the destination host through the channel. Every time a report arrives at a source host for the primary or the secondary channels, a release message releases resources from that source host up to the previous switch. In the former case, real-time traffic is redirected through the secondary channel, that is, the secondary channel becomes the new primary channel. In the latter case, real-time messages continue flowing through the primary channel. In any case, after reconfiguration, the secondary channel will be re-established if possible.

At the same time that the adjacent switches detect the fault, the dynamic reconfiguration protocol described in [1, 6] is triggered. This process performs sequences of partial routing table updates to avoid stopping traffic, and trying to update routing tables in such a way that a secondary channel could be re-established for each affected real-time channel. After reconfiguration, a new secondary channel is allocated, if possible, for each affected real-time channel regardless of whether the old secondary channel has become the new primary channel or the old secondary channel was broken down.

4 Increasing the real-time capabilities

This section describes the main modifications we have introduced in our protocol in order to increase the number of supported real-time channels. A different architecture is needed to allow real-time channels to share a single virtual channel. Moreover, this has two relevant disadvantages. First, the existence of cyclic dependencies among real-time channels that could lead to a deadlocked configuration. Second, the increased complexity of deadline requirements. We present an elegant solution to solve both problems without reducing the real-time capabilities.

4.1 Switch architecture

Although a detailed hardware design is out of the scope of this paper, switch architecture is depicted (see Fig. 1(b)) to help readers to understand the new protocol that we propose. For comparison purposes, we also show the switch architecture used by the DRRTC protocol (see Fig. 1(a)). As shown in Fig. 1, both protocols use an input-buffered switch with virtual channels. Virtual cut-through is used because it may replace wormhole in the near future in NOWs [9]. Physical link bandwidth is 1.28 Gbps, and links are 8-bit wide.

In the DRRTC protocol, each output port has sixteen virtual channels so that thirteen RTC virtual channels can be reserved for real-time channels (see Fig. 1(a)). Each real-time channel flowing through an output port reserves a virtual channel. Thus, the maximum number of real-time channels that can be established through an output port is thirteen.

On the other hand, in the new protocol, each output port has four virtual channels so that a single virtual channel is shared among all real-time channels flowing through each output port (Fig. 1(b)). Consequently, we eliminate the constraint due to the limited number of virtual channels in order to increase the number of supported real-time channels. The use of virtual channels other than the RTC ones (see Subsection 4.2) is the same in both cases. Min and UD (Up*/Down* routing) virtual channels are used for fully adaptive minimal routing by best-effort traffic, and C/RTC is used by protocol control messages and all control messages generated during reconfigurations. To build a

deadlock-free routing function on Min and UD we use the methodology described in [23].

Primary and Secondary store the amount of bandwidth reserved by primary and secondary channels, respectively. RVC stores the reserved virtual channels. Note that RVC is not needed by the new protocol since a single virtual channel is used for all real-time channels. The Control Unit processes requests and all control messages generated by reconfiguration.

The Control Port allows switches to inject control messages. The Channel Table keeps track of all channels that go through the switch (see Table 1). In the DRRTC protocol, the input and output virtual channels for each real-time channel must be recorded in the table. The Virtual Channel Arbiter implements the link scheduling algorithm used to forward messages. The scheduling algorithm is based on the one of Infiniband [15]. Three levels of priority are defined: real-time traffic, control traffic and best-effort traffic. Whenever an output port becomes free, it checks for the existence of real-time messages requesting that port. It proceeds in a round-robin fashion on all the input ports. All packets at a priority level are sent before any packet at a lower priority level. To ensure forward progress on the best-effort virtual channels, the maximum reservable bandwidth for real-time traffic is bounded. The scheduling algorithm becomes simpler when using a shared virtual channel for real-time traffic since only virtual channel per port must be considered. Finally, we use a full crossbar in the modified DRRTC protocol because of the reduction in the number of virtual channels required.

4.2 Recovering from cyclic dependencies among real-time channels

The DRRTC protocol allocates a virtual channel for each real-time channel along its path from source to destination host. Meanwhile, the new protocol allows all the real-time channels to share the same virtual channel. Therefore, the secondary channels in the new protocol do not consume resources. Moreover, this simplifies the resource reservation process because a dedicated virtual channel is no longer needed. However, this simplification is not as simple as it could seem to be at first glance. Sharing a single virtual channel among all real-time channels flowing through each output port may introduce cyclic dependencies [7, 8] between RTC virtual channels and even lead to deadlock. This is because request messages are routed using adaptive routing so that the probability of finding a path with enough resources is maximized. As a consequence, RTC virtual channels may form a cycle in the channel dependency graph [7, 8]. In that case, the cycle cannot be broken using an escape channel [7, 8] because real-time messages must follow a fixed route from source host to destination host. Likewise, cycles could arise after reconfiguration between *old* real-time channels and *new* real-time channels. However, since the percentage of real-time traffic over the whole link bandwidth is very small (see Sect. 5), the occurrence of deadlocks is virtually eliminated [24]. Furthermore, the situation remains the same during reconfiguration, that is, deadlocks are very infrequent [13].

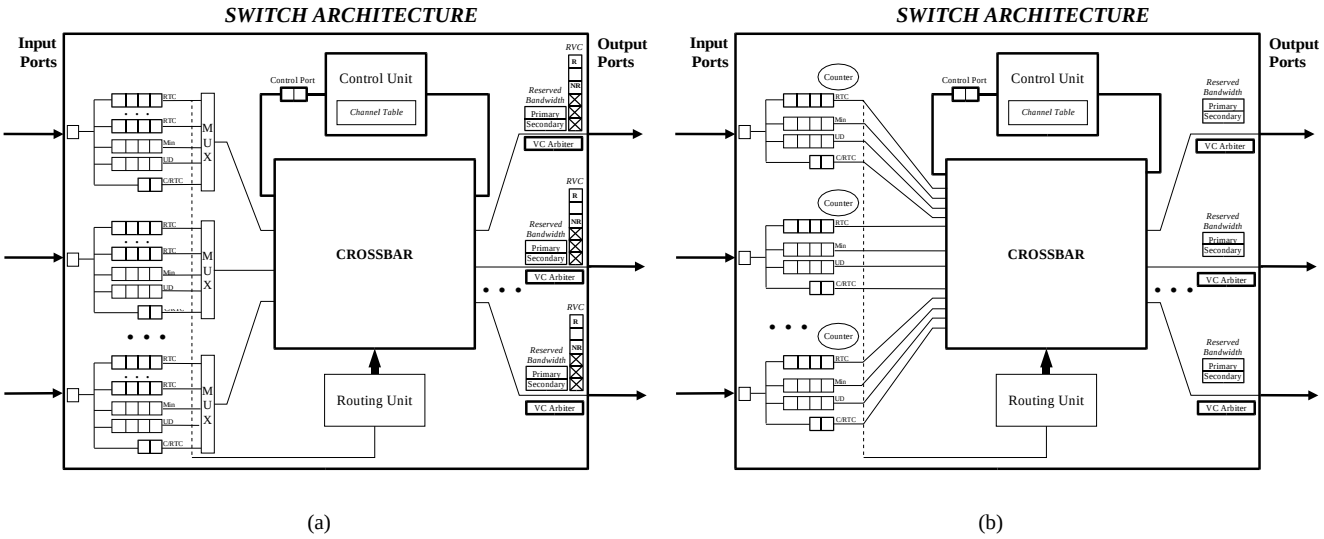
Table 2. Detected deadlocks as a function of the threshold used to trigger the counter for no topology changes and for deactivating the root node.

	Detected Deadlocks	
Counter (cycles)	No Change	Change
512 (2 packets)	10216	12718
768 (3 packets)	214	239
1024 (4 packets)	0	3
1280 (5 packets)	0	0
1536 (6 packets)	0	0
1792 (7 packets)	0	0
2048 (8 packets)	0	0

In spite of unlikeliness of deadlocks, the protocol has been modified to recover from deadlocked configurations. Each RTC virtual channel has an associated counter. This counter is set to zero every time a real-time packet goes through the crossbar. In this way, the counter represents the number of cycles that a real-time message is waiting to be forwarded. If the counter reaches a certain threshold, the packet is considered as to be deadlocked. Threshold value must be chosen according to the different number of real-time packets that may be forwarded before the blocked one to avoid detecting false deadlocks (see Table 2). In our experiments, a value of 1280 cycles would be enough. To recover from a deadlock, the real-time message is discarded and the corresponding real-time channel is torn down. The switch sends a RTC_DEADLOCK message to the source host and a release message to the destination host. When the RTC_DEADLOCK message arrives at the source host, a release message releases the reserved resources from that source host up to the sender switch. From this point, the protocol behaves in the same way as in the case of a topology change.

4.3 Evaluation of deadline requirements

A real-time channel is accepted if, and only if, the maximum latency for real-time messages is shorter than the channel deadline. To compute the maximum delay, we must consider the maximum delay in each switch times the number of switches crossed by the channel [2].



(a) (b)

Figure 1. 1(a) a virtual channel per real-time channel (DRRTC protocol) 1(b) a single virtual channel for all real-time channels (new protocol)

A real-time packet from a certain channel shares each virtual channel along its path with packets from different real-time channels. Thus, considering a FIFO behavior, all the packets that arrive before it must leave before it. Furthermore, all possible real-time packets from all the input ports must be accounted for. Therefore, the maximum number of real-time packets P to consider before a real-time packet crosses is:

$$P = \frac{B_{Size}}{P_{Size}} \times VCsPort \times Ports$$

where B_{Size} is the buffer size, P_{Size} is the packet size of real-time packets, $VCsPort$ is the number of RTC virtual channels per port and $Ports$ is the number of external ports per switch. In this case, $VCsPort$ is one and $Ports$ is seven. $Ports$ includes neither the Control Port nor the input port because the RTC virtual channel for real-time traffic has the highest priority, and the real-time channels are not allowed to go backwards. Hence, the maximum delay D is:

$$D = P \times P_{Size} \times Switches + C$$

or

$$D = B_{Size} \times Ports \times Switches + C$$

where C is the chosen counter threshold and $Switches$ is the number of switches crossed by the channel. We are assuming that a real-time packet might be delayed by one deadlocked packet at most. This assumption is even pessimistic due to the unlikeliness of deadlocks [13, 24]. Finally, note that deadline requirements for any real-time channel are independent from each other, that is, the establishment of a new real-time channel will not affect the performance of the previously established ones.

5 Experimental testbed

The cluster is composed of a set of switches connected by point-to-point links and hosts attached to switches through two network interface cards (NICs). In this way, we eliminate single points of failure. Network topologies were randomly generated and are completely irregular. However, some restrictions were applied for the sake of simplicity. First, all switches have eight ports, four ports connected to other switches and four ports connected to hosts. Second, two switches are not connected by more than one link. Finally, the assignment of hosts to switches was made so that the intersection of the sets of hosts connected to two switches is one host at most. This is because we want to avoid bottlenecks when re-establishing channels after reconfiguration.

Simulation was used instead of analytical modeling. Our simulator models the network at the packet level. Simulation results were generated from three irregular topologies (T1, T2, and T3) consisting of 64 switches and 128 hosts each one (2 NICs per host). Workload is composed of CBR real-time channels of 1 Mbps, whose deadline is equal to inter-arrival time (IAT), and best-effort messages. Best-effort message distribution is uniform and hosts inject messages into the network at 100 Mbps each. The maximum

amount of reservable bandwidth for real-time channels on each link is 10% of the total bandwidth, that is, 128 Mbps. Packet length of real-time messages is 256 bytes and packet length of best-effort messages is also 256 bytes. All hosts try to establish the same number of real-time channels. For different experiments, the number of real-time channels per host varies between two and four (2RTC, 3RTC and 4RTC) for the DRRTC protocol (dedicated virtual channels), and between twenty and forty (20RTC, 30RTC and 40RTC) for the new protocol (shared virtual channel). Therefore, the total number of real-time channels in the network is 256/2560, 384/3840 or 512/5120 according to the number of real-time channels per host. Destinations of channels are randomly chosen among all hosts in the network. Real-time channels are initially established during an interval of time proportional to the number of channels per host. Finally, a 2048 cycles counter was used to avoid false deadlock detection.

6 Performance evaluation

In this section, the performance of the new protocol is evaluated and compared with that of the DRRTC protocol [12]. In particular, we analyze what happens when a single switch or a single link is turned on/off.

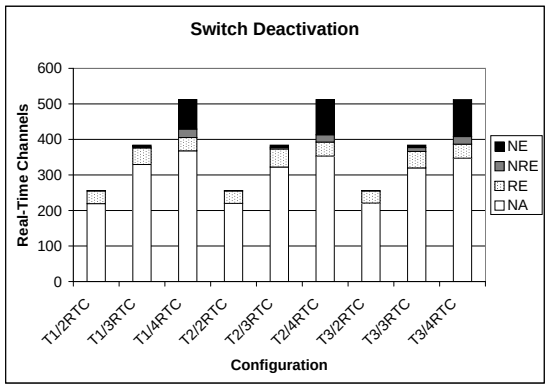
6.1 Switch deactivation and activation

In Fig. 2(a) and Fig. 2(b), we show the evolution of real-time channels for different configurations when a single switch is turned off for the DRRTC protocol and for the new one. Each column represents the total number of channels that hosts are trying to establish, that is, 256/2560, 384/3840 and 512/5120 for 2RTC/20RTC, 3RTC/30RTC and 4RTC/40RTC, respectively. *NE* corresponds to initially non-established channels and the rest corresponds to successfully established channels. As shown in Fig. 2(a) and Fig. 2(b), for all topologies, as the number of channels per host increases, the number of initially non-established channels increases too. In the new protocol, whereas 100% of channels are established for 20RTC per host, less than 1% of channels are non-established for 30RTC per host, and the percentage of non-established channels varies from 19.4% to 22.1% for 40RTC per host. The percentages are very similar in the DRRTC protocol. Note, however, that ten times more channels can be established with the new protocol.

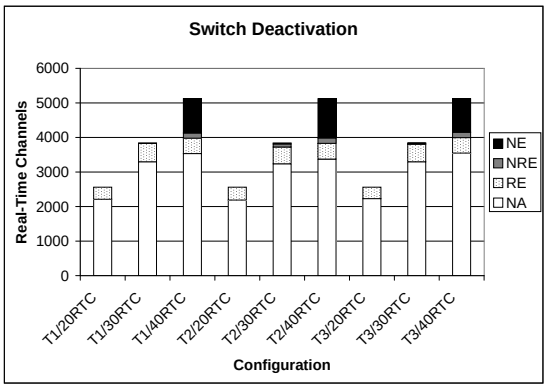
For each topology, we successively simulate the deactivation of eight switches, one by one, that is, each switch is deactivated while the rest remain activated. Switches are randomly chosen among all switches in the network. In the figures, mean values are represented. *NA* corresponds to the average number of non-affected real-time channels, and the rest corresponds to the average number of interrupted real-time channels. *RE* is the average number of re-established real-time channels after reconfiguration, and *NRE* is the average number of non re-established real-time channels because a new secondary channel could not be allocated after reconfiguration. The average number of real-time channels affected by switch deactivation is very similar for all configurations However, the average numbers of re-established channels differ considerably from each other according to the number of real-time channels that hosts are trying to establish. As shown in Fig. 2(a) and Fig. 2(b), for all topologies, as the number of real-time channels per host increases, the average number of re-established channels decreases. All interrupted channels are re-established for 2RTC and 20RTC. The average numbers of re-established channels are very similar for 3RTC and 30RTC per host, and are superior to 85% in all cases. For 4RTC per host, results are worse than for 40RTC per host In the former case the average value varies from 61% (T1) to 66% (T2) and, in the latter case, it varies from 74% (T2) to 75% (T1). Finally, note that switch activation does not affect any already established real-time channel. Consequently, no further analysis is needed.

We have analyzed the behavior of both the DRRTC protocol and the new protocol. Now, we explain why results get worse when increasing the number of real-time channels per host in both cases. In Fig. 3(a) and Fig. 3(b), we show the number of the reserved virtual channels for all switch-to-switch links before reconfiguration for 2RTC and 4RTC per host, respectively. For each switch, four bars represent the reserved virtual channels of its four ports connected to other switches. As we can observe, for 2RTC per host, only a few ports have no free virtual channels (only 47% of virtual channels are reserved). However, for 4RTC per host, most ports have consumed all its virtual channels (87% of virtual channels are reserved). Hence, the average number of established channels decreases as the channels per host increase because free virtual channels are used up in most ports. In Fig. 3(c) and Fig. 3(d), we show the reserved bandwidth for all switch-to-switch links before reconfiguration for 20RTC and 40RTC per host, respectively. For each switch, four bars represent the total amount of reserved bandwidth. As we can see, for 20RTC per host, most ports have not used all the available bandwidth yet. However, for 40RTC per host, most ports have consumed all its available bandwidth. Therefore, the average number of established channels decreases as the channels per host increase because reservable bandwidth is exhausted in most ports. Note that, with the new protocol, the limit is imposed by the maximum amount of reservable bandwidth, (that is arbitrarily set to 10% of the total link bandwidth) but not by any physical constraint. [3]

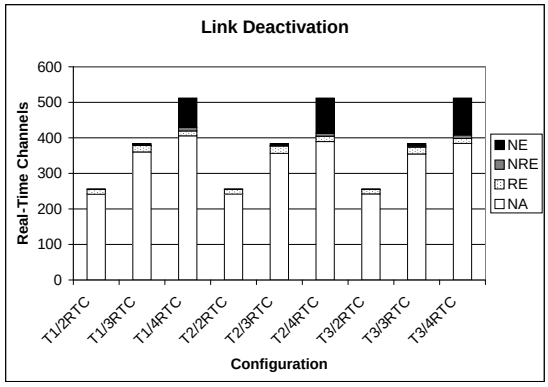
[3] In both cases, the reasoning after reconfiguration is the same as the previous ones. These figures have been omitted for the sake of brevity.



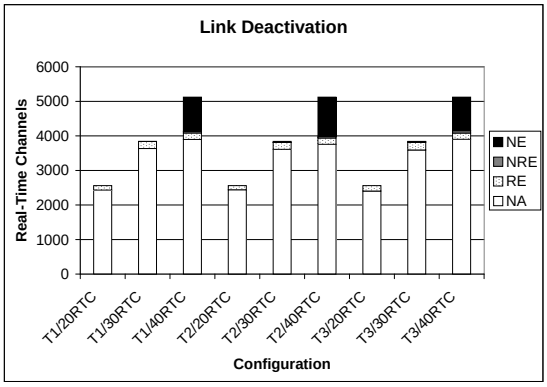
(a) Switch deactivation (dedicated virtual channels)



(b) Switch deactivation (shared virtual channel)



(c) Link deactivation (dedicated virtual channels)



(d) Link deactivation (shared virtual channel)

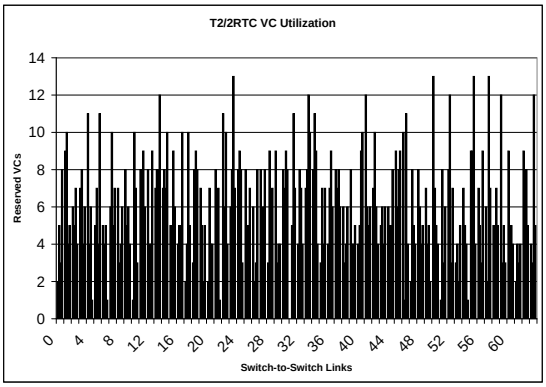
Figure 2. Evolution of real-time channels for different configurations when a single switch/link is turned off for the DRRTC protocol and for the new protocol. Configurations correspond to topologies T1, T2 and T3 when hosts try to establish 2RTC/20RTC, 3RTC/30RTC and 4RTC/40RTC per host, respectively

6.2 Link deactivation and activation

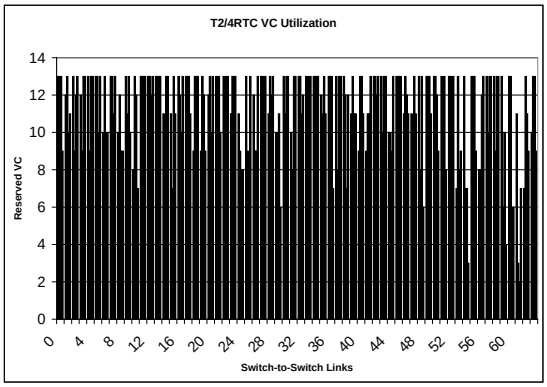
In Fig. 2(c) and Fig. 2(d), we show the evolution of the real-time channels for different configurations when a single link is turned off. For each topology, we simulate the deactivation of the four switch-to-switch links, one by one, of the same switches used to generate the results in Fig. 2(a) and Fig. 2(b). The average number of channels affected by link deactivation is approximately the same for all configurations in both cases (it varies from 4% for T1/3RTC to 6% for T3/20RTC). As expected, it is lower than the one for switch deactivation. Apart from that, results keep the same proportions as in the case for switch deactivation. Finally, note that link activation does not affect any already established real-time channel. So, no further analysis is needed.

7 Conclusions and future work

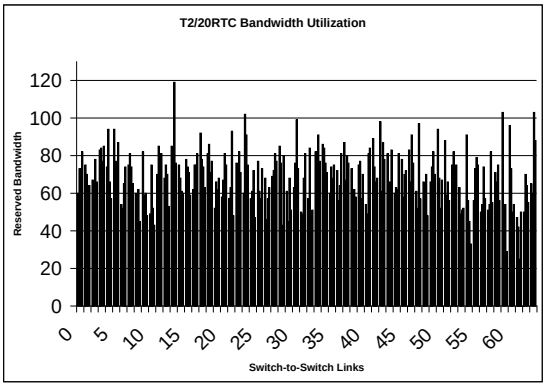
In this paper, a new protocol, that provides topology change- and fault-tolerant real-time communication services on NOWs and clusters, has been proposed and evaluated. This protocol overcomes the main drawback of a previously proposed protocol, called *Dynamically Re-established Real-Time Channels (DRRTC)*, which is physically limited by the number of virtual channels per port. To eliminate that physical constraint, the new protocol allows several real-time channels to share a given virtual channel so that it allows up to ten times the original number of supported real-time channels. This introduces two new problems that are successfully managed by the new protocol: the existence of cyclic dependencies among different real-time



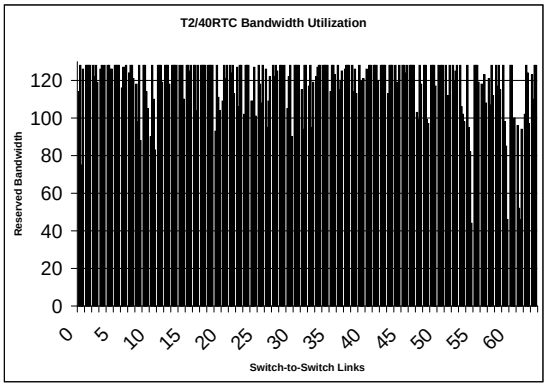
(a) Virtual channel usage before switch deactivation when establishing 2RTC (dedicated virtual channels)



(b) Virtual channel usage before switch deactivation when establishing 4RTC (dedicated virtual channels)



(c) Bandwidth usage in Mbps before switch deactivation when establishing 20RTC (shared virtual channel)



(d) Bandwidth usage in Mbps before switch deactivation when establishing 40RTC (shared virtual channel)

Figure 3. Resource usage before reconfiguration for the DRRTC protocol and for the new protocol

channels and the increased complexity of deadline requirements. The occurrence of deadlocked configuration would be detected by using a hardware counter. The affected channel would be torn down and set up again as in the case of a topology change. However, this is not a major problem because the probability of deadlocks is negligible.

We have evaluated its behavior and compared with that of the DRRTC protocol when a single switch or a single link is turned on/off for different topologies and workloads. All interrupted real-time channels are re-established after a topology change when twenty real-time connections per host are established. As the workload increases, channel recovery guarantees decrease because the reservable bandwidth is exhausted in most ports. In this way, the factor of improvement would be even larger if we allowed a larger fraction of link bandwidth to be devoted to real-time channels. Moreover, with the new protocol, topology change tolerance is only limited by the available bandwidth to establish real-time channels, as well as by the topology connectivity, because the physical constraint imposed by the number of virtual channels is eliminated.

Using the ideas presented in this paper, future work involves: a quantitative characterization of the DRRTC protocol under multiple topology changes and an analysis of the optimal assignment of hosts to switches within a bounded distance.

Acknowledgments

The authors thank *Francisco J. Alfaro* at the University of Castilla-La Mancha and *Lorenzo Fernández* at the University of Murcia for their comments during the development of this work. This work has been supported in part by the Spanish CICYT under grant TIC2000-1151-C07-03.

References

[1] F. J. Alfaro, A. Bermudez, R. Casado, F. J. Quiles, J. L. Sanchez, and J. Duato. Extending Dynamic Reconfiguration to NOWs with Adaptive Routing. In *Workshop on Communication, Architecture, and Applications for Network-based Parallel Computing*, January 2000. Held in conjunction with HPCA-6, Toulouse, France.

[2] F. J. Alfaro, J. L. Sanchez, and J. Duato. A Strategy to Manage Time Sensitive Traffic in Infiniband. In *Proceedings of Workshop on Communication Architecture for Clusters*, April 2002. Held in conjunction with IPDPS'02, Fort Lauderdale, FL.

[3] ATM Forum. *ATM Forum Traffic Management Specification. Version 4.1*, May 1995.

[4] D. R. Avresky and Y. Varoglu. Dynamically Scaling Computer Networks. In *Proceedings of Workshop on Fault-Tolerant Parallel and Distributed Systems*, April 2001. Held in conjunction with IPDPS'01, San Francisco, CA.

[5] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su. Myrinet: A Gigabit-per-Second Local-Area Network. *IEEE Micro*, 15(1):29–36, February 1995.

[6] R. Casado, A. Bermudez, F. J. Quiles, J. L. Sanchez, and J. Duato. Performance Evaluation of Dynamic Reconfiguration in High-Speed Local Area Networks. In *Proceedings of Sixth International Symposium on High Performance Computer Architecture (HPCA-6)*, January 2000.

[7] J. Duato. A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks. *IEEE Transactions on Parallel and Distributed Systems*, 4(12):1320–1331, December 1993.

[8] J. Duato. A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks. *IEEE Transactions on Parallel and Distributed Systems*, 6(10):1055–1067, October 1995.

[9] J. Duato, A. Robles, F. Silla, and R. Beivide. A Comparison of Router Architectures for Virtual Cut-Through and Wormhole Switching in a NOW Environment. In *Proceedings of International Parallel Processing Symposium*, April 1999.

[10] J. Duato, S. Yalamanchili, M. B. Caminero, D. Love, and F. J. Quiles. MMR: A High-Performance Multimedia Router-Architecture and Design Trade-Offs. In *Proceedings of Fifth International Symposium on High Performance Computer Architecture (HPCA-5)*, January 1999.

[11] J. Fernández, J. M. García, and J. Duato. A New Approach to Provide Real-Time Services in High-Speed Local Area Networks. In *Proceedings of Workshop on Fault-Tolerant Parallel and Distributed Systems*, April 2001. Held in conjunction with IPDPS'01, San Francisco, CA.

[12] J. Fernández, J. M. García, and J. Duato. Performance Evaluation of Real-Time Communication Services on High-Speed LANs under Topology changes. In *Proceedings of International Conference on High Performance Computing*, December 2001.

[13] L. Fernández, J. M. García, and R. Casado. On Deadlock Frequency during Dynamic Reconfiguration in NOWs. In *Proceedings of Euro-Par Conference*, volume 2150 of *Lecture Notes in Computer Science*, pages 630–638. Springer, 2001.

[14] S. Han and K. G. Shin. A Primary-Backup Channel Approach to Dependable Real-Time Communication in Multihop Networks. *IEEE Transactions on Computers*, 47(1):46–61, January 1998.

[15] InfiniBand Trade Association. *InfiniBand Architecture Specification Volume 1. Release 1.0*, October 2000.

[16] D. D. Kandlur, K. G. Shin, and D. Ferrari. Real-Time Communications in Multi-hop Networks. *IEEE Transactions on Parallel and Distributed Systems*, 5(10):1044–1056, October 1994.

[17] O. Lysne and J. Duato. Fast Dynamic Reconfiguration in Irregular Networks. In *Proceedings of International Conference on Parallel Processing*, August 2000.

[18] Myricom, Inc. *Myrinet GM documentation*, May 1999. http://www.myri.com/GM.

[19] R. Pang, T. Pinkston, and J. Duato. The Double Scheme: Deadlock-free Dynamic Reconfiguration of CutThrough Networks. In *Proceedings of International Conference on Parallel Processing*, August 2000.

[20] G. F. Pfister. *In Search of Clusters*. Prentice Hall PTR, second edition, 1998.

[21] J. Rexford, J. Hall, and K. G. Shin. A Router Architecture for Real-Time Communication in Multicomputer Networks. *IEEE Transactions on Computers*, 47(10):1088–1101, October 1998.

[22] T. Rodeheffer and M. D. Schroeder. Automatic Reconfiguration in Autonet. In *Proceedings of ACM Symposium on Operating System Principles*, October 1991.

[23] F. Silla and J. Duato. Improving the Efficiency of Adaptive Routing in Networks with Irregular Topology. In *Proceedings of International Conference on High Performance Computing*, December 1997.

[24] S. Warnakulasuriya and T. M. Pinkston. Characterization of Deadlocks in Irregular Networks. In *Proceedings of International Conference on Parallel Processing*, September 1999.

[25] K. H. Yum, A. Vaidya, C. R. Das, and A. Sivasubramaniam. Investigating QoS Support for Traffic Mixes with the Media Worm Router. In *Proceedings of the Sixth International Conference on High Performance Computer Architecture (HPCA-6)*, January 2000.