A New Approach to Provide Real-Time Services on High-Speed Local Area Networks

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

In the past few years, networks of workstations (NOWs) and clusters, based on high-speed local area networks (LANs), have emerged as a serious alternative to supercomputers and high-performance servers. Meanwhile, applications demanding real-time network services have also suffered a substantial growth. In order to use NOWs for distributed real-time processing, a topology change and fault-tolerant mechanism that guarantees the maximum latency or the minimum bandwidth in the worst case must be provided. Up to now, the backup channel protocol (BCP), based on real-time channels, provides fault-tolerant real-time services. But in this approach, fault tolerance is limited by the alternative paths provided by the routing function to establish the backup channels and topology change tolerance is not supported. On the other hand, dynamic reconfiguration updates the routing tables without stopping user traffic when a topology change or fault occurs. However, dynamic reconfiguration by itself does not provide neither quality of service nor real-time services, but it provides support for an additional mechanism designed to meet real-time requirements.

In this paper, we propose a new ***hardware-supported protocol to provide topology change and fault-tolerant real-time services on NOWs****. The novelty of our proposal primarily relies on the ability to assimilate hot topology changes and faults while still providing real-time services through backup channels and dynamic reconfiguration. Our protocol increases fault tolerance beyond the level provided by the backup channel protocol so that fault tolerance is only limited by topology connectivity. Furthermore, to our knowledge, our protocol is the only mechanism which is able to assimilate hot updates without stopping neither real-time traffic nor normal network operation.*

1. Introduction

The growing interest on network of workstations (NOWs) and cluster computing has been fuelled in part by the availability of powerful microprocessors and high-speed local area networks (LANs) as off-the-shelf commodity components. NOWs and clusters, based on high-speed LANs, constitute a serious alternative to supercomputers or MPPs (Massive Parallel Processors) and are the most cost-effective platform for high-performance servers. Meanwhile, applications demanding **real-time** network services, such as process control or factory automation, have also suffered a substantial growth. As a result, there is a lot of interest in using NOWs and clusters as platforms for distributed real-time applications. Distributed real-time applications impose strict conditions on traffic, with limitations on maximum message latency, maximum message delay variation and message loss rate parameters. Additionally, topology change and fault tolerance must be provided to ensure real-time services even in the worst case.

Current high-speed LAN designs [2, 5] are focused on achieving the minimum average latency and the maximum average throughput. Therefore, in order to use these high-speed LANs for hard real-time applications, a topology change and fault-tolerant mechanism that guarantees the maximum latency or the minimum bandwidth in the worst case must be provided. Moreover, high-speed LANs could also be used for other non real-time applications with less strict time constraints. Thus, network bandwidth must be shared among real-time traffic as well as best-effort one.

The concept of real-time channel [8] is one of the most well-known solutions to the problem of meeting message delivery deadlines in real-time systems. However, while this approach is suitable to provide hard real-time guarantees, it is not able to achieve neither topology change nor fault tolerance. The backup channel protocol (BCP), developed by Shin *et al.* [6], based on real-time channels,

performs recovery from faults by means of additional resources (backup channels). Nevertheless, in case of topology changes, which are the most frequent cases in current NOWs, this scheme is not able to manage this new situation efficiently.

On the other hand, dynamic reconfiguration, recently proposed by Duato *et al.* [1, 3], updates the routing tables without stopping user traffic. Note that dynamic reconfiguration by itself does not provide neither quality of service nor real-time services, but it provides support for an additional mechanism designed to meet real-time requirements.

Therefore, our proposal is based on the combination of dynamic reconfiguration with the backup channel protocol. The main objective of our proposal is to develop a new **hardware-supported protocol to provide topology change and fault-tolerant real-time services on NOWs**. This mechanism is being developed for NOWs with irregular topology which are the most cost-effective and scalable alternative to supercomputers. In this environment, topology changes are more likely due to switches/hosts being turned on/off, link remapping, hot expansion, hot replacement, etc. The novelty of our proposal primarily relies on the ability to assimilate hot topology changes and faults while still providing real-time services through backup channels and dynamic reconfiguration.

The rest of this paper is organized as follows. Next section presents our approach. In Section 3, the modified switch architecture providing support for our scheme is presented. A qualitative analysis is performed in Section 4. Section 5 describes related work. Finally, we present our concluding remarks and feasible ways of future work.

2. Our protocol

In order to support real-time communications, we set up a primary channel and a single secondary one for each real-time channel (the terms secondary channel and backup channel are used interchangeably). When either a fault or a hot topology change breaks down a primary channel, real-time messages are redirected through its secondary channel and the dynamic reconfiguration algorithm, described in [1, 3], is triggered. The procedure to be followed for interrupted secondary channels is the same apart from real-time traffic is unaffected.

After reconfiguration, a new secondary channel is allocated for each affected real-time channel because either the old secondary channel has become the new primary channel or the old secondary channel was broken down. The 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. This strategy provides more cost-effective fault tolerance than previously proposed strategies. Besides, the dynamic reconfiguration algorithm will not affect the deadline of real-time messages flowing through real-time channels by allowing message traffic during reconfiguration.

Figures 1, 2 and 3 show real-time channel establishment for a source host, a destination host and an intermediate switch respectively. State diagrams are used in order to completely specify the behavior of the protocol. State diagrams for hosts correspond to single real-time channels. State diagram for switches corresponds to a single primary or secondary channel. The notation conventions for state diagrams are the following: **bold font** is used for sent messages and actions, normal font is used for received messages and *italic font* is used for satisfied conditions.

2.1. 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 secondary one. Each of them is a set $\{H_1, H_2, B, D, T\}$ where 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. Real-time messages flow through the primary channel from H_1 to H_2. In the meantime, the secondary channel remains idle and does not consume link bandwidth but just a virtual channel in each switch along its path from H_1 to H_2. When the primary channel fails, the secondary channel becomes the new primary channel and real-time messages are redirected through it. Enough resources are assigned to each channel to meet its bandwidth and deadline requirements before real-time messages are transmitted. In order to maximize fault tolerance, the primary channel and the secondary channel should not share physical resources, that is, disjoint physical paths are essential.

Before transmitting real-time messages, H_1 must reserve the necessary resources for both the primary and the secondary channel. 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 availability of resources must be checked by means of an **admission control strategy**. For each possible output port provided by the routing function for a RTC_REQUEST message, the admission control strategy determines the availability of a free virtual channel and enough bandwidth (B) to set up the channel. Output ports without enough resources are no longer considered. The selection of an output port, among all ports with available resources, involves a partial decision about the channel path. This decision must choose an appropriate port in order to solve the **disjoint path selection problem** for primary and secondary channels (we will deal with this problem later).

If there are enough resources and an appropriate output port can be found, resources are allocated, channel is added to the channel table (see Table 1), and the request is forwarded to the next switch. If not, a best-effort message, called RTC_RESPONSE(False), is returned to H_1 following a different path from that of RTC_REQUEST. The corresponding RTC_MSG(Release)* will be ignored by the switch in the initial state.

Both reserved virtual channel and bandwidth are registered by means of especial registers located in the switch output ports as shown in Figure 4. Channel table content is shown in Table 1. Global channel identifier consists of CHANNEL, HSSOURCE, TYPE and ATTEMPT. ATTEMPT allows to distinguish among different attempts for the same real-time channel.

When a RTC_REQUEST message arrives at H_2, the primary channel is accepted if and only if channel length is not too long to prevent real-time messages from meeting their deadlines. *MaxLength* is the maximum admissible channel length in order to real-time messages meet their deadlines. Then, H_2 sends a best-effort message, called RTC_RESPONSE(True), back to H_1.

If H_1 receives a RTC_RESPONSE(False) message, resources are released by means of a RTC_MSG(Release) message and RTC_REPORT(*) messages received for that attempt are ignored.

If channel timeout expires, resources are released by means of a RTC_MSG(Release) and RTC_RESPONSE(*) or RTC_REPORT(*) messages received for that attempt are ignored. Channel timeout allows H_1 to release resources when neither RTC_RESPONSE(True) nor RTC_RESPONSE(False) messages are received. After a few cycles, H_1 will try to establish the primary channel again until a maximum number of attempts is reached. *MaxAttempts* is the maximum number of attempts to establish a primary or a secondary channel.

If H_1 receives a RTC_RESPONSE(True) message, the secondary channel will be established likewise the primary one. Once both the primary and the secondary channel have been successfully established, the real-time channel establishment process has finished. Otherwise, resources for both the primary and the secondary channel are released, and the real-time channel is rejected. Note that several channels could be concurrently established.

As mentioned earlier, an essential condition to guarantee our protocol behaves correctly is the use of disjoint physical paths for the primary and the secondary channel. Next, we are going to describe the selection of channel routes to satisfy that constraint. As the primary and the secondary channels should not share physical resources, the routing for RTC_REQUEST messages should be flexible enough to provide disjoint physical paths among source and destination hosts. To ensure their paths do not share neither switches nor links[1], besides checking the availability of resources, the control admission strategy looks up the channel identifier[2], included in the RTC_REQUEST message, in the channel table. If an entry is found, a RTC_RESPONSE(False) message is sent back to H_1 (see Figure 3), and H_1 will try to establish the channel again after a few cycles. If not, some criterion must be applied to select an output port. In order to deal with this problem, a different selection function from that of best-effort messages is used for request messages. Several selection functions are being considered:

- *Round-Robin*. All the valid output channels are alternatively selected for routing request messages with the same destination host.
- *Minimal*. Request messages are routed using minimal routing.
- *Bandwidth Balancing among Primary Channels*. The output channel with minimum amount of reserved bandwidth for primary channels is chosen. Ties are broken by selecting the output channel randomly.
- *Bandwidth Balancing among Primary and Secondary Channels*. The output channel with minimum amount of reserved bandwidth for primary channels is chosen. In the case of a tie, the output channel with minimum amount of reserved bandwidth for secondary channels is chosen. The remaining ties are broken by selecting the output channel randomly.

Combinations of these strategies, such as Minimal and Round-Robin, are also being analyzed. In any case, we are interested in the simpler strategy which is able to maximize the channel establishment rate while minimizing the number of attempts.

Once an output port is selected, resources are allocated and the request is forwarded through it 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.

2.2. Real-time channel operation

Once the real-time channel has been successfully established, H_1 begins to inject 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 RTC_MSG(Release) messages. In the latter case,

[1] Initially, up to two NICs per host are assumed so that the primary and the secondary channels have not to share neither links nor switches.

[2] The only difference between a primary and a secondary channel identifier is the type field (see Table 1).

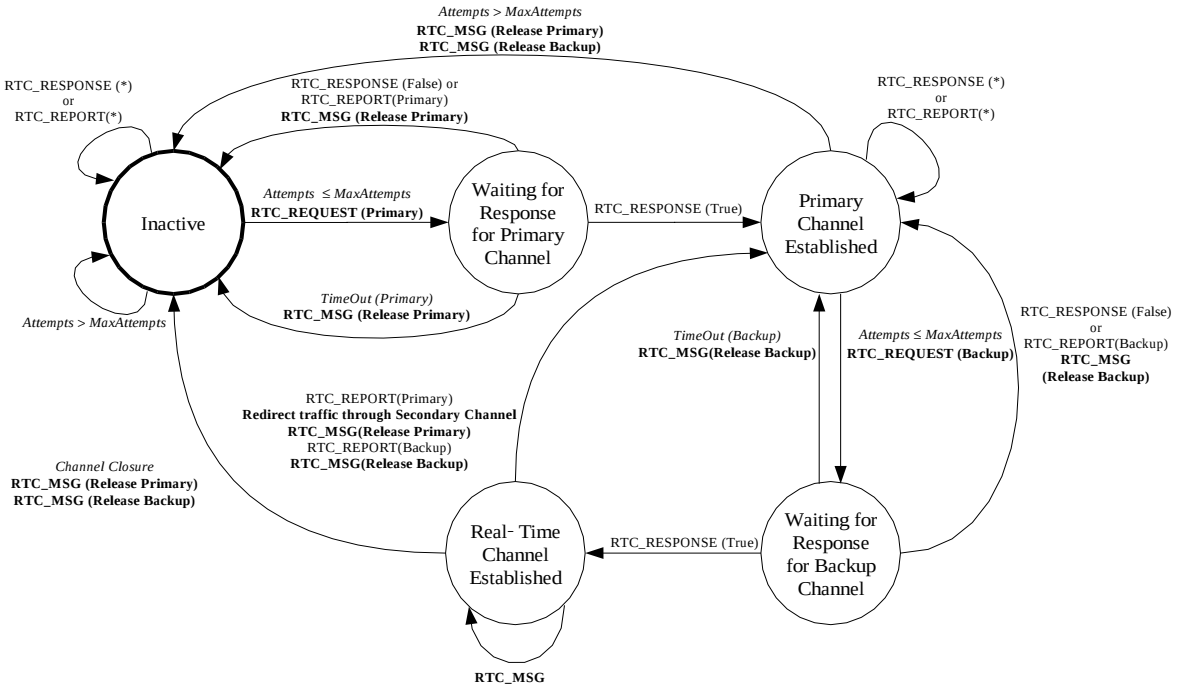

Figure 1. State diagram for Source Host (H_1).

Field	Meaning
CHANNEL	Channel identifier in the source host
HSSOURCE	Source host identifier
TYPE	Primary or Secondary
ATTEMPT	Number of attempts
Bandwidth	Reserved bandwidth
Deadline	Maximum delay or deadline
LinkIn	Input port
VCIn	Virtual channel of the input port
LinkOut	Output port
VCOut	Virtual channel of the output port

Table 1. Real-time channel table.

RTC_MSG messages are redirected through the secondary channel. Note that if the secondary channel is broken down, real-time traffic is unaffected.

2.3. 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 satisfying real-time requirements.

Let us assume that a single link fails. Next, the two adjacent switches detect the fault and determine the broken channels looking at their real-time channel tables.

For each channel whose LinkOut field matches with the broken link, a best-effort message, called RTC_REPORT, is sent to its corresponding HSSOURCE. The switch remains in the releasing state until the corresponding RTC_MSG(Release) is received. If report timeout expires, a RTC_REPORT message is sent again.

For each channel whose LinkIn field matches with the broken link, a RTC_MSG(Release) message is sent to the destination host through the channel. Every time a report arrives at HSSOURCE for the primary or the secondary channel, a RTC_MSG(Release) releases resources from HSSOURCE to the switch that detected the fault, and RTC_RESPONSE(*) messages received for that attempt are ignored. 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 is triggered. This process performs sequences of partial routing table updates to avoid stopping user 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 because ei-

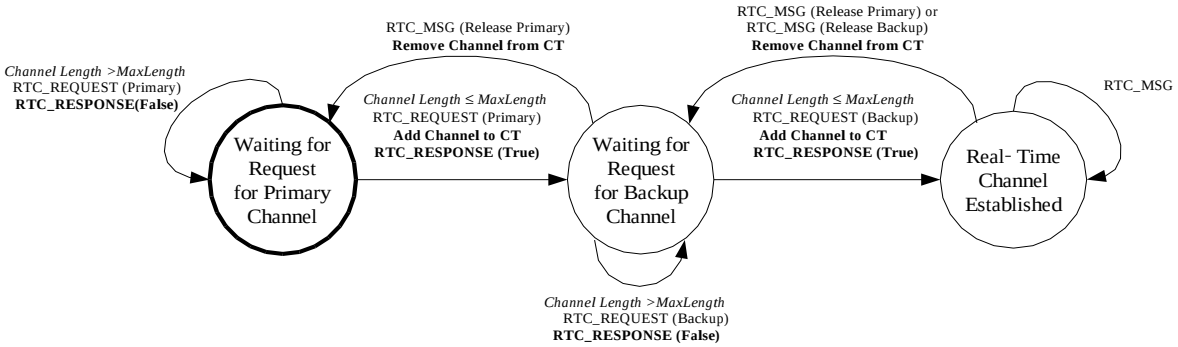

Figure 2. State diagram for Destination Host (H_2).

ther the old secondary channel has become the new primary channel or the old secondary channel was broken down.

Therefore, our protocol is topology change and fault-tolerant while satisfying real-time requirements because of the restrictions imposed in the channel establishment phase.

2.4. Fault tolerance property of our protocol

Finally, this section shows the fault tolerance property of our protocol, that is, how our protocol is able to recover itself from the loss of any type of message. Next, we analyze all possibilities.

If a RTC_REQUEST message gets lost, the timeout associated with the channel will expire (see Figure 1). Then, reserved resources will be released by means of a RTC_MSG(Release) message. Finally, if the maximum number of attempts has not been reached, a new RTC_REQUEST message will be sent. For RTC_RESPONSE messages, the situation is the same as above (see Figure 1).

The loss of a RTC_MSG(Release) message could cause resources to be reserved and not released, therefore degrading performance. Nevertheless, when a fault or topology change causes the loss of a RTC_MSG(Release) message, the next switch of the channel route detects the fault or change (see Figure 3). Consequently, it sends a new RTC_MSG(Release) for that channel.

If a RTC_REPORT message gets lost, the sender switch will not receive the corresponding RTC_MSG(Release). Next, the timeout associated with the report will expire (see Figure 3). Then, the RTC_REPORT message will be sent again.

As shown above, the fault tolerance property of our protocol is only limited by topology connectivity. In any case, we ensure the protocol releases all unused resources whenever a channel is released because of a fault or topology change.

3. Modified switch architecture

Additional hardware is essential to support topology change and fault-tolerant real-time channels. Although a detailed hardware design is out of the scope of this paper, switch architecture is depicted to help readers to understand our proposal. As shown in Figure 4, our approach uses an input-buffered switch with virtual channels. The highlighted hardware elements provide support for our protocol.

Each switch has a fixed number of ports. Every port has the same number of virtual channels: RTC virtual channels for real-time traffic[3], MIN adaptive virtual channels (minimal routing) and UD virtual channels (Up*/Down* routing) for best-effort traffic (see Subsection 3.2), and C/RTC virtual channels for reconfiguration control traffic and channel establishment traffic. The CONTROL UNIT manages all control messages due to reconfiguration, and all messages used to establish and to tear down real-time channels [4]. The CONTROL PORT allows the control unit to inject messages. The CHANNEL TABLE (see Table 1) has an allocated entry for each channel, primary or secondary, that goes through the switch. Each output port has three registers storing its reserved resources. PRIMARY and SECONDARY are the amount of reserved bandwidth for primary and secondary channels respectively. RVC keeps track of the reserved virtual channels. The VIRTUAL CHANNEL ARBITER included in each output port implements a scheduling algorithm to provide an appropriate forwarding behavior to support real-time traffic. A multiplexed crossbar connects input ports to output ports. Since a virtual channel is used for each connection[5], a demultiplexed crossbar becomes prohibitive.

[3]These virtual channels could be also used by best-effort traffic if they have not been reserved.

[4]Note that the control unit is essential to implement dynamic reconfiguration.

[5]Initially, up to 16 virtual channels per port are being considered.

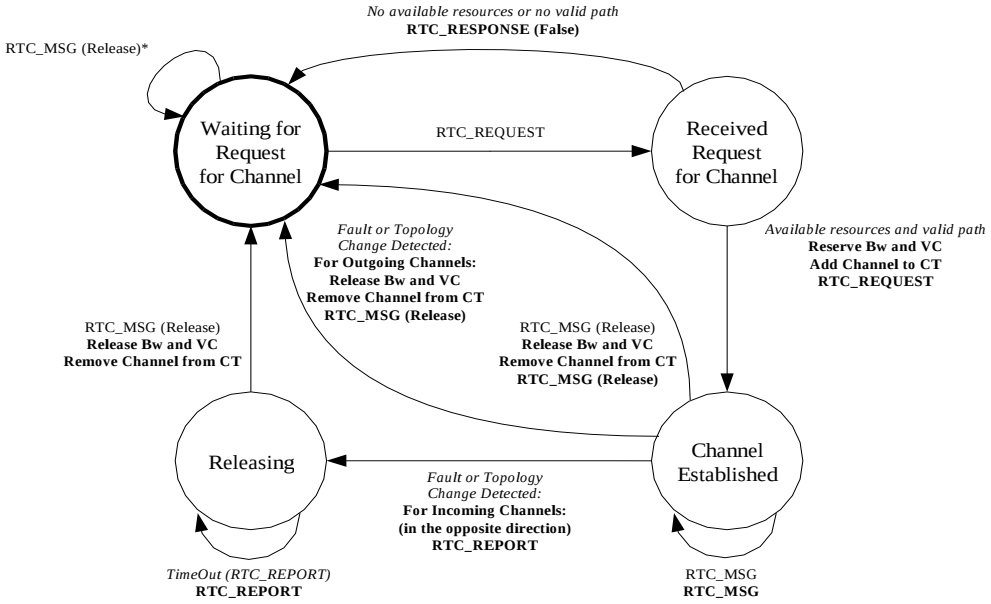

Figure 3. State diagram for a Switch.

3.1. Switching technique

Most commercial switches designed for high-speed networks implement wormhole switching [2, 5]. However, long wires required in NOWs lead to large buffers to support pipelined channels and to avoid buffer overflow while sending flow control signals. On the other hand, in virtual cut-through switching, buffer size is independent of wire length and flow control is simpler. Moreover, the traditional disadvantages of virtual cut-through, such as buffer requirements and packetization overhead, disappear in NOWs. The wire lengths make the buffer requirements of both wormhole and virtual cut-through to be very similar. Most message-passing libraries split messages into fixed-size packets for efficiency purposes. Consequently, a virtual cut-through switch can be simpler than a wormhole one for best-effort traffic, while still achieving the advantages of using pipelined routing, virtual channels and adaptive routing [4]. For real-time traffic, a connection-oriented and reservation-based switching technique is essential to support real-time channels. In order to share resources between the two traffic types, a pipelined switching technique, similar to pipelined circuit switching, is used.

3.2. Routing unit

The routing unit executes the routing algorithm. The routing algorithm can be deterministic or adaptive. Deterministic routing simplifies the design of the dynamic reconfiguration algorithm [1]. However, when a high number of channels is established, best-effort traffic can suffer starvation because of its lower priority (see Subsection 3.3). Providing a higher number of alternatives to best-effort messages could alleviate this problem.

To build an adaptive deadlock-free routing algorithm, we use the methodology proposed in [15] for wormhole networks that is also valid for virtual cut-through switching, modified as indicated in [4]. Given a deadlock-free routing function (Up*/Down*), we add virtual channels in parallel with the existing ones (Min and C/RTC). The new virtual channels can be used without any restriction as long as the original channels are used exactly in the same way as in the original routing function. Real-time messages flow through pre-established real-time channels.

3.3. Switch scheduling

Switch scheduling refers to the problem of matching input ports with requested output ports in a conflict-free way. Since our switch is asynchronous, when an output port becomes free, an input port must be chosen among all possible candidates. Each output port has a separate arbiter, called *virtual channel arbiter* (see Figure 4), that selects the next packet to forward from the set of candidate packets available for transmission through that port.

The scheduling algorithm used by arbiters is based on that of InfiniBand [7, 13]. Virtual channel arbiters implement a two-level priority scheme, using preemptive scheduling layered on top of a weighted fair scheme, where

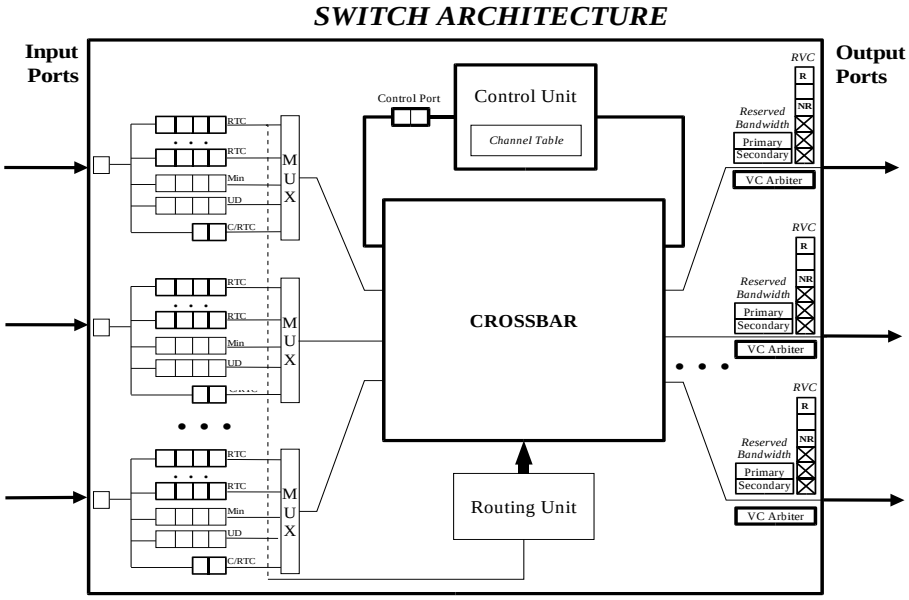

Figure 4. Modified Switch Architecture.

all packets at a precedence level are sent before any packet at a lower precedence level. Additionally, the algorithm provides a method to ensure forward progress on the low-priority virtual channels. Each virtual channel arbiter has an *arbitration table* which consists of three components, High-Priority, Low-Priority and Limit of High-Priority. The High-Priority and Low-Priority components are each a list of {VC,W} pairs. Each list entry contains a virtual channel number (VC) and a weighting value (W) specifying the amount of bandwidth allocated to that VC when its turn in the arbitration occurs. The Limit of High-Priority component indicates the amount of high-priority packets that can be transmitted without an opportunity to send a low priority packet. Within each high or low-priority table, weighted fair arbitration is used. The order of entries in each table specifies the order of virtual channel scheduling.

In the case of our protocol, virtual channels are assigned to the two lists as follows. RTC virtual channels reserved for real-time channels are included in the High-Priority list. Their weighting values are proportional to their associated bandwidth. The rest of RTC virtual channels are included in the Low-Priority list. Min and UD virtual channels are always included in the Low-Priority list. Finally, C/RTC virtual channels are inserted in the High-Priority list. Note that real-time traffic and reconfiguration and channel establishment traffic are assigned the same priority level to simplify design. However, C/RTC virtual channels may be assigned the lowest weights and RTC virtual channels may appear several times in the list.

4. Qualitative analysis

This section summarizes the benefits and design trade-offs of our approach. The most important concepts are the following:

- Topology change tolerance. To our knowledge, our protocol is the only one which is able to assimilate hot topology changes without stopping neither real-time traffic nor normal network operation.
- Fault tolerance is only limited by topology connectivity and not by the constraints imposed by the routing function to avoid deadlocks. Hence, recovery guarantees are expected to be better than with any other scheme.
- Channel recovery delay. The channel recovery delay is the time to complete reconfiguration plus the channel establishment delay for the new secondary channel. Dynamic reconfiguration process takes a few milliseconds to finish while the mean time between failures is much longer [9, 11]. Channel establishment delay for the new secondary channel is shorter than the necessary time to complete reconfiguration.
- Service disruption affects those channels whose primary channel is interrupted when a fault or topology change occurs. It lasts from the fault or topology change detection to the moment in which the source host receives the RTC_REPORT message.
- Hardware cost. See Subsection 3.

5. Related Work

Static reconfiguration techniques (Autonet [14] and Myrinet with GM [11]) stop user traffic to update routing tables. Although reconfigurations are not frequent, they can degrade performance considerably [14]. Moreover, real-time constraints can not be met because of traffic disruption. Duato *et al.* [1, 3] have proposed a dynamic reconfiguration algorithm, called *Partial Progressive Reconfiguration*, to minimize the negative effects of static reconfiguration on network performance. The protocol guarantees that the global routing algorithm remains deadlock-free at any time. Pinkston *et al.* [12] have developed a simple but effective strategy for dynamic reconfiguration in networks with virtual channels. Lysne *et al.* [10] aim at reducing the scope of reconfiguration identifying a restricted part of the network, the *skyline*, as the only part where a full reconfiguration is necessary.

Shin *et al.* have proposed the *Backup Channel Protocol* (BCP) [6] to achieve fault-tolerant real-time communications. 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.

6. Concluding remarks and future work

In this paper, a new hardware-supported approach to provide real-time services on NOWs in an scenario with topology changes and faults has been presented. Our proposal uses real-time channels and the backup channel protocol to achieve topology change and fault-tolerant real-time communications. The novelty of our proposal primarily relies on the ability to assimilate hot topology changes and faults while still providing real-time services by means of dynamic reconfiguration. In this way, the number of allowed topology changes or faults is not limited by the number of possible backup channels that the routing function can provide but just by the topology connectivity.

Using the ideas presented in this paper, future work involves a quantitative characterization of our protocol in terms of resource overhead, recovery delay and recovery guarantees; an analysis of the impact of single and multiple link/switch faults/changes on real-time and best-effort traffic; and the adaptation of our protocol to InfiniBand [7].

7. Acknowledgments

The authors thank *Francisco J. Alfaro* at the University of Castilla La Mancha for his comments and suggestions during the course 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 *Proceedings of Communications, Architecture, and Applications for Network-based Parallel Computing Workshop (CANPC'00)*, January 2000.

[2] 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.

[3] 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.

[4] 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 13th International Parallel Processing Symposium*, pages 240–247. IEEE Computer Society Press, April 1998.

[5] D. Garcia and W. Watson. ServerNet II. In *Proceedings of Parallel Computing Routing and Communication Workshop*, June 1997.

[6] 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), 1998.

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

[8] 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.

[9] C. Labovitz, A. Ahuja, and F. Jahanian. Experimental Study of Internet Stability and Wide-Area Networks Failures. In *Proceedings of Fault-Tolerant Computing Symposium*, June 1999.

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

[11] Myricom, Inc. http://www.myri.com.

[12] 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.

[13] J. Pelissier. Providing Quality of Service Over Infiniband Architecture Fabrics. In *Proceedings of Hot Interconnects*, August 2000.

[14] T. Rodeheffer and M. D. Schroeder. Automatic Reconfiguration in Autonet. *Proceedings of the 13th ACM Symposium on Operating System Principles*, pages 183–187, February 1991.

[15] F. Silla and J. Duato. Improving the Efficiency of Adaptive Routing in Networks with Irregular Topology. In *Proceedings of the 1997 International Conference on High Performance Computing (HiPC'97)*, December 1997.