In-Transit Buffers: A Mechanism to Support Minimal Routing in Myrinet*

J. Flich, M. P. Malumbres, P. López, J. Duato

Dpto. Informática de Sistemas y Computadores, Universidad Politécnica de Valencia

Camino de Vera, 14, 46071–Valencia, Spain. E-mail: {jflich,mperez,plopez,jduato}@gap.upv.es

Due to the increasing computing power of microprocessors and the high cost of multiprocessors, networks of workstations (NOWs) are currently being considered as a cost-effective alternative for parallel computing. One of the preferred network technologies for building NOWs is Myrinet [1], which provides simple and very flexible network designs. In particular, it allows the network behavior to be changed through the Myrinet Control Program (MCP) executed by the LANai processor in the network interface card (NIC).

Myrinet uses up*/down* routing [4] to build paths between network hosts. Unfortunately, up*/down* routing is not always able to provide a minimal path between some pairs of hosts, as shown in the following example. In Figure 1.a, a message transmitted from Switch 4 to Switch 1 cannot go through any minimal path. The shortest path (through Switch 6) is not allowed since the message should traverse a link in the "up" direction after one in the "down" direction which is forbidden in up*/down* routing. All the allowed paths (through switches 0, 2, and through switches 0, 5) are non-minimal. The number of forbidden minimal paths increases as the network becomes larger. Consequently, minimal routing is usually disallowed due to the up*/down* restrictions. Another drawback of up*/down* routing is that it forces most of the traffic to cross the root switch (Switch 0), leading to saturation at relatively low traffic.

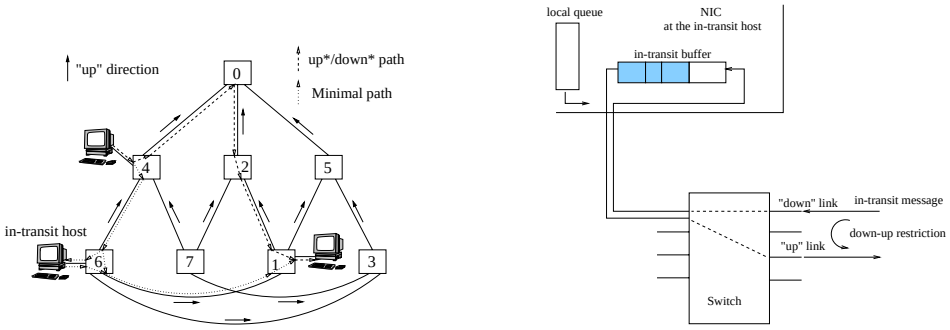

Figure 1: a) Up*/down* and Minimal Routing. b) Implementation of the in-transit buffer mechanism

The basic idea to eliminate these drawbacks consists of splitting the forbidden minimal paths into several up*/down* paths. On each path, an intermediate host is selected as the destination and, at this host, packets are ejected from the network and later re-injected into it. In other words, the dependencies between "down" and "up" links are removed by the use of some buffers at the intermediate hosts (i.e. in-transit buffers). In Figure 1.a we can see that, with the in-transit buffer mechanism, a minimal route can be used to route packets from Switch 4 to Switch 1. To break the channel dependencies, packets are sent to a host connected to the intermediate Switch 6. This host will re-inject packets as soon as possible.

The in-transit buffer mechanism adds latency to each packet crossing intermediate hosts and also uses some additional resources in both the network (i.e. links) and the network interface cards (i.e. memory pools and DMA engines). On the other hand, with this mechanism, "down" to "up" transitions are allowed. The resulting routing algorithm is less restrictive so that less traffic needs to cross the root switch than in the original up*/down* routing algorithm, and it always provides minimal paths among all nodes.

Figure 1.b shows the implementation of the in-transit buffer mechanism. A packet that needs the in-transit buffer will be addressed to the in-transit host. The in-transit host will re-inject the packet as

*This work was supported by the Spanish CICYT under Grant TIC97–0897–C04–01 and by Generalitat Valenciana under Grant GV98-15-50

soon as possible into the network, forwarding it to its destination host or to another in-transit host. To implement the mechanism, some NIC memory is needed and the MCP program has to be modified. In order to minimize the introduced overhead, as soon as the in-transit packet header is processed and the required output channel is free, a DMA transfer is programmed to re-inject the in-transit packet. There is no problem if the DMA transfer begins before the packet has been completely received, because it will continue arriving at the same rate that it is being transmitted[1], assuming that all the links in the network have the same bandwidth[2]. The only additional requirement is that the message is completely stored in the NIC memory at the source host before starting transmission to avoid interference with the host I/O bus.

To make this mechanism deadlock-free, it must be guaranteed that an in-transit packet that is being re-injected can be completely ejected from the network if the re-injected part of the packet becomes blocked, thus removing potential channel dependencies that may result in a deadlock (down-up transitions). So, when an in-transit packet arrives at a given host, care must be taken to ensure that there is enough buffer space to store it at the interface card before starting the DMA transfer. Otherwise, the MCP stores the packet in the host memory, considerably increasing overhead. Also, because we are assigning buffers in host memory to packets, we need to avoid possible deadlock situations due to dependencies among the buffers. We solve this by grouping those buffers into buffer classes [3].

We are evaluating the new mechanism by simulation. We have simulated 40 topologies using network sizes of 8, 16, 32, and 64 switches for uniform, bit-reversal, hot-spot, and local traffic patterns. Message sizes of 32, 512, and 1024 bytes are used.

	32 bytes			1024 bytes		
Switches	Min	Max	Avg	Min	Max	Avg
8	-9.0105	10.0971	-3.1212	-17.2995	13.4601	-8.0626
16	9.2622	62.8934	33.4776	0.4845	49.7260	26.6592
32	66.4511	139.5732	99.2126	49.9866	100.7512	77.3265
64	160.0910	288.9706	220.5531	124.6746	220.2373	164.8961

Table 1: Percentages of throughput increase when using the in-transit buffer mechanism for the uniform distribution.

For uniform traffic distribution (Table 1), we find that the in-transit buffer mechanism always increases network throughput for networks of 16 switches or larger. Average improvement ranges from 26% for 16-switch networks to 220% for 64-switch networks. For 8 switches, most up*/down* paths are minimal, thus leaving little potential for performance improvement by the proposed mechanism. For hot-spot, bit-reversal, and local traffic distributions, similar results are obtained. Regarding latency, this mechanism only increases latency significantly for short messages (13% on average for 32-byte messages). For longer messages, latency increase is minimal and could even decrease for large networks as more minimal paths are provided by this routing mechanism.

We are currently implementing the in-transit buffer mechanism in the Myricom GM message-passing software [2].

References

[1] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. Seizovic and W. Su, "Myrinet - A gigabit per second local area network," *IEEE Micro*, pp. 29–36, February 1995.

[2] GM protocol, 'http://www.myri.com/GM'

[3] I. S. Gopal, "Prevention of store-and-forward deadlock in computer networks,," *IEEE Transactions on Communications*, vol. COM-33, no. 12, pp. 1258-1264, December 1985.

[4] M. D. Schroeder et al., "Autonet: A high-speed, self-configuring local area network using point-to-point links," Technical Report SRC research report 59, DEC, April 1990.

[1] Due to limited memory bandwidth in the NICs, a source host may inject *bubbles* into the network, thus lowering the effective reception rate at the in-transit host. This problem has been addressed and can be easily avoided when implementing the MCP code. Also, future implementations of Myrinet interfaces will eliminate this problem.

[2] Myrinet supports mixing links with different bandwidth.