Improving the Up*/Down* Routing Scheme for Networks of Workstations *

José Carlos Sancho and Antonio Robles

Departamento de Informática de Sistemas y Computadores
Universidad Politécnica de Valencia
P.O.B. 22012,46071 - Valencia, SPAIN
{jcsancho,arobles}@gap.upv.es

Abstract. Networks of workstations (NOWs) are being considered as a cost-effective alternative to parallel computers. Many NOWs are arranged as a switch-based network with irregular topology, which makes routing and deadlock avoidance quite complicated. Current proposals use the $up^*/down^*$ routing algorithm to remove cyclic dependencies between channels and avoid deadlock. Recently, a simple and effective methodology to compute $up^*/down^*$ routing tables has been proposed by us. The resulting $up^*/down^*$ routing scheme makes use of a different link direction assignment to compute routing tables. Assignment of link direction is based on generating an underlying acyclic connected graph from the network graph. In this paper, we propose and evaluate new heuristic rules to compute the underlying graph. Moreover, we propose a traffic balancing algorithm to obtain more efficient $up^*/down^*$ routing tables when source routing is used. Evaluation results show that the routing algorithm based on the new methodology increases throughput by a factor of up to 2.8 in large networks, also reducing latency significantly.

Keywords: Networks of workstations, irregular topologies, routing algorithms, deadlock avoidance.

1 Introduction

NOWs are arranged as a switch-based network with irregular topology which provides the wiring flexibility, scalability, and incremental expansion capability required in this environment. Routing in irregular topologies can be based on either source or distributed routing. In the former case, routing tables are used at each host to obtain the port sequence to be used at intermediate switches to reach the destination. In order to achieve high bandwidth and low latencies, NOWs are often connected using gigabit local area network technologies. There are recent proposals for NOW interconnects like Autonet [8], Myrinet [1], Servernet II [4], and Gigabit Ethernet [9].

Several deadlock-free routing algorithms have been proposed for NOWs, such as $up^*/down^*$ routing [8], adaptive-trail routing [5], minimal adaptive routing [7], and smart-routing [2]. However, we will focus on $up^*/down^*$ routing because it is the most popular routing scheme currently used in commercial networks, like Myrinet [1].

* This work was supported by the Spanish CICYT under Grant TIC97-0897-C04-01.

A. Bode et al. (Eds.): Euro-Par 2000, LNCS 1900, pp. 882–889, 2000.

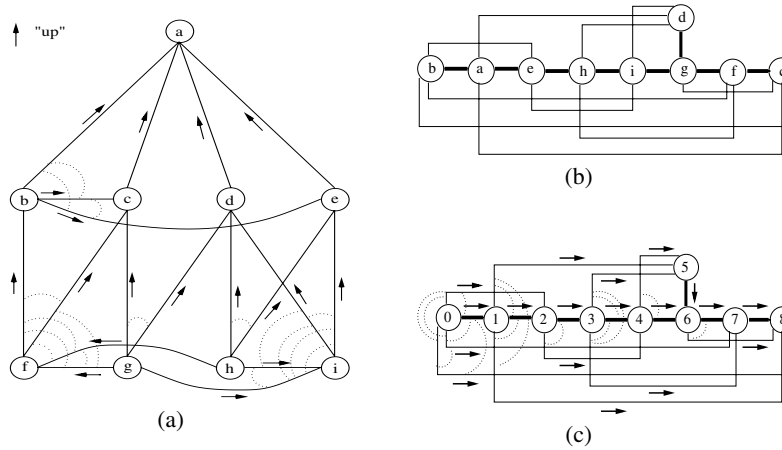

Fig. 1. (a) Generated BFS spanning tree for a 9-switch network with assignment of direction to links. (b) Generated DFS spanning tree, and (c) assignment of direction to links for the same 9-switch network.

In this paper, we propose and evaluate new heuristic rules and a new traffic balancing algorithm that improve the methods to compute $up^*/down^*$ routing tables in a NOW environment when source routing is used. Evaluation results show that the routing algorithm based on the new methodology increases throughput by a factor of up to 2.8 for large networks, also reducing latency significantly. The rest of the paper is organized as follows. In Section 2, the $up^*/down^*$ routing scheme and the methodologies to compute its routing tables are described. In Section 3, new heuristic rules to compute the $up^*/down^*$ routing scheme are proposed. Section 4 describes the proposed traffic balancing algorithm when using source routing. Section 5 shows performance evaluation results. Finally, in Section 6 some conclusions are drawn.

2 Up*/Down* Routing

$Up^*/down^*$ is the most popular routing scheme currently used in commercial networks. In order to compute $up^*/down^*$ routing tables, different methodologies can be applied. These methodologies are based on an assignment of direction ("up" or "down") to the operational links in the network by building a spanning tree. These methodologies differ in the type of graph to be built. One methodology is based on a BFS spanning tree, such as it was proposed in Autonet [8], whereas another methodology is based on a DFS spanning tree, as it has been recently proposed in [6].

In networks without virtual channels, the only practical way of avoiding deadlocks consists of restricting routing in such a way that cyclic channel dependencies [1] are

[1] There is a channel dependency from a channel c_i to a channel c_j if a message can hold c_i and request c_j. In other words, the routing algorithm allows the use of c_j after reserving c_i. Also, there is a routing restriction when there is no channel denpendency.

avoided [3]. To avoid deadlocks while still allowing all links to be used, $up^*/down^*$ routing uses the following rule: a legal route must traverse zero or more links in "up" direction followed by zero or more links in "down" direction. Thus, cyclic channel dependencies are avoided by imposing routing restrictions, because a message cannot traverse a link along the "up" direction after having traversed one in "down" direction. Next, we describe how to compute both a BFS and DFS spanning tree, and how to assign direction to links in each graph.

2.1 Computing a BFS Spanning Tree

First, to compute a BFS spanning tree, a switch must be chosen as the root. Starting from the root, the rest of the switches in the network are arranged on a single BFS spanning tree. The "up" end of each link is defined as: $1)$ the end whose switch is closer to the root in the spanning tree; $2)$ the end whose switch has the lower identifier, if both ends are at switches at the same tree level. Figure 1(a) shows the resulting link direction assignment for a 9-switch network.

2.2 Computing a DFS Spanning Tree

Like in BFS spanning trees, an initial switch must be chosen as the root before starting the computation of a DFS spanning tree. The rest of the switches are added following a recursive procedure [6]. This procedure builds a path that connects all the switches in the network. Figure 1(b) shows the DFS spanning tree obtained from the same network graph used in Figure 1(a). Unlike in the BFS spanning tree, adding switches to build the path is made by using heuristic rules. We will address this issue later.

Next, before assigning direction to links, switches in the network must be labeled with positive integer numbers. A different label is assigned to each switch. The "up" end of each link is defined as the end whose switch has the higher label. Figure 1(c) shows the label assigned to each switch. Note that, the DFS spanning tree achieves a lower number of routing restrictions, as can be seen by dashed lines in Figures 1(a) and 1(c) for the BFS and DFS spanning trees, respectively.

3 Applying New Heuristic Rules

Several spanning trees can be computed. In order to achieve better performance, heuristics to find the suitable spanning tree are needed. For BFS spanning trees, heuristic rules can only be applied to choose the root switch. The number of different BFS spanning trees that can be computed on a network graph is limited by the number of switches in the network. However, when computing a DFS spanning tree, heuristic rules can be applied to both the selection of the root switch and the selection of the following switches of the spanning tree. Notice that the number of spanning trees that could be computed in this case is very large.

We first focus on the heuristic rules for selecting the root switch. So far, two approaches have been used to select the root of a spanning tree: (R0) to select the switch with identifier equal to zero, like in DEC AN1 [8]; (R1) to select the switch with lower

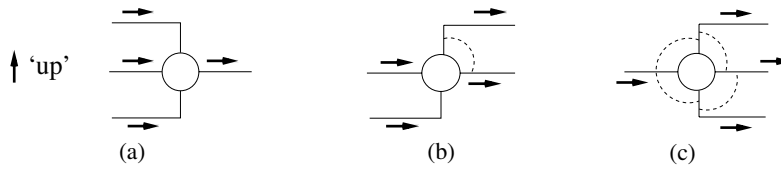

Fig. 2. Different link orientation patterns in a DFS spanning tree.

average topological distance to the rest of the switches, like Myrinet [1]. We propose a new heuristic rule that will be referred to as R2. The heuristic is based on computing all the spanning trees and selecting one of them based on two behavioral routing metrics, that is: (1) the average number of links in the shortest routing paths between hosts over all pairs of hosts, referred to as *average distance*; and (2) the maximum number of routing paths crossing through any network channel, referred to as *crossing paths*. We first compute the metrics for each spanning tree obtained by selecting the root among every switch in the network. Finally, the switch selected as the root will be the one that provides the lower value for the crossing paths metric. In case of tie, the switch with lower value of average distance will be selected. In short, the switch selected as the root will be the one that allows more messages to follow minimal paths and provides a better traffic balancing. The time complexity to compute the new heuristic rule is $O(n^3)$, where n is the number of switches.

Unlike BFS spanning tree, after selecting the root switch, a DFS spanning tree still allows heuristic rules to be applied to the rest of the switches when building the spanning tree. We propose the two following heuristic rules: (H1) The switch with higher average topological distance to the rest of the switches is selected as the next switch in the spanning tree, and so on. This heuristic was proposed in [6]. (H2) The switch with a higher number of links connecting to switches that already belong to the spanning tree is selected as the next switch. In case of tie, the H1 heuristic rule is applied. The H2 heuristic reduces the number of routing restrictions by increasing the number of switches whose links exhibit the orientation patterns shown in the Figures 2(a) and 2(b), which provide a lower number of routing restrictions in the switch than that provided by the link orientation pattern shown in Figure 2(c).

Table 1 shows the values of the behavioral routing metrics computed for several network[2]sizes using the $up^*/down^*$ routing algorithm based on both BFS and DFS spanning trees, which have been obtained according to the heuristic rules proposed above[3]. Besides the *average distance* and the *crossing path* metrics we have also included the *restrictions per switch* metric that is the average number of routing restrictions per switch. As can be seen, a lower values of metrics are obtained when using the R2 and H2 heuristic rules to compute spanning trees.

[2] For further details on topology generation, see Section 5.1.

[3] For DFS spanning trees, we assume that the R2 heuristic is used to select the root.

Table 1. Behavioral routing metrics for BFS and DFS spanning trees using different heuristics.

Spanning tree	Network size	Average distance		Crossing paths		Restrictions per node	
		R1	R2	R1	R2	R1	R2
BFS	16	2.208	2.133	37	23	3.375	3.125
	32	3.102	2.871	173	63	3.562	2.937
	64	4.013	3.787	593	238	3.281	2.875
		H1	H2	H1	H2	H1	H2
DFS	16	2.108	2.091	23	23	3.125	2.875
	32	2.792	2.752	73	43	2.821	2.625
	64	3.634	3.590	204	190	2.687	2.585

4 Traf c Balancing Algorithm

When a routing algorithm able to provide partial adaptivity, such as $up^*/down^*$ routing, is implemented using source routing, a strategy to select a single path between each pair of hosts is needed. Different selection policies can be applied, such as random and round-robin selections. However, they do not guarantee a suitable traffic balancing in the network, which may reduce network performance. We propose a traffic balancig algorithm that tries to achieve an uniform channel utilization, avoiding that a few channels become a bottleneck in the network.

First, the algorithm associates a counter to every channel in the network. Each counter is initialized to the number of routing paths crossing the channel, that is, the channel utilization. Additionally, a cost function associated to every routing path according to its channel utilization is evaluated. The procedure defined below is applied repetitively to the channel with highest value of counter. In each step, a routing path crossing the channel is selected to be removed if there is more than one routing path between the source and the destination switch of this routing path. In this way, we prevent the network to become disconnected. When a routing path is removed, the counters associated with every channel crossed by the path are updated. If there is more than one routing path able to be removed in a channel, the algorithm will first choose the routing path whose source and destination hosts have the highest number of routing paths between them. The algorithm finishes when the number of routing paths between every pair of hosts is reduced down to the unit. The time complexity to compute this traffic balancing algorithm is $O(n^2 * diameter)$, where n is the number of switches. This time is much lower than that exhibited by other proposals, such as smart-routing [2].

5 Performance Evaluation

In this section, we evaluate by simulation the performance of the $up^*/down^*$ routing scheme when the heuristic rules and the traffic balancing algorithm proposed in Sections 3 and 4, respectively, are applied to compute the routing tables. Table 2 shows

the acronyms for the $up^*/down^*$ routing algorithms evaluated according to type of spanning tree, heuristic, and traffic balancing algorithm used.

Table 2. Acronyms used for the $up^*/down^*$ routing algorithms.

Routing algorithm	Spanning tree	Root heuristic	Path heuristic	Traffic balancing
UD_BFS1	BFS	R1	-	No
UD_BFS2	BFS	R2	-	No
UD_BFS2b	BFS	R2	-	Yes
UD_DFS1	DFS	R2	H1	No
UD_DFS2	DFS	R2	H2	No
UD_DFS2b	DFS	R2	H2	Yes

5.1 Network Model

Network topology is completely irregular and has been generated randomly. We have evaluated networks with 16, 32, and 64 switches. For space reasons, the results for 64 switches have not been plotted. We have generated ten different topologies for each network size analyzed. The maximum variation in throughput improvement of UD_DFS2b routing with respect to UD_BFS1 routing is not larger than 20%. Results plotted in this paper correspond to the topologies that achieve the average behavior for each network size. We assume that every switch in the network has 8 ports, using 4 ports to connect to workstations and leaving 4 ports to connect to other switches. For message length, 32-flit and 512-flit messages were considered. Different message destination distributions have been used like uniform, bit-reversal, and matrix transpose.

In order to obtain realistic simulation results, we have used timing parameters for the switches taken from a commercial network. We have selected Myrinet because it is becoming increasingly popular due to having very good performance/cost ratio. According to Myrinet switches, the latency through the switch for the first flit is 150 ns, and after transmitting the first flit, the switch transfers at the link rate of 6.25 ns per flit. The clock cycle is 6.25 ns. Each switch has a crossbar whose arbiter processes one message header at a time. Flits are one byte wide and the physical channel is one flit wide. Also, source routing and wormhole switching is used like in Myrinet.

5.2 Simulation Results

Figures 3(a) and 3(b) show the average message latency versus accepted traffic for networks with 16 and 32 switches, respectively. Message size is 32 flits and uniform destination distribution is used. We can observe that the new heuristic (H2) to compute the DFS spanning tree allows UD_DFS2 to reduce latency with respect to UD_DFS1

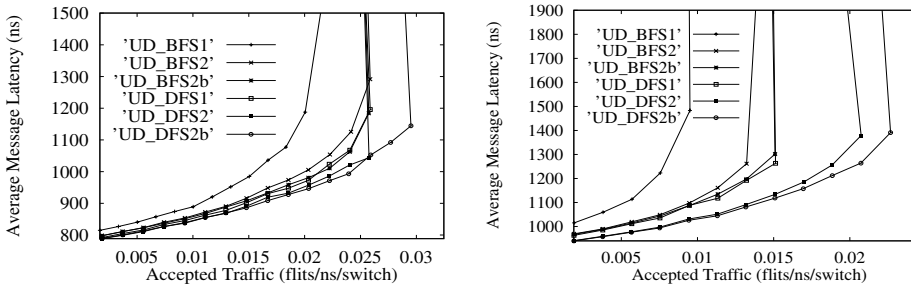

Fig. 3. Average message latency vs accepted traffic. Network size is (a) 16 and (b) 32 switches. Message length is 32 flits. Uniform distribution.

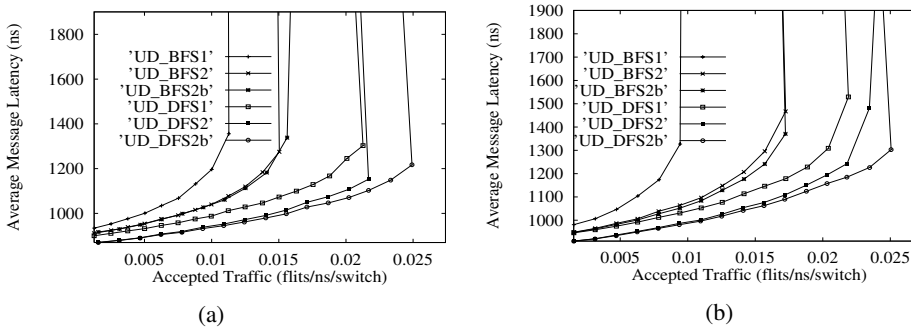

Fig. 4. Average message latency vs accepted traffic. Network size is 32 switches. Message length is 32 flits. (a) Bit-reversal and (b) matrix transpose message distributions.

for every value of traffic. It is due to the fact that the new heuristic introduces lower number of routing restrictions than the previous heuristic (H1), allowing more messages to follow minimal paths. Obviously, the improvement is higher in large networks because messages can profit more from following minimal paths. The improvement in throughput of UD_DFS2 with respect to UD_BFS1 achieves a factor of up to 2.8 for a 64-switch network. Also, the new heuristic (R2) to select the root significantly improves the performance of the $up^*/down^*$ routing scheme based on BFS spanning tree with respect to the R1 heuristic. The improvement in throughput of UD_BFS2 with respect to UD_BFS1 ranges from 20% for small networks to 60% for large networks.

Moreover, the traffic balancing algorithm only contributes to improve throughput for small network sizes, especially when $up^*/down^*$ routing is based on DFS spanning tree. It is due to the fact that channel utilization is higher than for large networks. As a consequence, an algorithm to balance traffic will achieve more benefits. The improvement in throughput of UD_DFS2b with respect to UD_DFS2 is about 16% in 16-switch networks.

For space reasons, the results for long message (512-flits) are not plotted. The improvement in performance of the $up^*/down^*$ routing schemes based on a DFS spanning tree with respect to those based on BFS spanning tree decreases slightly with respect to the one achieved with short messages. Similar results were obtained in [6].

Figures 4(a) and 4(b) show the results for a 32-switch network when using message distributions with temporal locality, such as bit-reversal and matrix transpose. The improvement in performance of UD_DFS2 with respect to UD_DFS1 is noticeably decreased, although the latency reduction is still significant for the entire range of traffic. Notice that UD_DFS2b increases throughput with respect to UD_BFS1 up to a factor of 2.5.

6 Conclusions

In this paper, we have proposed new heuristics to obtain the underlying graph used by the $up^*/down^*$ routing scheme to compute the routing tables. Moreover, an algorithm to balance the traffic in the network using source routing has been proposed in order to avoid that some channels become a bottleneck in the network. The main contribution of these techniques is that they are able to improve network performance without adding resources to the network that would increase its cost. Simply, routing tables have to be updated.

The simulation results modeling a Myrinet network show that the $up^*/down^*$ routing algorithm based on DFS spanning tree, when the new heuristic to compute the spanning tree and the traffic balancing algorithm are applied, almost triples the throughput in large networks with respect to the $up^*/down^*$ routing algorithm based on BFS spanning tree currently used in commercial networks. For smaller networks, performance improvement is also smaller but the proposed heuristic rules always improve latency and throughput.

References

1. N. J. Boden et al., Myrinet - A gigabit per second local area network, *IEEE Micro*, vol. 15, Feb. 1995.
2. L. Cherkasova, V. Kotov, and T. Rockicki, Fibre channel fabrics: Evaluation and design, *29th Hawaii International Conference on System Sciences*, Feb. 1995.
3. W. J. Dally and C. L. Seitz, Deadlock-free message routing in multiprocessors interconnection networks, *IEEE Transactions on Computers*, vol. C-36, no. 5, pp. 547-553, May. 1987.
4. D. García and W. Watson, Servernet II, in *Proceedings of the 1997 Parallel Computer, Routing, and Communication Workshop*, Jun 1997.
5. W. Qiao and L. M. Ni., Adaptive routing in irregular networks using cut-trough switches, in *Proc. of the 1996 Int. Conf. on Parallel Processing*, Aug. 1996.
6. J.C. Sancho, A. Robles, and J. Duato, New Methodology to Compute Deadlock-Free Routing Tables for Irregular Networks, in *in Proc. of CANPC'00*, Jan. 2000.
7. F. Silla and J. Duato, Improving the Efficiency of Adaptive Routing in Networks with Irregular Topology, in *1997 Int. Conference on High Performance Computing*, Dec. 1997.
8. M. D. Schroeder et al., Autonet: A high-speed, self-configuring local area network using point-to-point links, *SRC research report 59*, DEC, Apr. 1990.
9. R. Sheifert, *Gigabit Ethernet*, Addison-Wesley, April 1998.