Multicasting at the Host Interface Level in Wormhole Networks*

Claudio Casetti, Emilio Leonardi, Fabio Neri
Dipartimento di Elettronica
Politecnico di Torino, Torino, Italy
email: {casetti,leonardi,neri}@polito.it

Cosimo Anglano
Dipartimento di Informatica
Università di Torino, Torino, Italy
email: mino@di.unito.it

Abstract

In this paper we address the problem of providing transparent, reliable, and efficient network-level multicasting in wormhole LANs. We describe some alternatives for achieving deadlock-free multicasting using fast buffer reservation techniques at the host interface level. Tradeoffs involving complexity and performance of various solutions are discussed, and are illustrated using simulation results. Our results show that in most cases the straightforward approach of sending multiple unicast copies of the multicast message at the originator host turns out to be the most simple and effective approach.

1. Introduction

Wormhole-routing networks [15] are a popular approach to build fast, low-latency interconnects both for interhost communication over small areas, and intrahost communication for multiprocessor machines. The main characteristics of wormhole networks are wormhole or cut-through packet switching [12] to reduce the latency, and some strict form of flow and congestion control in order to avoid packet losses due to buffer overflow. Since the physical layer of these networks is in general reliable, network sources can assume that, if they send out a packet, or 'worm', on the network, it will eventually be received at the destination. An important example of wormhole communication network is the crossbar-based Myrinet [1], which uses back-pressure flow control, and source routing, to implement local and campus networks with Gb/s links. Myrinet will be taken as the reference architecture in this paper.

Ideally, the wormhole LAN should perform as a very high speed LAN: host interfaces should attach to a seamless 'LAN cloud'. As such, it should provide low latency (the worm can be injected whenever the host interface is free), guaranteed delivery (after the worm has been completely output to the network), and low cost. The wormhole LAN offers several advantages over traditional shared-medium LANs. The major advantage is a better scalability, due to the mesh topology, with multiple simultaneous transmissions. These advantages, however, do not come for free. The choice of blocking (and possibly backpressure) upon contention, instead of dropping worms, is prone to deadlocks. Another limitation is multicasting, which is the focus of the paper. We require multicasting to be as reliable as in traditional LANs: once the multicast message has been handed to the network, it can be assumed to be delivered to all the members of the multicast group. Several multicast groups can be simultaneously active in the network, and any member of the group can be a source of multicast messages. Also, multicast latency must be comparable to unicast latency (as is in Ethernet). Providing such multicast service is an easy task on shared-medium LANs with linear topology, but it becomes anything but easy on deadlock-prone wormhole networks with mesh topology.

*This work was partially supported by the Italian National Research Council (CNR). The authors are indebted to Mario Gerla and Simon Walton of the Computer Science Department at UCLA for fruitful discussions on the work reported here.

2. Wormhole Routing Networks

Wormhole routing is often the scheme of choice in low-latency, high-speed networks [12, 15, 13]. Asynchronous wormhole-routing LANs are a promising and cost-effective alternative to ATM for high-bandwidth, low-latency applications in the local area and campus environments. They can also be considered an interesting alternative with respect to the IEEE 802.3z Gigabit Ethernet.

In wormhole routing, the (variable-size) unit of information transfer is a worm, which can range in length from a few bytes to several thousand bytes. Low latency is achieved by using cut-through or wormhole routing instead of the conventional store-and-forward approach [12]: each intermediate node forwards the worm to the desired output port (if available) as soon as the head of the worm is received, without waiting for the entire worm to be assembled. Thus, the worm can stretch across several nodes and links at any one time. Upon blocking due to a busy output port, while in cut-through switching the tail of the blocked worm is totally reassembled in the switch, in wormhole routing the worm is 'frozen' in place (by backpressure), possibly keeping upstream links and buffers in a busy state. This means that cut-through switching is more demanding in terms of input buffer space at switches.

Wormhole networks are designed as loss-free systems, and congestion is prevented by means of some form of flow control [6]. The most common flow control scheme

in wormhole LANs is backpressure: worms are blocked on their way to the destination if no resources are available along the path. As circular waits are possible, backpressure is prone to deadlocks [3, 15]. The issue of deadlocks in computer systems has been well studied (see, e.g., [10]).

One popular approach to avoiding deadlocks is to restrict the routes that a message may take towards the destination. Such an approach was adopted in the Autonet [18], and in Myrinet [14, 1]. Another well known approach in store-and-forward packet networks to avoid deadlocks consists in defining a partial order on the buffers (hierarchy of packet buffer pools) at each node [5, 8, 9], and allocating them on various criteria like number of hops traveled, the route which the packet is traveling on, etc. An approach similar to the hierarchy of buffer pools, called *virtual channels* technique [3, 6], combines restricted routing and buffer partition to achieve deadlock-free, minimum-distance routing in the wormhole routing context. Multiplexing of several virtual channels on the same physical channel requires that the switch have the ability to interleave worms. Virtual channels allow minimum distance routing, but require an increase in the hardware complexity of the node.

3. Multicasting in Wormhole Networks

Multicasting is a challenging issue in wormhole networks since the simplicity of the switches makes it difficult to support multicast trees in the fabric. Furthermore, the lack of internal store-and-forward buffering, coupled with the simultaneous propagation of the message on multiple paths increases the risk of deadlocks [7].

An alternative to providing multicast in the switching fabric is to handle the replication of multicast messages in the Host Adapter interfaces, i.e., at the interface between hosts and the 'LAN cloud'. Although very reliable, this solution does not scale well, since the source interface is tied up during the entire multicast session; latency can thus become large.

Another solution, also based on host adapter multicasting, is reported in [19], where a buffer credit scheme (which is the extension of the Illinois Fast Messages credit scheme [16]) is proposed.

Reliable multicasting can also be provided at the transport level (i.e., above switch and host interfaces) [4, 11]. While recognizing that transport level multicasting is an effective alternative, in this paper we pursue the goal of making worm LANs behave in all respects like a very fast Ethernet. Thus, we will focus only on network level reliable multicasting.

MULTICASTING BY THE HOST ADAPTER

A viable approach to multicasting is to do worm replication and retransmission entirely in the host adapter card. That is, the host adapter, which is attached to a switch through a bidirectional link, upon receiving the multicast worm, copies it to its local host and at the same time retransmits it to one or more hosts in the same multicast group. This way, multicast worms appear as normal unicast worms to switches. No modification to the switching fabric is required; rather, all the multicasting functions are implemented in the host adapter.

In the host adapter multicasting scheme, the hosts in the multicast group are organized in a predefined logical structure (a linear list or a tree, as discussed later). For the sake of simplicity, we consider only the case in which the forwarding of multicast messages is done by the members of the multicast group, although this is not necessarily the optimal solution. The basic operation carried out by an intermediate host adapter is to (a) recognize the worm as a multicast worm (by the multicast group ID carried in the header); (b) determine, from a pre-computed multicast group table, the subsequent host(s) (in the linear list or tree) to which the worm must be forwarded; (c) perform the transmission of the worm to each one of the immediate successors in a cut-through mode and, at the same time, copy the worm to the local host.

In the simplest implementation, host are arranged in a linear list, and each worm is sent to the host interface of the next host in the multicast path as if it were a unicast worm, until it has reached all hosts belonging to the multicast group.

The larger the traffic loading the network, the more critical the availability of hosts buffering becomes. Indeed, if a worm is backpressured by the network and cannot be successfully forwarded (because other worms are using buffering and transmission resources in the network), it can be blocked at a host. The host adapter must in this case provide for full buffering of at least one worm in its memory, in order to avoid potential path deadlocks.

THE ACK/NACK PROTOCOL

In our paper, we adopt with some improvements the ACK/NACK policy that was proposed in [7], which is more consistent with the wormhole routing philosophy, namely we acquire resources as we go, and block and wait when resources are not available. As an example, let us assume that host adapter A has buffered the entire worm X. It then forwards worm X (in cut-through mode) to successor B. The worm header bears the indication of the worm size. If B has enough buffer space, it accepts the worm and sends A a short control worm, the ACK. On receiving the ACK, A can discard worm X from the buffer at the transmission end. If host adapter B lacks buffer space, it drops the worm, returning a NACK control worm as explicit notification to the sender. On receiving the NACK, A stops transmission to B, and resumes transmission of the worm (from scratch) after a timeout, or when it gets a READY control worm from B; all the while, the worm sits in A's buffer. This solution allows a complete decoupling of the multicast strategy from the switch-level deadlock prevention and flow control algorithms. Therefore, temporary lack of buffers in a node does not tie up network resources.

The ACK/NACK protocol described in [7] is a simple form of window protocol with unit windows (i.e., a stop-and-wait window protocol). To achieve a better performance when the host-to-host path exhibits large latency and the destination host buffer has a non-negligible probability of being busy, we consider the case in which host interfaces can

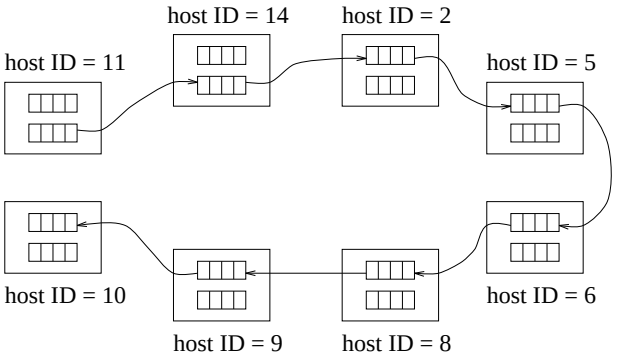

Figure 1. Deadlock prevention using two buffer classes

buffer $K \geq 1$ incoming multicast worms, logically forming a FIFO reception queue. The ACK control worm is issued only when the corresponding multicast worm hits the head of this FIFO queue, while the NACK control worm is issued as soon as an incoming multicast worm must be discarded due to lack of buffer space in the host interface card. The READY control message is sent in response to the oldest NACK-ed multicast message only when all multicast worms, but one, have cleared the buffer; it is not issued earlier in order to avoid undesired queue build-ups, hence delay increases, when the network traffic grows. This flow control mechanism reduces the probability of discarding worms at the host adapter, thereby significantly improving network performance, as will be shown later.

BUFFER DEADLOCK PREVENTION

The step-by-step buffer allocation scheme described in the previous section prevents path deadlocks. However, there is the potential for another form of deadlock: the adapter *buffer* deadlock. This situation can occur, for example, when two hosts are trying to send each other a worm: neither buffer becomes available because each one is waiting for the other to complete the transmission. A simple way of preventing this type of deadlock is to assure that all multicast messages propagate from lower ID to higher ID hosts in the multicast group. For example, the hosts can be ordered in a list by increasing ID number, and the multicast propagates from lowest ID to highest ID.

However, since multicast messages may originate from an arbitrary host in the group, they do not generally start from the lowest ID host, and at least one order reversal, i.e., a transmission from a higher ID to a lower ID host, may occur. To handle this situation, we can divide the multicast buffers in each host adapter in two classes: the first (or *lower*) class is used before the reversal; the second (or *upper*) class after the reversal. Each buffer class can store up to K worms. Fig. 1 shows a two buffer class allocation example. The multicast originates at host 11 and terminates at host 10.

More generally, the two-buffers scheme does avoid buffer deadlocks if host buffers (instead of host adapters) can be numbered in such a way that all multicast paths traverse buffers in increasing order. It is easy to see that this numbering is always possible at the cost of restricted host-to-host routing. Indeed, all routing schemes proposed in next section avoid buffer deadlocks using the two buffers in each host adapter.

The assignment of host IDs can be done separately for each multicast group in the network, or for the entire network. However, in the former case each host interface must have a pair of buffers for each multicast group to which it belongs. To avoid complexity, in particular in presence of dynamic multicast groups, we consider the latter case, i.e., network-wide host ordering.

The choice of the host ordering is critical. Of course, we want to assign, in each multicast group, adjacent IDs to hosts which are as close as possible to each other from a topological standpoint. If instead we devise a way of assigning a global ordering to all hosts in the network, we must export this information to the host ordering inside a multicast group. Although this solution does not guarantee to minimize the end-to-end delay inside a multicast group, it seems a sensible ordering guideline when dealing with general topologies.

To identify the global host ordering, we may consider the graph of host connectivity induced on the network topology: a complete graph, whose nodes are the topology hosts, and edges are weighted with some metric accounting for resources costs used in traversing the unicast path between the hosts, taking into account possible routing limitations.

4. Host Adapter Multicasting Strategies

We consider three different multicasting strategies: (1) multiple unicasts at the source, (2) the linear path, where each host in the multicast group sends a copy of the message to at most one other host, and (3) the rooted tree, where one host can duplicate the message to more than one other host.

These strategies have their drawbacks and tradeoffs, of course. While sending multiple copies seems more appealing for small, scattered multicast groups, the other strategies account for a shorter transmission activity at the originator, and can exhibit superior performance depending on the topology and on the traffic load, as will be shown later on. Indeed, since host adapters are equipped with a single transmitter (and a single receiver), when multiple copies are to be forwarded by a particular host interface, they must be transmitted one after the other. This means that, while the first copy can be forwarded in cut-through mode, the second and the subsequent copies must wait for the transmission of the previous copy to complete. This *serialization* of the multiple copies can remarkably increase latencies, especially at low loads.

Note that the multiple unicast and the linear path can be viewed as extreme cases of the tree approach: the multiple unicast is a tree of depth one, while the linear path is a skewed tree (i.e., of width one).

4.1. Multiple unicasts at the source

The first, straightforward multicasting strategy that we consider is having the originator host adapter take it upon itself to send a unicast copy of the multicast worm to each group member except for itself. This is similar to how broadcast is currently implemented in Myrinet.

The advantage of this strategy is simplicity: since relaying of the multicast worm is not necessary, no buffer deadlock can occur, hence no ACK/NACK protocol nor buffer reservation are necessary.

On the negative side, one may object that the approach does not scale well, as several resources are needlessly used for large multicast groups, and the latency is increased due to the serialization of the copies at the single source transmitter.

4.2. Multicasting on a linear path

In the case of multicasting over a linear path, we form a directed circuit among the hosts that are in the multicast group. There are two possible approaches to transmitting around the circuit: the *ring-like* routing, and the *bus-like* routing. They both require the two-buffers scheme to prevent buffer deadlocks.

For the ring-like routing, hosts originating a multicast transmission are required to send only one worm, which will visit group members one by one. Assuming that there is a host ordering inside the group (which reflects the global host ordering of the network), we force multicast worms to travel from lower ID hosts to higher ID hosts, as detailed above.

In the bus-like variant of the linear path strategy, two worm transmissions are required to the source host adapter (except for the lowest and highest ID hosts), one towards higher ID hosts, the other one towards lower ID hosts. Host operations upon reception of a worm remain the same. When the worm reaches either the lowest or the highest ID host, it is not forwarded any further.

As for the global host ordering required by the two-buffers deadlock prevention strategy, although we do not have any 'optimal' algorithm, we can refer to the graph of host connectivity induced on the network topology, and look for a Hamiltonian path (which is not difficult to find on a complete graph). Hosts can be numbered (and thus ordered) according to their position on the Hamiltonian path,

4.3. Multicasting on a rooted tree

Intuitively, the latency of multicast messages on linear paths should be reduced by allowing some hosts to make several copies of an incoming worm and to forward them separately to different subsets of the multicast group. The delivery of a multicast worm can be described in this case as the traversal of a rooted tree, whose root is the host which originates the worm, and the remaining nodes are the other hosts of the multicast group[1]. Each node has as many children nodes as the number of different copies it generates, and these children are the hosts to which the various copies are transmitted. Note that this scheme requires that k different trees are built for each multicast group made up by k hosts.

One may further observe that, at light load (i.e., when there is no contention for bandwidth or buffers), the tree will lead to higher latency than multicasting on a linear path because of the serialization of multiple copies at hosts with two or more children. A limitation on the maximum branching factor (i.e., on the number of children) can limit this drawback. On the other hand, if the load is heavy, practically requiring full worm reassembly at almost every host, the parallelism of the tree will help reduce latency. The tree will also help reduce latency when cut-through operation in the host adapter is not available, as is the case in Myrinet.

It is worth noticing that this scheme does not guarantee the total ordering of multicast worms.

4.4. On computing 'optimal' multicast trees

Given a source host belonging to a multicast group, it is possible to build several multicast trees which differ from each other in the latency of the multicast messages injected by the source. We define the *optimal* multicast tree (OMT) as a multicast tree which minimizes the latency of the multicast worm generated by its root, i.e., the time taken to reach the last host of the group.

In order to build an OMT, a tradeoff between the cost of longer paths and the cost of serialization at branching points must be considered. As a matter of fact, sometimes a longer path (i.e., a path crossing more transmission links) may be more effective than a shorter one in which several serializations take place. In some special cases, such as when all the arcs of the host connectivity graph have the same weight, and when no load dependency of the transmission and serialization costs is assumed, techniques like dynamic programming or branch-and-bound may be effectively used to compute OMTs, as for instance done in [17]. However, since both the transmission cost and the serialization cost depend on the network load, the computation of OMTs is in general very difficult, if not unfeasible. We therefore propose a simple heuristic algorithm, based upon the host connectivity graph, which allows to compute suboptimal multicast trees in which the influence of the network load is neglected.

Let us now discuss our heuristic for the computation of (suboptimal) multicast trees. Each path in a multicast tree is associated with a cost directly proportional to the latency of the multicast message following that path, and the cost of the multicast tree is defined as the cost of its largest-cost path. In computing path costs, we neglect traffic-dependent delays, i.e., delays due to backpressure blocks and ACK/NACK retransmissions. We define the cost of a path $v_1, v_2, \ldots, v_n$ as $\sum_{i=1}^{n} (\#C(v_i) - 1) \times CT + TX(v_i, v_{i+1})$, where $\#C(v_i)$ is the order of transmission at host v_i of the copy of the multicast destined to host v_{i+1} (i.e., the cost of serialization at branching points), CT is the time taken by a single copy operation (one worm transmission time), and $TX(v_i, v_{i+1})$ is the cost of a worm transmission from v_i to v_{i+1} (equal to the propagation delay, since we assume cut-through mode of operation).

[1] As noted above, we consider only the case in which the branching points of the tree (where message duplications occur) are host adapters belonging to the multicast group.

To determine a (suboptimal) multicast tree we compute the spanning tree of the host connectivity graph (restricted to the hosts of the multicast group, under the constraints dictated by the two-buffers scheme either in the wrap-around or in the up/down versions) which has the minimum cost. Note that this is different from the classical "Minimum Spanning Tree" problem, due to the different cost definition, and to the deadlock prevention constraints.

Consider first the wrap-around rooted tree. The algorithm examines each host in the multicast group in increasing ID order (allowing at most one order reversal, as noted above) and computes a spanning tree with root s by determining the minimum-cost path connecting s with each of them. The path from s to a given host v_i is computed by comparing the cost of reaching v_i either by sending to it a copy of the multicast worm directly from s, or by forwarding the worm in cut-through mode to one of the nodes with a lower ID than v_i (for which a path has been already computed) and then by making an additional copy.

For the up/down rooted tree, our heuristic algorithm proceeds as follows. A subtree comprising all hosts with higher ID than the source host is first built according to the procedure described above for the wrap-around case. Note that no ID order reversal occurs in this case. Lower ID hosts are then successively scanned in decreasing ID order. For each considered host, the lowest cost path is selected among all paths obtained by reaching it directly from each host already belonging to the tree, possibly taking into account the cost of extra copies. The considered node is then added to the tree, and the next lower ID host is considered.

5. Simulation Results

In this section, we present selected simulation results to assess the performance of the multicasting strategies presented in the paper.

The network configuration we consider is a 8×8 Bidirectional Manhattan Street Network (BMSN) [2], with link propagation delay equal to 50 byte times, corresponding to 125 m at 640 Mb/s. Multicast worm sizes are randomly extracted from a truncated geometric distribution, with average 100 bytes and maximum 1000 bytes. The slack buffer size at switches is equal to 5 times the link size (250 bytes in our case). Therefore, under low load conditions, we can assume that a multicast worm can possibly stretch through 2 switches, on average. One host is attached to each switch. Hosts are originally numbered 0 through 63 (see Fig. 2) and the global host ordering is so that hosts lying on odd rows are assigned increasing IDs from left to right, while the reverse applies to hosts on even rows. Note that this numbering follows an Hamiltonian path on the topology.

The virtual channel approach is used to avoid path deadlocks for unicast transmissions, since it allows minimum distance routing and better performance (see [6]). The two-buffers scheme, when needed, is used to prevent buffer deadlocks.

There is only one multicast group, whose members are chosen along a triangle (hosts 0, 1, 2, 3, 13, 15, 16, 17 and 31 in Fig. 2). The host order inside the multicast group reflects the global BMSN ordering. We consider cases where both multicast and unicast traffic are present. Unicast worm sizes are randomly extracted from a truncated geometric distribution, with average 400 bytes and maximum 4000 bytes. The Poisson unicast traffic load is evenly divided among all hosts and is altogether always equal to 6.4 (normalized value with respect to the link capacity). Thus, a single host generates a normalized unicast traffic equal to 0.1 (i.e., it transmits on all its output links 0.1 times the data rate of one link). The multicast load too is Poisson, and evenly partitioned among all hosts in the group. In our implementation, multicast transmission and multicast worm forwarding have priority over local unicast transmissions.

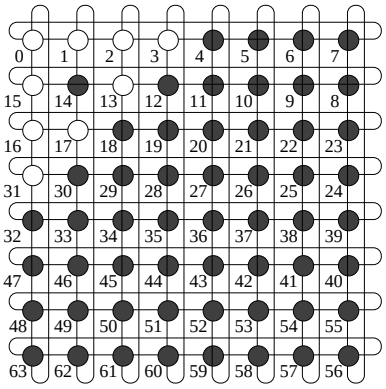

Figure 2. The considered network topology. Nodes belonging to the multicast group are highlighted.

The capacity K of the FIFO reception queue in host interfaces for the ACK/NACK protocol is supposed to be equal to 4 worms.

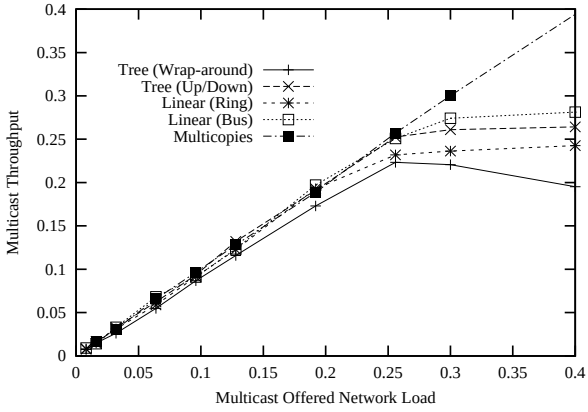

Figure 3. Normalized throughput for multicast transmission without unicast

It is possible to catch a first glimpse of how the different

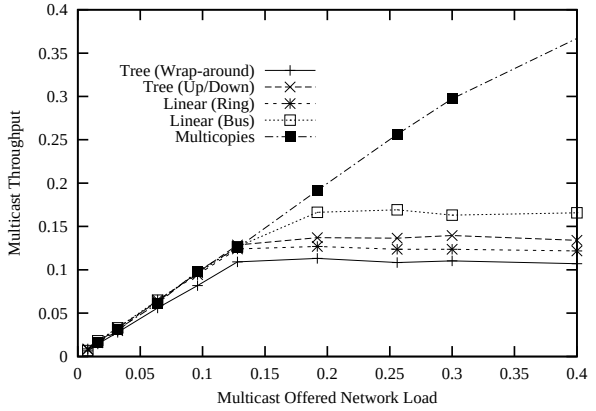

Figure 4. Normalized throughput for multicast transmission with unicast

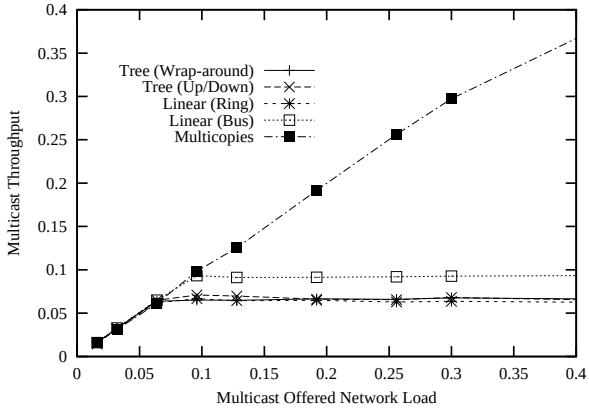

Figure 5. Normalized throughput for multicast transmission with unicast when $K = 1$

routing strategies work by looking at Figs. 3 and 4, showing the normalized multicast throughput for scenarios with or without the presence of unicast traffic, as a function of the overall normalized multicast offered load. It can be seen that sending multiple unicast copies at the originator host attains a higher throughput and is hardly sensitive to unicast traffic, whereas routings over linear paths and rooted trees are not as efficient when the network traffic increases, and their throughput is significantly reduced by the presence of unicast traffic. The tree-based strategies, in particular, do not show the expected good results. This is most likely due to the ACK/NACK retransmission protocol, which throttles multicast traffic due to the unit-window flow control. Moreover, a congestion increase due to the transmission of control worms (ACKs, NACKs, and READYs) may occur, as can be in part seen in Fig. 3, where the throughput curve for the tree/wrap-around strategy reaches a maximum and then decreases for increasing offered load.

Note that using more sophisticated window protocol (with larger windows) may not be the desired solution, due to increased delays for large loads: larger windows entail larger buffers, which tend to be full when the load increases.

Losses are experienced for normalized multicast throughputs above 0.2, corresponding to $0.2/9$ normalized multicast host throughput (multicast messages per average message transmission time), or to $8 \times 0.2/9 \approx 0.18$ normalized unicast load per host. Note that worm LANs, which are characterized by low hardware costs, hence permit to install excess bandwidth in the network, are normally supposed to operate at low normalized offered traffics.

The capability of forming a FIFO reception queue for multicast messages at host interfaces, which was proposed in this paper, greatly improves performance. For example, Fig. 5 reports the same results of Fig. 4 when $K = 1$, i.e., with at most one multicast worm per buffer.

More insight can be gained from Figs. 6 and 7, referring to the same set of simulations as above and showing the observed *message delays* (i.e., the time elapsed between the insertion of a multicast message in the transmission queue and the reception of the last byte by all members of the multicast group), measured in byte times, versus the normalized offered network load. It can be seen that routings on linear paths have some edge over multiple unicast at low loads and in absence of unicast traffic. Clearly, doing multiple copies easily results in high latency at low loads due to the need for serialization, while the linear path choice benefits from a single copy traveling along an unloaded network. The 'optimal' tree routings show in this case marginal delay advantages with respect to linear paths only at low loads.

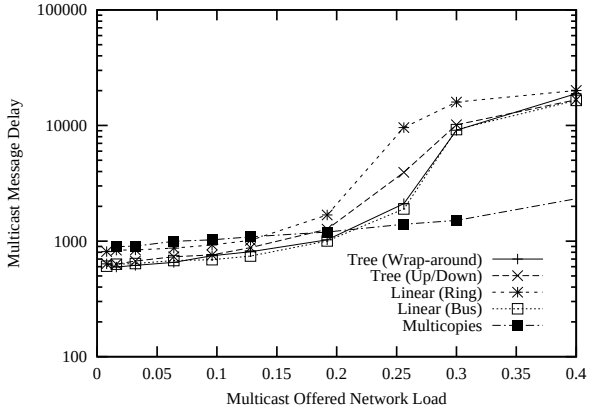

Figure 6. Message delay for multicast transmission without unicast

These results underline the utmost sensitivity of these routings to network traffic load, as was pointed out in previous sections. Such observations, coupled with the fact that the tree-based strategy falls short of providing the optimum choice at all loads, could spark a debate over the design of the

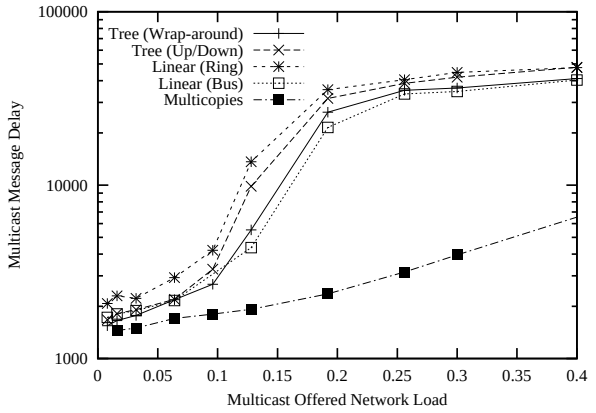

Figure 7. Message delay for multicast transmission with unicast

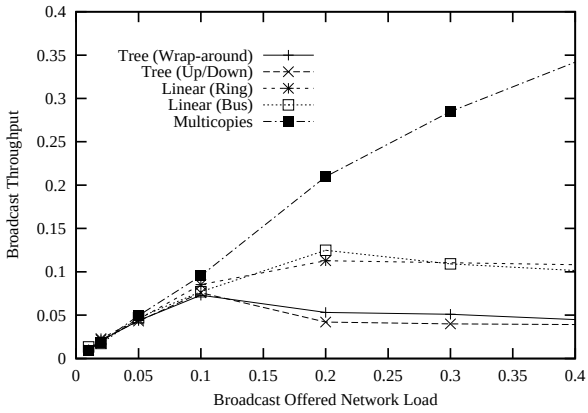

Figure 9. Normalized throughput for broadcast transmission with unicast

tree-based strategy algorithm. One might argue that the chosen algorithm does not weigh paths against their congestion level, thus feeding routing tables only suboptimal solutions. However, providing for a load-sensitive algorithm, beside increasing the algorithm's complexity, would probably induce non-stationary, unpredictable behaviors in the routing tables updating process, making it extremely difficult to preserve tables' consistency.

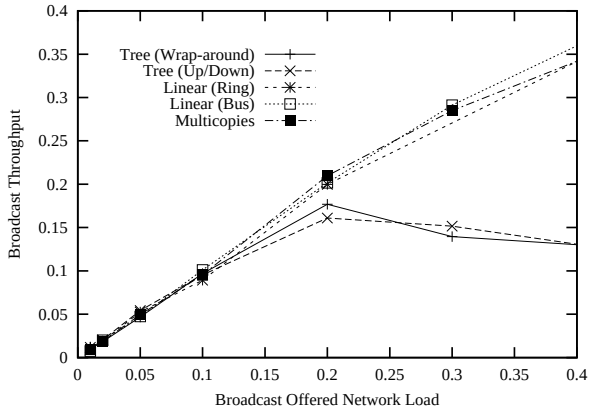

Figure 8. Normalized throughput for broadcast transmission without unicast

The behaviors pinpointed in the multicast case can be easily mapped onto the broadcast case. Figs. 8 and 9 show the throughput for all routing strategies with and without unicast traffic. Rather surprisingly, sending multiple unicast messages to every node of the network again performs better than both linear path routings and rooted tree routings. The gap between the latter two is even wider than in the multicast case. Only in absence of unicast traffic (which however is an unlikely situation), the linear path approaches provide performance similar to the multicopies strategy. The indications that can be drawn appear to be more favorable to the multicopies strategy since they relate to a setting where an optimal linear path can be identified quite easily (i.e., nodes 0 through 63 in increasing order). An explanation of this behavior is suggested by the way the routing strategies use networks and user resources: while, on the one hand, sending a single copy to all nodes (or two copies to half of the nodes) has a relatively low impact on network resources because it – theoretically – reduces the number of links a broadcast worm traverses, on the other hand the use of the ACK/NACK protocol that comes with the linear path and rooted tree routings throttles the multicast throughput and congests host adapter buffers, thus slowing down the process as the network load increases. Simulation results effectively help us understand which effect is predominant.

Figs. 10 and 11 complete the picture providing the message delay in the broadcast case as a function of the normalized network load, with and without unicast traffic. Note that at low loads the two rooted tree strategies exhibit the expected reduction in message delays, due to the large amount of parallelism permitted by rich trees. This is true both with and without unicast traffic. These results confirm the original intuition that the increased complexity of the rooted tree approach can provide reduced delays, at the cost of a significant reduction of the network capacity. Thus, tree-based strategies can be interesting if we accept the statement that (a) reducing latencies is critical in wormhole networks, and (b) wormhole LANs are over-provisioned with bandwidth, hence load is very light.

6. Conclusions

We have studied solutions for reliable, efficient, low-latency multicasting at the host adapter in wormhole networks using fully distributed schemes, without centralized

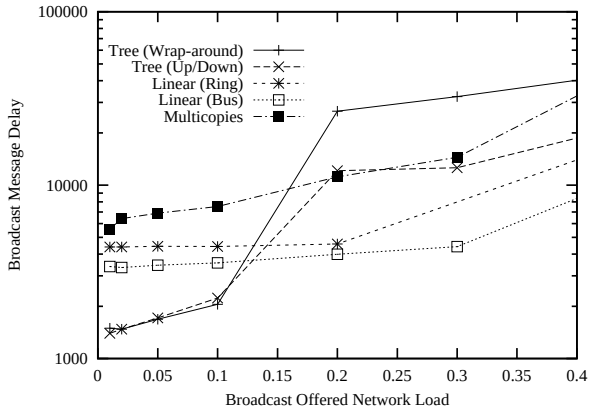

Figure 10. Message delay for broadcast transmission without unicast

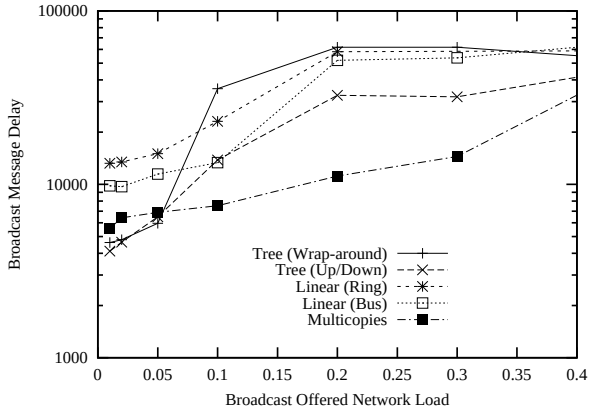

Figure 11. Message delay for broadcast transmission with unicast

controller. We have studied three solutions – multiple unicasts, linear path and rooted tree – which are actually closely related.

The linear path offers lower latency at very light loads (by virtue of cut-through at each node), while at heavier loads it degrades to store-and-forward routing. No significant differences between the bus-like and the ring-like versions were observed.

We expected the tree scheme to provide lower latency than the linear path because of parallelism. Simulation results show that it becomes true only for large multicast groups and at light loads. The cost is a reduction in the maximum achievable network throughput. No significant differences between the up/down and the wrap-around versions were seen. The multiple unicast approach, despite its scalability problems, has the big advantage of simplicity, avoiding adapter-buffer deadlock prevention mechanisms.

References

[1] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, and W.-K. Su. Myrinet: A Gigabit-Per-Second Local-Area Network. *IEEE Micro*, 15(1), Feb. 1995.

[2] F. Borgonovo and E. Cadorin. Locally-Optimal Deflection Routing in the Bidirectional Manhattan Network. In *Proceedings of IEEE INFOCOM '90*, pages 458–464, June 1990.

[3] W. Dally and C. Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Transactions on Computers*, C-36(5):547–553, May 1987.

[4] S. Floyd, V. Jacobson, S. McCanne, C.-G. Liu, and L. Zhang. A Reliable Multicast Framework for Lightweight Sessions and Application Level Framing. In *Proceedings of ACM SIGCOMM '95*, pages 342–356, Aug. 1995.

[5] M. Gerla and L. Kleinrock. Flow Control: A Comparative Survey. *IEEE Transactions on Communications*, COM-28(4):553–574, Apr. 1980.

[6] M. Gerla, E. Leonardi, F. Neri, and P. Palnati. Congestion Control in Asynchronous, High-Speed Wormhole Routing Networks. *IEEE Communications Magazine, feature topic on Flow and Congestion Control*, 11(34):58–69, Nov. 1996.

[7] M. Gerla, P. Palnati, and S. Walton. Multicasting Protocols for High-Speed, Wormhole-Routing Local Area Networks. In *Proceedings of ACM SIGCOMM '96*, Aug. 1996.

[8] I. Gopal. Prevention of Store-and-Forward Deadlock in Computer Networks. *IEEE Transactions on Communications*, COM-33(12):1258–1264, Dec. 1985.

[9] K. Günther. Prevention of Deadlocks in Packet-Switched Data Transport Systems. *IEEE Transactions on Communications*, COM-29(4):512–524, Apr. 1981.

[10] R. Holt. Some Deadlock Properties of Computer Systems. *ACM Computing Surveys*, 4(3):178–196, Sept. 1972.

[11] S. Kasera, J. Kurose, and D. Towsley. Scalable Reliable Multicast Using Multiple Multicast Groups. In *Proceedings of ACM Sigmetrics*, June 1997.

[12] P. Kermani and L. Kleinrock. Virtual Cut-through: A New Computer Communication Switching Technique. *Computer Networks*, 3(3):267–286, Sept. 1979.

[13] J.-P. Li and M. Mutka. Priority Based Real-Time Communication for Large Scale Wormhole Networks. In *Proceedings of International Parallel Processing Symposium*, pages 433–438, May 1994.

[14] Myricom, Inc. *Myrinet in Brief*, 1993.

[15] L. Ni and P. McKinley. A Survey of Wormhole Routing Techniques in Direct Networks. *IEEE Computer*, 26(2):62–76, Feb. 1993.

[16] S. Pakin, M. Lauria, and A. Chien. High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet. In *Proceedings of Supercomputing '95*, Dec. 1995.

[17] J. Park, H. Choi, N. Nupairoj, and L. Ni. Construction of Optimal Multicast Trees Based on the Parameterized Communication Model. In *Proceedings of the International Conference on Parallel Processing*, volume I, pages 180–187, 1996.

[18] M. Schroeder, A. Birrell, M. Burrows, H. Murray, R. Needham, T. Rodeheffer, E. Satterthwaite, and C. Thacker. Autonet: A High-Speed, Self-Configuring Local Area Network Using Point-to-Point Links. *IEEE Journal on Selected Areas in Communications*, 9(8):1318–1335, Oct. 1991.

[19] K. Verstoep, K. Langendoen, and H. Bal. Efficient Reliable Multicast on Myrinet. In *Proceedings of the 1996 International Conference on Parallel Processing*, volume 3, pages 180–187, Aug. 1996.