Multicast in Wormhole-Switched Torus Networks using Edge-Disjoint Spanning Trees [1]

Honge Wang† and Douglas M. Blough‡

† *Myricom Inc., 325 N. Santa Anita Ave., Arcadia, CA 91006,* ‡ *School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332-0250*

A tree-based multicast algorithm for wormhole-switched networks which makes use of multiple edge-disjoint spanning trees is presented. The disjoint spanning-tree multicast, or DSTM, algorithm provides deadlock-free multicast routing that is fully compatible with unicast. Application of the DSTM algorithm to 2-dimensional torus networks is considered. A family of constructions of two spanning trees in the torus is given along with a formal proof of their edge-disjointness. Two constructions from this family are selected and shown to produce diameters no greater than twice that of the torus. Flit-level simulation results are presented to show that DSTM outperforms the best single spanning tree multicast approach by up to a factor of two. The DSTM algorithm is also simulated for different spanning tree constructions. The results show that our novel tree construction is significantly better for multicast than those produced by a general tree construction method that applies to arbitrary-topology networks [16]. Finally, two approaches to providing single link fault tolerance with DSTM are presented and evaluated.

Key Words: Deadlock freedom, edge disjoint spanning trees, multicast, torus network, wormhole switching.

1. INTRODUCTION

Multicast communication in a multicomputer system involves one node sending a message to a subset of the other nodes in the system. Multicast can be used to build many useful operations such as barrier synchronization, cache invalidation in distributed shared memory systems, and collective communication as defined in MPI. Since these operations are widely used, low communication latency and small latency variation for multicast messages are extremely important.

Among the existing switching techniques, wormhole switching has proven to be the most practical low-latency communication technique for multiprocessor systems and cluster computing applications [14, 15]. Recent research has begun to investigate how to implement deadlock-free multicast efficiently with wormhole switching. Most of this work has focused

[1] This research was supported by the National Science Foundation under Grant CCR-9803741.

on path-based approaches that use multidestination worms [10, 13, 15, 20]. Path-based approaches suffer from either extremely long paths (leading to high latency and large latency variation) or multiple costly message startups.

Research on tree-based approaches [7, 11] has proposed variants of up-down routing on a globally-agreed spanning tree. In basic up-down routing, a spanning tree is constructed and is known to all nodes. A message is first transmitted along up links which lead the message towards the root, and is then sent along down links which lead the message away from the root. If messages are confined to using only the links of the spanning tree, then the root of the tree becomes a severe bottleneck. Gerla [7] and Libeskind-Hadas [11] provide variations of up-down routing for multicast that allow off-tree links to be used in some cases but their approaches still result in unbalanced traffic in the network. With most of the traffic concentrating on tree links, congestion problems still occur, especially at the tree links close to the root.

In this paper, a tree-based multicast algorithm, referred to as disjoint spanning tree multicast (DSTM), is proposed. DSTM uses multiple edge-disjoint spanning trees in a network and makes use of modified up-down routing to provide deadlock-free multicast independently within each tree. In this paper, we focus on implementation and evaluation of DSTM in 2-dimensional (2-D) torus networks.

A scalable header scheme for single-tree multicast is first introduced, followed by the routing algorithm which makes use of a combination of source routing and distributed routing. A construction of two edge-disjoint spanning trees in 2-D torus networks is then given, leaving only two links unused in the entire network. Each unicast message is routed using the shorter of the paths in the two spanning trees. Each multicast message is assigned completely to one of the two trees at random. Since all but two links are used as tree links, traffic is better balanced throughout the network as compared to single tree approaches. Both analytic and simulation results demonstrate that significant performance improvements are achieved by our algorithm.

Our tree construction is compared with the Roskind and Tarjan algorithm [16] which gives a construction of multiple edge-disjoint spanning trees in arbitrary networks. We show that our tree construction produces trees with significantly shorter maximum and average path lengths compared to the Roskind and Tarjan algorithm applied to 2-D torus networks. Finally, we present and evaluate two methods for tolerating single link faults using dual edge-disjoint spanning trees, assuming global fault information and local fault information respectively.

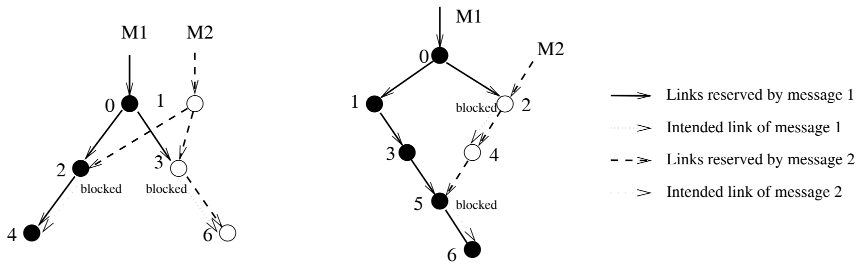
The DSTM Algorithm is sufficiently general to be used in any wormhole-switched network in which multiple edge-disjoint trees can be constructed. This includes most common networks possessing a regular structure and many irregular networks as well. For these networks, DSTM can use the full resources of the network more efficiently than single tree or single path approaches.

2. RELATED WORK

Recent research on hardware supported multicast in wormhole-routed networks can be broadly divided into path-based and tree-based approaches. Path-based approaches either construct a single path spanning all nodes of the network [10, 13, 20] or use an underlying deadlock-free unicast routing algorithm to construct multiple paths which together cover the destinations of the multicast [15]. One drawback of the path-based approaches is that the path lengths can be extremely long, especially in the former approach, and this leads to high latency variation. In the latter approach, multiple message startups are required which cause worm transmissions to be serialized. This approach also incurs multiple startup penalties, each of which can be a significant fraction of total unicast message delay.

Tree-based multicast algorithms attempt to deliver the message to all destinations in a single multi-head worm that splits at some routers. Recent research on hardware supported tree-based multicast in wormhole-routed networks [7, 11] uses up-down routing with a globally-agreed spanning tree to avoid deadlock. Up-down routing was originally proposed for unicast communication. In up-down routing, one node is chosen and a spanning tree is constructed rooted at this node. All links are designated as *up* or *down* links with respect to the root. *Tree links* are the links on the spanning tree, while *non-tree links*, or *cross links*, are the remaining network links that are not on the tree. A link is *up* if it goes from a node at a lower level in the tree to a node at a higher level. A link from a node at one level to a node at a lower level is a *down* link. For nodes at the same level, node IDs are used to classify links as either up or down. Up-down routing routes a message along zero or more up links, followed by zero or more down links. Since up-down routing uses links in a strictly monotonic order, circular waiting is prevented and deadlock freedom is achieved for unicast.

When up-down routing is used for multicast, routers need to be augmented with replication capability. Messages are delivered to all destinations in a single multi-head worm that splits at some routers. Each branch of the message may advance independently after the split. However, deadlock is an issue and the problem is exacerbated if cross links are used. With cross links, one router may have multiple predecessors. This can introduce dependencies between the branches of different multicast trees as shown in Figure 1 (a). In this example, two multicast messages M1 and M2 both want to reach destination nodes 4 and 6. Message 1 is waiting for the links from node 3 to node 6 which is held by message 2, while message 2 is waiting for the link from node 2 to node 4 which is held by message 1. Multiple predecessors also introduce dependencies between the diverging branches of the multicast tree for a single multicast worm. In the example shown in Figure 1 (b), a multicast worm is deadlocked by a unicast worm. In this example, message 1 wants to reach destination nodes 4 and 6, and message 2 wants to reach destination node 6. This situation can occur when message 1 is requesting the link from node 2 to node 4, which is held by message 2, and message 2 is requesting the link from node 5 to node 6, which is held by message 1. This leads to a deadlock as is evident from the figure.


M1
M2
0
1
2
3
4
6
blocked
blocked
M1
M2
0
1
2
3
4
5
6
blocked
blocked
Links reserved by message 1
Intended link of message 1
Links reserved by message 2
Intended link of message 2

(a) Message 1's destinations: (4, 6)
Message 2's destinations: (4, 6)

(b) Message 1's destinations: (4, 6)
Message 2's destination: (6)

FIG. 1. Deadlock between (a) two multicast worms and (b) a multicast worm and a unicast worm.

In [7], Gerla, et al. allow unicast messages to use cross links as shortcuts. Multicast is implemented by a unicast from the source to the root followed by a multicast to all destination nodes initiated by the root. Multicast messages are restricted to tree links only. Multicast messages are totally ordered by the root and therefore deadlock will not occur between two multicast messages. However, when a multicast message is blocked by a unicast message at an intermediate node, it has to suspend its transmission and yield its path to the unicast message. Its transmission is resumed after the unicast message finishes using those links. This prevents deadlock between unicast messages and multicast messages. However, a procedure to reassemble multicast messages is then required at intermediate routers. Gerla's design favors unicast traffic but the approach can also be modified to favor multicast traffic. However, in either case, the intelligence required in the router to handle the blocking situation complicates the router design considerably.

Libeskind-Hadas, et al. [11] loosen the tree link requirement in their single phase adaptive multicast (SPAM) algorithm. SPAM distinguishes between down tree links and down cross links. A unicast message is routed along zero or more up links, followed by zero or more down cross links, followed by zero or more down tree links. A multicast message uses this approach to reach the least common ancestor of its destination set. It is restricted to down tree links after that. The algorithm specifies that if multiple output ports are required at a single intermediate router, they must be reserved atomically. This ensures that two multicast messages do not deadlock at port reservation time. The performance of SPAM is evaluated in [11] and a complete proof of its deadlock freedom is given in [12].

The above two algorithms, although they loosen some of the restrictions of pure up-down routing, still suffer from the problem of traffic that is not balanced between tree links and non-tree links. This produces a potential congestion problem, especially at the tree links around the root while under-utilizing most of the cross links. In general, a spanning tree in a network consisting of N nodes uses $N-1$ links as tree links. This is only a small portion of the links in most commonly used regular topologies and does not increase with the addition of links. A large portion of the scarce network resources, and thus the communication bandwidth, is wasted.

A method to better balance the traffic load is to construct multiple edge-disjoint spanning trees in the network, using as many links as tree links as possible. Although, to our knowledge, we are the first to study the multiple tree approach for wormhole-routed multicast, others have studied the combinatorial problem of constructing multiple edge-disjoint spanning trees in a network. A number of works have studied this problem in hypercubes, either focusing on the existence or on the construction of multiple edge-disjoint spanning trees [1, 2, 3, 4, 8, 9, 18]. Roskind and Tarjan [16] presented a construction of a maximal set of edge-disjoint spanning trees of minimum total edge cost in an arbitrary connected graph. The Roskind and Tarjan algorithm, although it provides a general construction for an arbitrary topology network, might not produce trees that are well suited for multicast.

This paper proposes the use of multiple edge-disjoint spanning trees for multicast communication in direct networks. We introduce a scalable header scheme for tree-based multicast that works with the routing procedure at each router. Considering the 2-dimensional torus network as the topology of the target system, we give a construction of two edge-disjoint spanning trees in the network. Flit-level simulation results show greatly reduced latency and significantly postponed network saturation compared with existing single spanning tree approaches. Comparison is also carried out between the structures of our trees and those produced by the Roskind and Tarjan general tree construction algorithm. It is shown that our tree construction yields significantly shorter combined diameter and average distance. Single link faults can be tolerated by DSTM also, resulting in only relatively small increases in diameter and average distance.

Another recent tree-based approach for meshes is based on the concept of a quad-branch multicast tree [23]. While this approach is tree-based, it sends separate copies of a multicast message into sub-trees of the source and hence it produces multiple message start-ups. It also uses virtual channels and employs the input-buffer-based switch architecture [19], which assumes that routers have enough buffer space to store entire multicast packets. This capability is used to break potential deadlocks. Hence, the approach is more like virtual-cut-through switching than wormhole switching in this regard.

3. MULTICAST ALGORITHM

The Disjoint Spanning Tree Multicast (DSTM) Algorithm is a deadlock-free tree-based multicast algorithm. It constructs multiple edge-disjoint spanning trees in a network and uses our deadlock-free multicast routing algorithm modified from SPAM [11] within each tree. Deadlock freedom is ensured between messages in different trees because the trees are completely edge-disjoint. The fact that a maximum number of links are used as tree links better balances the traffic in the network. This section introduces the DSTM Algorithm, first describing the multicast routing procedure used within a single spanning tree and then discussing how the overall algorithm works using multiple trees.

3.1. Single Tree Multicast

In our algorithm, we make use of a modified version of the SPAM [11] algorithm as the underlying deadlock-free multicast routing algorithm. Only tree links are used. A multicast message is first routed to the least common ancestor (LCA) of its destination set along zero or more up links followed by zero or more down links. The message is then split at the LCA and is restricted to using only down links after that. In the special case where the multicast has only one destination, i.e, a unicast, the LCA is the destination and the multicast algorithm is reduced to a unicast algorithm.

A multicast message uses an asynchronous multi-header worm [14] to reach its destinations. Multiple branches advance independently without coordinating with each other. Header flits are used to set up the routers along the path(s) while data flits are pipelined along all branches, being replicated at all splitting routers. A data flit is replicated at a splitting router only if all enabled outgoing links have free buffer space. When one branch is blocked, some of the trailing flits (header flits, or data flits, or both) may be blocked behind a splitting router. In this situation, other non-blocking branch(es) may continue advancing with the flits that already passed the blocking point. Empty flits or idle time slots are added to those branches until the blocking is resolved.

3.1.1. Scalable Header Scheme

Our tree-based multicast algorithm requires atomic link reservation in order to avoid deadlock at link reservation time. In [11], this approach is proposed with headers that consist of a bit string having one bit for each possible destination. This approach, however, is not scalable. In this subsection, we present a new scalable header scheme which enables atomic link reservation.

Header flits carry the routing information either implicitly or explicitly [5, 6]. Implicit routing information can be either the address of a destination node or the offset to it.

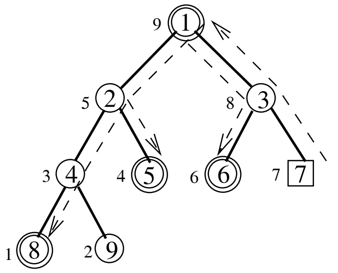
Intermediate routers make routing decisions locally on receiving header flits carrying implicit routing information. Explicit routing information is often used in source routing where the source prepares the complete routing information for all routers along the path. Intermediate routers simply set up the routers as indicated. With multicast, it is not feasible to carry all destinations' information in a single header flit. However, with wormhole routing where fixed size buffers or even a single buffer are present at the routers, atomic link reservation requires a router to be aware of all requested outgoing links based on the first arriving header flit without waiting for the complete destination set to arrive.

We propose a scalable header scheme which uses a combination of implicit and explicit routing information. All-header encoding scheme [5] is used with each header flit carrying one address plus a few ancillary bits, explicitly indicating the link reservation information at the addressed router. The node address is used by intermediate routers to route the flit based on local routing decisions and the link reservation information tells a designated router the requested outgoing links for the message. Since a splitting router needs to be aware of all outgoing links requested by a multicast, a header flit is needed for each splitting node in addition to each destination node. A partial order is enforced so that the header flits for all nodes that are on a single path follow the path order. This is to ensure that the first header flit that arrives at a splitting router contains the necessary information for atomic link reservation. The header flits for nodes that are in different branches after the splitting node may have any order in the message.

In the example of a network connected by routers with four inter-router connection ports, an outgoing link is referred to as "O_i" with i being the port number to which it connects. One more port is used for communication between a router and its local node, where a consumption link forwards data from the router to the local node. The port corresponding to the consumption link is henceforth denoted by "C", . The header flits then have a format of $\{$"$Node$", "C", "O_1", "O_2", "O_3", "O_4", "S"$\}$. Field "Node" carries the ID or the address of the router to be reached. Fields "C", "O_1", "O_2", "O_3", "O_4" are each a single bit, carrying explicit outgoing link reservation information for the addressed router. A value of 1 indicates the corresponding link is requested by this multicast. Bit "S" is a splitting flag indicating whether this flit has passed the first splitting node or not. In a multicast, this bit is initialized to 0 for all header flits at the source node and is modified to 1 at the first splitting point. Unicast messages do not use the "S" field.

This header scheme is described with an example in Figure 2 where a binary multicast spanning tree is constructed in the network, assuming the left branches of all nodes are O_1 and the right branches are O_2. Node IDs are the numbers inside the circles.

In Figure 2, a binary spanning tree is constructed rooted at node 1. A multicast message is sent by node 7 to the set of destination nodes $\{1, 5, 6, 8\}$. Node 7 prepares header flits as shown in Figure 2. The dashed lines indicate the paths taken by the message. The message is first routed to the least common ancestor node 1, then split at node 1 and routed down to both branches. The left branch is again split at node 2. A total of 3 paths exist in this multicast. The header flits include all destination nodes $\{1, 5, 6, 8\}$ and the splitting nodes $\{1, 2\}$. Partial order should be maintained for nodes on each path. Previous header flits set the routers along the path(s) appropriately until the last splitting router(s) and are then stripped from the header. A trailing header flit follows one of the paths till the last reserved router on its path, and is then routed to a farther destination, setting the routers along its way. In the example shown in Figure 2, the path orders $\{1, 2, 8\}$, $\{1, 2, 5\}$ and $\{1, 6\}$ should be maintained in the header. However, nodes on different paths, such as nodes 6 and 8,



Header

	Node	C	O_1	O_2	O_3	O_4	S
h1	1	1	1	1	0	0	0
h2	2	0	1	1	0	0	0
h3	5	1	0	0	0	0	0
h4	6	1	0	0	0	0	0
h5	8	1	0	0	0	0	0

Legend — Number in the circle: Node ID

Number to the left of the circle: Node's PN number

□ : Source node ◎ : Destination node

FIG. 2. Header scheme in tree-based multicast

can be in any order after the splitting node 1. We have header flits ordered as $\{1, 2, 5, 6, 8\}$ in Figure 2. Many different orders can produce the necessary partial order, however, such as $\{1, 2, 5, 8, 6\}$, $\{1, 6, 2, 8, 5\}$, etc. An algorithm by which source nodes can compute a correct header encoding for a multicast message is given in Appendix A.

3.1.2. Routing Procedure

In conjunction with our scalable header encoding scheme, a distributed routing procedure is carried out at intermediate routers. Upon the arrival of a header flit at a router, field “$Node$” is first checked. If field “$Node$” contains the address of the local node, this header flit is stripped from the header and consumed locally. The link reservation information in this flit is used to set up the router. A 1 in field “C” indicates that the local node is a destination and the consumption channel needs to be reserved. Otherwise, the local router is a splitting router. A node can also be both a destination and a splitting node when $C = 1$ and at least one $O_i = 1$. If the node is a splitting node, the router sets a local split flag and all tailing headers will have their “S” field set as they pass through.

For header flits that are not addressed to the local node, field “S” is used to identify whether this flit has passed the least common ancestor (LCA) where the first split occurs. A header flit that has not yet reached the LCA should follow the path reserved previously by the first header flit. When the “S” bit is set, the flit is routed directly towards its destination based strictly on local routing decisions. Field “S distinguishes a header flit in up-routing from the same flit in down-routing, in which the flit is destined to a descendant node of the local router, but needs to be routed up to the LCA first. In the example shown in Figure 2, the header flit for node 6 passes through node 3 twice, once going up the tree and again coming down. The flit has the value $\{$6, “1$''$, “0$''$, “0$''$, “0$''$, “0$''$, “0$''$$\}$ along the up route and the value $\{$6, “1$''$, “0$''$, “0$''$, “0$''$, “0$''$, “1$''$$\}$ along the down route. This enables the router at node 3 to distinguish these cases and determine how to route the flit in each case.

In summary, our single tree multicast scheme is scalable and uses a combination of source routing and distributed routing. The source node prepares header flits for the splitting routers and the destination routers, indicating requested outgoing links at an addressed router by accompanying the addresses with a few bits of link reservation information. Intermediate routers need to have enough intelligence to route a header flit in a distributed fashion based on local decisions, following up-down routing. Partial order ensures that the intermediate routers are ready for header flits coming later along the same path.

3.2. Disjoint Spanning Tree Multicast Algorithm

3.2.1. Algorithm Overview

In a 2-D N-node torus network, there are $2N$ links. Since a spanning tree uses $N - 1$ links in any network with N nodes, a maximum of 2 edge-disjoint spanning trees can be constructed in a 2-D torus network. In a network comprised of two edge-disjoint spanning trees, there is a path in each spanning tree connecting any two given nodes. To save network resources, we use the tree with the shorter path to transmit unicast messages. While for multicast messages, since the amount of resources that will be taken can not be easily foreseen, a single tree is picked randomly for each multicast.[2] Alternatively, the destination set for an individual multicast could be partitioned into two multicast messages each transmitted in one of the trees to a subset of the destination nodes. However, this will increase computational complexity and introduce more traffic, and is therefore likely to have worse performance.

For both unicast and multicast messages, our routing algorithm described in Section 3.1 is used in each tree. All transmission is restricted to tree links. We give a construction in Section 4 which leaves only two links as non-tree links. Traffic distributes among all but two network links. Deadlock freedom is achieved within each tree as in standard up-down routing [7]. Note that, to guarantee that the trees are truly edge-disjoint, two consumption channels are needed at each node, one for each tree to consume the messages arriving over that tree. DSTM is then deadlock-free because the two trees are completely edge disjoint.

3.2.2. Router Requirements

Two distinct approaches have been proposed for tree-based multicast using cut-through switches. The approach adopted in this paper is the use of asynchronous multiheader worms with wormhole switching. This approach requires coordinated flow control among multiple branches after a split point in the multicast tree. It also requires that links representing multiple children of a node in the multicast tree be reserved atomically. The primary advantage of this approach is that it can achieve deadlock freedom within the wormhole switching framework so that only a single flit buffer per virtual channel is required at each router. The requirements for coordinated flow control and atomic link reservation are necessary whether the multicast algorithm uses a single tree as in [11] or multiple trees as in DSTM.

An alternate approach proposed in [23] prevents deadlock and also avoids coordinated flow control by buffering complete multicast packets within a router. This is similar to virtual cut through switching in preventing deadlock. It also allows flits to be copied to one output port while buffering them for later copying to a distinct output port. This avoids the need for coordinated flow control. The primary disadvantage of this approach is the need for very large buffers inside the routers.

[2] Under this assumption, when a long multicast message is split into multiple multicast packets which are each sent through a random tree, packets could arrive out of order at some destinations. As in any packet-switched network, there must be a layer of software to assemble and reorder packets at destination nodes. Note that a similar situation can occur if adaptive routing algorithms are employed, even for unicast. If desired, this situation can be avoided by simply forcing all packets from the same message to use the same multicast tree.

The primary additional cost imposed on the router when going from a single multicast tree to multiple trees is the need to store information in each router about each tree. So for DSTM, the storage cost inside each router is approximately doubled. It is not necessary to store the entire multicast tree in each router. In our approach, nodes are ordered according to a post-order traversal of the tree. Each node is then required to store, for each of its sub-trees, the lowest and highest numbered nodes within the sub-tree. For the binary tree construction presented in this paper, this requires $4 \log N$ bits of storage for one tree vs. $8 \log N$ bits for two trees. Minor additional logic is required for DSTM to determine which tree is being used from the link on which the header flit arrives. The information of that tree is then used to make the routing decision, which is done just as in the single tree multicast algorithm.

4. CONSTRUCTION OF EDGE-DISJOINT SPANNING TREES IN TWO-DIMENSIONAL TORUS NETWORKS

In this section, we prove that any 2-dimensional torus network contains two edge-disjoint spanning trees. The proof is constructive and, hence, provides a method to produce two such spanning trees given any 2-D torus network. We also discuss the structure of the trees produced by our construction.

THEOREM 4.1. *Two edge-disjoint binary spanning trees can be constructed in any 2-dimensional torus, with 2 edges unused.*

Proof. Given a $K \times M$ torus (K columns and M rows) as shown in Figure 3, let nodes R_1 and R_2 be two starting nodes to construct tree 1 and tree 2, respectively. Henceforth, refer to these two trees as T_1 and T_2. Nodes are denoted by N subscripted by their coordinates, with the first coordinate representing the column and the second coordinate representing the row. For example, R_1 and R_2 are denoted as N_{x_1,y_1} and N_{x_2,y_2} respectively. In general, R_1 and R_2 could have the same x coordinate or y coordinate, or R_1 and R_2 could even be the same node. However, in this constructive proof, we assume $x_1 \neq x_2$ and $y_1 \neq y_2$.

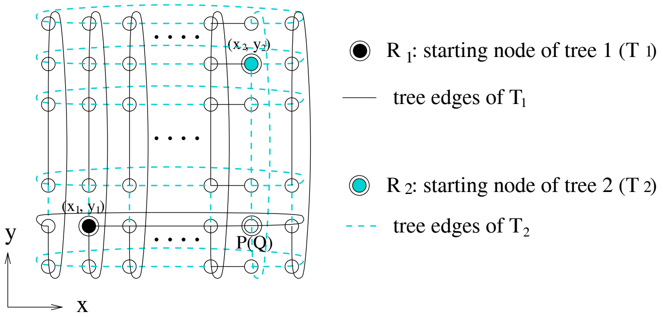

(x2, y2)
(x1, y1)
P(Q)
y
x
● R 1: starting node of tree 1 (T 1)
tree edges of T1
● R 2: starting node of tree 2 (T 2)
tree edges of T2

FIG. 3. Construction of dual edge-disjoint spanning trees in a torus network

For any integer i used as x coordinate in a torus with K columns, we define $i^+ = (i+1) \bmod K$ and $i^- = (i-1) \bmod K$. For any integer j used as y coordinate in a torus with M rows, we define $j^+ = (j+1) \bmod M$ and $j^- = (j-1) \bmod M$.

We define $H_{i,j}$ to be the edge connecting node $N_{i,j}$ and $N_{i^+,j}$, and $V_{i,j}$ as the edge connecting node $N_{i,j}$ and N_{i,j^+}. $H_{*,j}$ denotes all the edges in row j, i.e. $H_{*,j} = \{H_{i,j} : i \in \{0, ..., K-1\}\}$. Note that the edges of $H_{*,j}$ form a cycle. Similarly, $V_{i,*}$ denotes all the edges in column i, i.e. $V_{i,*} = \{V_{i,j} : j \in \{0, ..., M-1\}\}$, which also form a cycle. In a 2-dimensional torus, $H_{i,j}$ and $V_{i,j}$ with $i \in \{0, ..., K-1\}, j \in \{0, ..., M-1\}$ contains all the edges of the network. As a natural extension of this notation, we use $V_{*,*}$ to denote all column edges in the network, $H_{*,*}$ to denote all row edges, and $N_{*,*}$ to denote all nodes.

In Step 1, consider the tree T_1 initially containing only the node R_1. Starting from R_1, search the torus along $+x$ direction, wrapping around if necessary and adding the nodes and edges to T_1 as visited, until all the nodes in row y_1 are in T_1. At the end of this step, a total of K nodes and $K-1$ edges are in T_1, where the nodes are N_{*,y_1}, i.e. $\{N_{i,y_1} : i \in \{0, ..., K-1\}\}$, and the edges are $H_{*,y_1} - H_{x_1^-,y_1}$. Since the edges of H_{*,y_1} form a cycle on row y_1, the absence of edge $H_{x_1^-,y_1}$ breaks the cycle and makes T_1 a linear array which is a degenerate tree.

In Step 2, similar to Step 1, consider the tree T_2 initially containing only the node R_2. Starting from node R_2, search the torus along $-y$ direction, wrapping around if necessary and adding the nodes and edges to T_2 as visited, until all the nodes in column x_2 are in T_2. At the end of this step, a total of M nodes and $M-1$ edges are in T_2, where the nodes are $N_{x_2,*}$, i.e. $\{N_{x_2,j} : j \in \{0, ..., M-1\}\}$, and the edges are $V_{x_2,*} - V_{x_2,y_2}$. As in Step 1, this makes T_2 a linear array which is a tree.

In Step 3, for each node that is already in T_1 except N_{x_2,y_1}, repeat the following operation, henceforth referred to as *single column search*. Search the nodes along $-y$ direction, wrapping around if necessary and adding the nodes and edges to T_1 as visited, until all the nodes with same x coordinates are in T_1.

Each single column search starting from node $N_{i,y_1}, i \in \{0, ..., x_2-1, x_2+1, ..., K-1\}$ results in a total of $M-1$ nodes being added to T_1. The added nodes are $N_{i,*} - N_{i,y_1}$. At the end of this step, including the nodes that are already in T_1 from Step 1, T_1 has the node set $N_{*,*} - N_{x_2,*} + N_{x_2,y_1}$. Since searches start from tree nodes which are connected to all the other nodes, the adding of the search path which consists of the nodes and the edges along which nodes are visited maintains the connectedness of T_1.

In each single column search starting from node N_{i,y_1}, edges $V_{i,*} - V_{i,y_1}$ are added to T_1 also. The acyclic feature holds within any single column search because the absence of edge V_{i,y_1} breaks the cycle formed by $V_{i,*}$ on column i and a linear array is added to T_1 via a single node N_{i,y_1}. All single column searches and their resulting linear arrays are separate from each other because of the different columns investigated. No cycle is formed. The newly added edges in this step are $V_{*,*} - V_{x_2,*} - V_{*,y_1}$. Since T_1 is connected and acyclic, it is still a tree.

In Step 4, symmetric operations to Step 3 are done for T_2. Single row searches are done along $+x$ direction starting from every node in T_2 except N_{x_2,y_1}. The achieved node set after this step is $N_{*,*} - N_{*,y_1} + N_{x_2,y_1}$. Edges that are added to T_2 in this step are $H_{*,*} - H_{x_2^-,*} - H_{*,y_1}$. Connectedness and the acyclic feature are maintained for T_2 as in Step 3, which implies that T_2 is still a tree.

At the beginning of Step 5, all nodes except $N_{x_2,*} - N_{x_2,y_1}$ are in T_1. Note that in Step 4, edges $H_{x_2^-,*}$ are not used. In Step 5, add edges $H_{x_2^-,*} - H_{x_2^-,y_1}$ to T_1 (note that $H_{x_2^-,y_1}$ is already in T_1 from Step 1). Since edge $H_{x_2^-,j}$ connects nodes $N_{x_2,j}$ and $N_{x_2^-,j}$, which are already in T_1, these edges bring all remaining nodes in the network into T_1. Hence, T_1 spans all the nodes at the end of this step.

The acyclic feature is maintained because each newly added node has only one edge connected to the rest of T_1, with which no cycle can be formed. Hence, T_1 is a tree containing all nodes, i.e. a spanning tree.

In Step 6, similar to Step 5, the edges that are unused in Step 3, $V_{*,y_1} - V_{x_2,y_1}$, will bring the remaining nodes at the end of Step 4 into T_2, which are $N_{*,y_1} - N_{x_1,y_1}$. T_2 spans all the nodes at the end of this step and is still a tree by the same reasoning as in Step 5.

The fact that T_1 and T_2 are spanning trees is now proved.

Now, consider edge sets used by T_1 and T_2. Table 1 lists the tree considered and the edge set used in each step as specified earlier.

TABLE 1

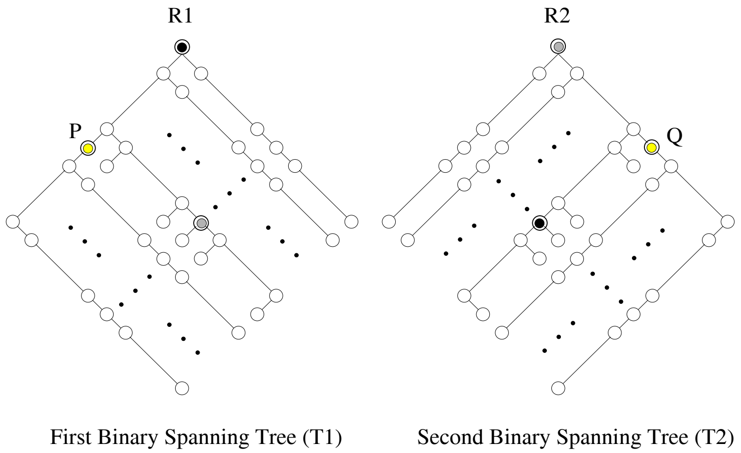
Edges used in T_1 and T_2

Step	Tree	$H_{i,j}$	$V_{i,j}$
1	T_1	$H_{*,y_1} - H_{x_1^-,y_1}$	null
2	T_2	null	$V_{x_2,*} - V_{x_2,y_2}$
3	T_1	null	$V_{*,*} - V_{x_2,*} - V_{*,y_1}$
4	T_2	$H_{*,*} - H_{x_2^-,*} - H_{*,y_1}$	null
5	T_1	$H_{x_2^-,*} - H_{x_2^-,y_1}$	null
6	T_2	null	$V_{*,y_1} - V_{x_2,y_1}$

In Table 1, edges in $H_{i,j}$ column form a non-overlapped $H_{i,j}$ edge connection, which is $H_{*,*} - H_{x_1^-,y_1}$. Edges in $V_{i,j}$ column form a non-overlapped $V_{i,j}$ edge connection, which is $V_{*,*} - V_{x_2,y_2}$. Therefore, the edge-disjoint feature of T_1 and T_2 is proved. All edges but two, $H_{x_1^-,y_1}$ and V_{x_2,y_2}, are used in the two trees.

Finally, in a 2-dimensional torus, every node has four inter-router links. As two edge-disjoint spanning trees are constructed out of these links, every node has at least one link and at most three links in one tree. Therefore, nodes with parent can have at most two children in any tree. This ensures all subtrees are binary. Since root nodes in our construction has two children in each tree, binary feature is then proved. ■

In the construction presented in the Theorem 4.1 proof, the method used to construct T_1 is called a $(+x, -y)$ method, because the sequence of visiting network nodes is along $+x$ direction followed by $-y$ direction. Similarly, the construction method for T_2 is called a $(-y, +x)$ method. The two edge-disjoint spanning trees produced by this construction are shown in Figure 4.



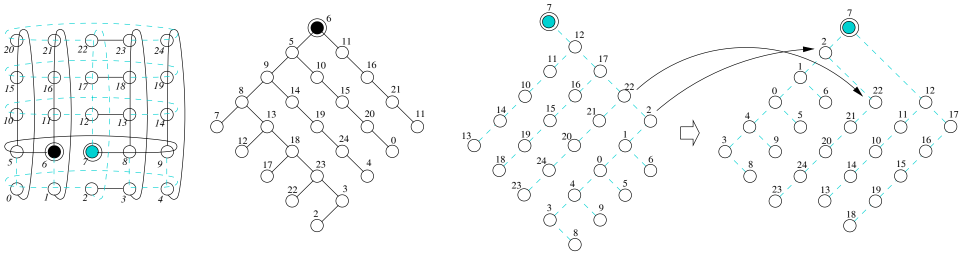
First Binary Spanning Tree (T1) Second Binary Spanning Tree (T2)

FIG. 4. Constructed dual edge-disjoint binary spanning trees

As the root node may become the bottleneck when the network traffic gets higher, it is critical to make the trees binary and balanced with respect to the root. Compared with a tree having higher degree at the root, fewer branches at the root mean larger subtrees that can handle more traffic locally without passing it through the root. Balancing the trees with respect to the root also results in more traffic being handled without going through the root node.

The construction of Theorem 4.1 classifies links into two spanning trees. Using the starting nodes for the construction as the root nodes guarantees that the constructed trees are binary but they may not be balanced. However, it is not necessary to use the starting nodes as the root nodes. For example, in Figure 4, nodes P and Q may be selected as root nodes to better balance the two trees while still maintaining their binary nature. One possible drawback is that nodes P and Q are actually the same node in the network, located at the intersection of the row of R_1 and the column of R_2 as shown in Figure 3.

When the two starting nodes are located in the same row or the same column and are chosen as the root nodes, our construction gives only one branch at the root for one or both trees. These trees can be better balanced by moving a half branch to the other side of the root using the unused wrap-around links while maintaining their binary nature. One such example is shown in Figure 5. The change in tree structure is limited to the nodes on the part of the branch to be moved, i.e. a total of $\lceil K/2 \rceil$ nodes on a branch of K nodes. The remainder of the tree is unchanged.



5 x 5 torus network binary spanning tree 1: T1 binary spanning tree 1: T2

FIG. 5. Construction and adjustment of dual edge-disjoint binary spanning trees in a torus network when the two starting nodes are in the same row

5. PROPERTIES OF DISJOINT SPANNING TREE CONSTRUCTION

In a network comprised of two edge-disjoint spanning trees, we define the distance between any two nodes as the length of the shorter path in the two trees, and we denote this distance by $d(n_i, n_j)$. The combined diameter CD and average distance AD of such a network are then defined as in the single network case, i.e.

$$CD = \max_{\forall n_i, n_j} d(n_i, n_j)$$

and

$$AD = \frac{\sum_{\forall n_i, n_j} d(n_i, n_j)}{n(n-1)/2}$$

As non-shortest paths may be used in the DSTM Algorithm, unicast performance may be impacted compared with algorithms that use shortest paths. In this section, we study the combined diameter and average distance of the trees produced by the construction presented in Section 4. We also compare these values to the same values that result from the trees produced by the Roskind and Tarjan edge-disjoint spanning tree construction algorithm [16] applied to the torus.

5.1. Starting Node Selection

To simplify our study, we look at $\sqrt{N} \times \sqrt{N}$ torus networks which have the most symmetry. In such networks, there are $N \times (N-1)$ possible ways to select two different starting nodes for tree construction. Given a pair of starting nodes, there are 16 distinct methods to construct edge-disjoint trees with our construction method, i.e., use one of $(-x,-y)$, $(-x,+y)$, $(+x,-y)$, $(+x,+y)$ methods to construct T_1 and one of $(-y,-x)$, $(-y,+x)$, $(+y,-x)$, $(+y,+x)$ methods to construct T_2. All methods can be classified into one of the above categories because of the symmetry of the torus network. Therefore a total number of $16N(N-1)$ different constructions exist in a 2-D torus network.

This number can be reduced because of the symmetry of the torus network. First, since there exists a homomorphism that maps any node of the torus network onto any other node in the same network, the first starting node R_1 can be chosen randomly. We denote the coordinates of R_1 to be (R_{1x}, R_{1y}). Second, given the first starting node R_1, if we divide the network into four quadrants originated at R_1 as shown in Figure 6 (b), the selection of the second starting node R_2 will have four nodes as mirrors in each quadrant because the relative relationship between R_1 and R_2 are the same by rotating 90^o. Now we only consider nodes within the rectangular region defined by two corner nodes R_1 and P_1 whose coordinates are $(R_{1x} + \frac{\sqrt{N}}{2}, R_{1y} + \frac{\sqrt{N}}{2})$. Further note that nodes in this region are symmetric with respected to the diagonal from R_1 to P_1 by exchanging x and y as shown in Figure 6 (c). So the second starting node only needs to be selected from the triangle defined by R_1, P_1 and P_2 whose coordinates are $(R_{1x}, R_{1y} + \frac{\sqrt{N}}{2})$ with the number of possible selections of R_2 as $N/8$.

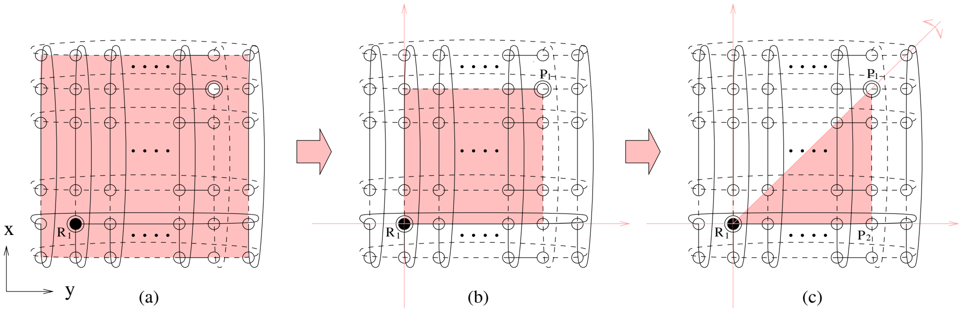


FIG. 6. Selection of starting node R_2.

However, the number of possible selections of R_2 is still quite large. We investigate R_2 along the diagonal from R_1 to P_1 and also along the row from R_1 to P_2. As expected, the minimum combined diameter and minimum average distance do exhibit a pattern. The two quantities both decrease while R_2 is attempted along the diagonal from R_1 to P_1 until it reaches the midpoint of the diagonal, then the quantities increase until R_2 reaches P_1. When R_2 is attempted along the row from R_1 to P_2, both quantities decrease until it reaches the midpoint which is $(R_{1x}, R_{1y} + \frac{\sqrt{N}}{4})$. The best trees along this row are constructed using $(-x,+y)$ method for T_1 and $(+y,-x)$ method for T_2. The minimum combined diameter and minimum average distance then increase until P_2 is reached. The attempt along the row results in better values for the two quantities than the attempt along the diagonal.

Based on the above analysis, the following two constructions are selected for our study.

DSTM–1. Arbitrarily select node R_1 with a coordinate of (R_{1x}, R_{1y}). Select node R_2 with coordinate $(R_{1x} + \frac{\sqrt{N}}{2}, R_{1y} + \frac{\sqrt{N}}{2})$. Construct T_1 using $(+x, -y)$ method starting from R_1 and construct T_2 using $(-y, +x)$ method starting from R_2. This construction is selected because it produces the most well-balanced trees after root node adjustment.

DSTM–2. Arbitrarily select node R_1 with a coordinate of (R_{1x}, R_{1y}). Select node R_2 with coordinate $(R_{1x}, R_{1y} + \frac{\sqrt{N}}{4})$. Construct T_1 using $(-x, +y)$ method starting from R_1 and construct T_2 using $(+y, -x)$ method starting from R_2. This construction is selected because it produces the shortest combined diameter and shortest average distance.

5.2. Properties

THEOREM 5.1. *The combined diameter of the edge-disjoint spanning trees used by DSTM–1 in a $\sqrt{N} \times \sqrt{N}$ torus is $2\sqrt{N} - 1$.*

The proof of Theorem 5.1 can be found in Appendix B.

THEOREM 5.2. *The combined diameter of the edge-disjoint spanning trees used by DSTM–2 in a $\sqrt{N} \times \sqrt{N}$ torus is $2\sqrt{N} - 2$.*

The proof of Theorem 5.2 is similar to that of Theorem 5.1 and is therefore omitted.

Compared with the diameter of $\sqrt{N}$ for $\sqrt{N} \times \sqrt{N}$ torus networks, the diameters for both DSTM–1 and DSTM–2 are increased by no more than a factor of two. In a wormhole switched network, in which the latency is considered to be insensitive to the path length because of the high-speed pipelined transmission and high start up penalty, a path length which is increased by a factor of two is generally acceptable, especially if it allows good multicast performance to be obtained.

THEOREM 5.3. *The header encoding complexity for DSTM multicast with D destinations in a $\sqrt{N} \times \sqrt{N}$ torus with the use of algorithm in Appendix 0.1. is $O(D \log D + N)$.*

The proof of Theorem 5.3 can be found in Appendix C.

5.3. Comparison

An alternative method for constructing two edge-disjoint spanning trees in a torus is to use the algorithm of Roskind and Tarjan [16]. Their algorithm finds a maximal set of lowest-cost edge-disjoint spanning trees in an arbitrary graph. The algorithm considers edges in cost-increasing order. Since, in our problem, all edges have unit cost, the algorithm can consider edges in any order. In fact, considering edges in two different orders can yield two different pairs of edge-disjoint spanning trees. One special case constructs two Hamilton paths when we consider links in a $N \times N$ torus in the following order: first horizontal links $H_{i,j}$ with $i = [0, 1..., N-1]$ and for each i, $j = [i, i^+, ..., i^-]$, followed by all vertical links $V_{i,j}$ with $j = [0, 1, ..., N-1]$ and for each $j \in [0, N-2]$, $i = [j, j^+, ..., j^-]$ and for $j = N-1, i = [0, 1, ..., N-2]$. Figure 7 shows the two Hamilton paths the above order produces in a 5×5 torus network. Since edge order impacts tree structure for the Roskind and Tarjan algorithm, we carried out a set of experiments, in which different random edge orderings were used. For a fixed-size torus, we used 500 different random orderings and used them to compute 500 pairs of spanning trees.

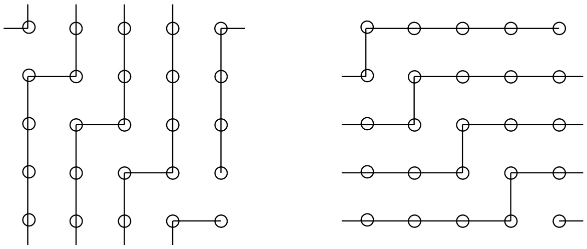
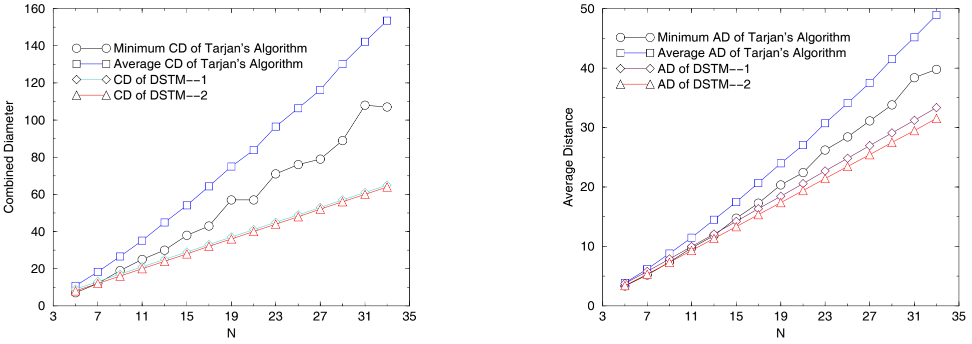


FIG. 7. Hamilton paths produced by the Roskind and Tarjan algorithm in a 5×5 torus network considering links in a special order.

The combined diameter results are listed in Figure 8(a). Minimum combined diameter represents the lowest combined diameter that was found among the 500 pairs of trees. Average combined diameter is the average taken over all 500 pairs. Figure 8(a) shows that for networks of size 9 × 9 or larger, the tree constructions used by DSTM–1 and DSTM–2 have lower combined diameter than the best pair found by the Roskind and Tarjan algorithm in 500 trials. For 33 × 33 networks, the minimum combined diameter found by the Roskind and Tarjan algorithm is about 65% higher and the average combined diameter is more than twice as high as those produced by our constructions.



(a) Combined diameter

(b) Average distance

FIG. 8. Combined diameter and average distance in $N \times N$ torus networks with no faults

Figure 8(b) shows the results for average distance where, for the Roskind and Tarjan algorithm, the average is taken over all pairs of nodes in all 500 pairs of trees. As shown in Figure 8(b), with the increase of the network size, the average distance increases roughly at a linear rate for all three algorithms. However, the increase in the Roskind and Tarjan algorithm is about 51% faster than DSTM–1 and about 59% than DSTM–2. The average distances for their algorithm are higher than those produced by our constructions for all cases considered, with the difference being about 50% for 33 × 33 networks.

6. EXPERIMENTAL RESULTS

To evaluate the performance of the DSTM algorithm, simulation has been done on a flit level simulator, MARS, originally developed at Harvey Mudd College. MARS includes a number of routing and multicast algorithms, including the SPAM algorithm using a single spanning tree [11]. We added the DSTM algorithm into this simulator. The DSTM algorithm is investigated for the tree construction of Section 4 and for several pairs of trees constructed by the Roskind and Tarjan algorithm. The Roskind and Tarjan constructions considered are a well balanced pair of trees with roughly average combined diameter and average distance over 500 random trials, a worst case pair of trees which corresponds to two Hamilton paths, and a well balanced pair of trees selected out of the 500 trials to have small combined diameter and average distance. The experiments are done in a 16×16 torus network. No virtual channels are needed by any of the algorithms.

The following parameters are used for all the experiments reported in this section. The startup latency is 10 microseconds. The router setup time for each message header is 40 nanoseconds. Channel propagation delay is 10 nanoseconds. The number of consumption channels at each node is two to maintain the disjointness between the spanning trees for DSTM algorithm. To ensure a fair comparison, two consumption channels are used for the SPAM algorithm also. The number of injection channels at each node is one which permits only one multicast to be launched from a node at any time. Each message contains 128 flits. The simulations were run until the results were within 1% of the mean, using 95% confidence intervals for all simulations.

The first set of simulation results examines the message latency of different algorithms in a network with traffic composed of 90% unicast messages and 10% multicast messages, with the average arrival rate varying from 0.001 to 0.014 messages per node per microsecond and following a Poisson distribution. The simulated algorithms include SPAM, DSTM–1, DSTM–2 and DSTM with the three constructions of dual edge-disjoint spanning trees produced by the Roskind and Tarjan algorithm that were detailed earlier. In this experiment, each multicast message has 48 destinations randomly selected from the system. The results are shown in Figure 9.

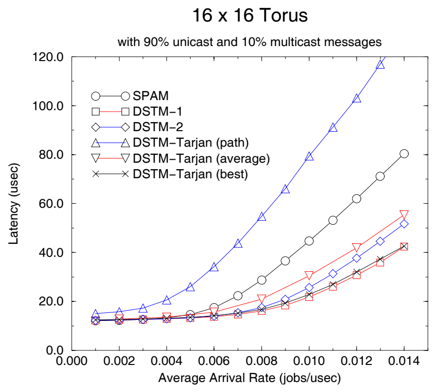


FIG. 9. Latency versus average arrival rate for traffic composed of 90% unicast and 10% multicast.

The results shown in Figure 9 are enlightening as to the relative merits of different tree-based multicast approaches. DSTM–Tarjan(path), however, is a pathological case where the edge-disjoint trees are actually Hamilton paths. This data point should not be used to draw conclusions about path-based approaches because the algorithm still treats the paths as trees and uses up-down routing. Hence, DSTM–Tarjan(path) will result in some unnecessary back-tracking of messages that would not occur if a path-based algorithm were used on the Hamilton paths.

A clear distinction can be seen in Figure 9 between the single tree approach (SPAM) and the four non-pathological dual tree approaches. All of the dual tree approaches significantly outperform the single tree approach. This clearly demonstrates the benefits obtained by the multiple edge-disjoint tree approach through its efficient and more well-balanced use of the links of the network.

In comparing the various dual tree approaches, Figure 9 shows that DSTM–1 is the best but that DSTM–Tarjan(best) is quite close to it in performance. Interestingly, DSTM–Tarjan(best) actually outperforms DSTM–2 despite DSTM–2's substantially lower combined diameter and average distance. This indicates that the use of trees that are well balanced with respect to the root is relatively more important than minimizing their combined diameter and average distance. DSTM–1 and DSTM–Tarjan(best) are both extremely well balanced in this respect while DSTM–2 is less balanced. Combined diameter and average distance do influence the results but are relatively less important than balancing the trees. The influence of distance is demonstrated by the superiority of DSTM–1 over DSTM–Tarjan(best) because the trees used in DSTM–Tarjan(best) are actually more balanced around the root than those of DSTM–1. The distance effect is also seen in the relatively poor performance of DSTM–Tarjan(average) for which the trees are extremely well balanced but have far higher combined diameter and average distance than the other DSTM algorithms.

In the simulations shown in Figure 10, the same algorithms are studied for pure multicast traffic with the number of destinations uniformly distributed from 5 to 10 and each destination being selected randomly from the whole network.

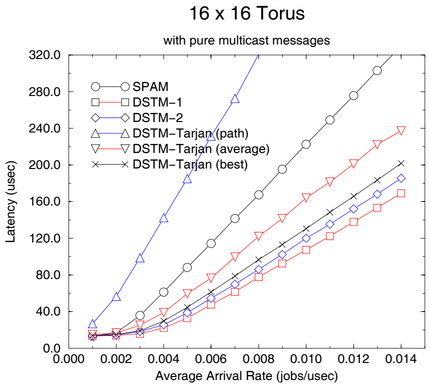


FIG. 10. Latency versus average arrival rate for pure multicast traffic.

Figure 10 again demonstrates the advantages of multiple tree approaches via the lower latencies exhibited by the non-pathological DSTM algorithms relative to SPAM's single tree approach. DSTM–1 is once again the best dual tree algorithm but in the pure multicast setting its performance is significantly better than DSTM–Tarjan(best).

Despite the existence of spanning trees generated by the Roskind and Tarjan construction that have performance close to that of DSTM–1, there are a number of benefits that make our construction method valuable. Our construction led to the first proof that two edge-disjoint spanning trees exist in any size torus. The Roskind and Tarjan construction does not tell us a priori how many spanning trees can be constructed in a given network. Furthermore, our construction is valid for any size torus, whereas the Roskind and Tarjan construction algorithm must be applied again for each new network. Good constructions can be obtained from their algorithm only by running a large number of random experiments. In addition, there is no way to know which of the pairs of trees produced by the Roskind and Tarjan algorithm will result in good multicast performance without simulating their operation. Thus, using their construction to design a DSTM algorithm having good performance for a particular network involves many costly random trials and several sets of extremely time-consuming flit-level simulation experiments. In contrast, the proven upper bound on diameter for any size torus and the simulation results obtained over a wide range of sizes mean that our construction is well characterized and has known good multicast (and unicast) performance for virtually any torus.

A final advantage of our tree construction is that it always produces binary trees. As discussed earlier, binary trees reduce the amount of traffic that must travel through the root compared to trees of higher degree. Binary trees also minimize the number of links that might have to be reserved atomically at any router. Thus, fewer links are reserved in advance of their actual use, meaning that throughput is increased compared to trees of higher degree.

7. SINGLE LINK FAULT TOLERANCE

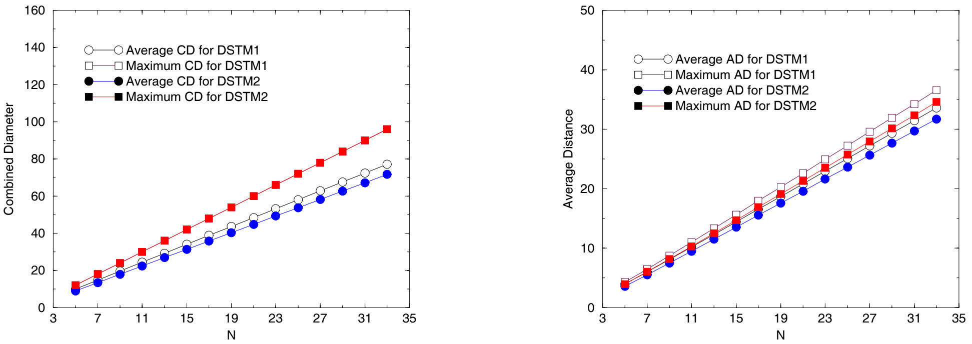
We consider static link faults which occur when a link is unused and does not interrupt a message during transmission. To a routing algorithm, a static fault appears as a permanent obstacle and a router attempts to route the message around it.

In a network containing two edge-disjoint spanning trees, a single link fault will result in at most one tree becoming disconnected. Since the other tree still maintains the connectivity of the network, single link faults can be taken care of with limited modification to our approach.

In general, fault information can be either globally known or only locally known. With global fault information, a distance table can be regenerated at every node for the faulty tree with the length of the faulty link considered as infinity. When a unicast message is present, the tree with the shorter path will be selected to deliver the message. In this situation, multicast messages are always delivered through the good tree. Deadlock freedom is maintained since basic up-down routing is still used.

The combined diameters for DSTM–1 and DSTM–2 in networks with a single link fault and assuming global fault information are shown in Figure 11(a), with the average and the maximum values among 500 trials with a single faulty link chosen at random. The average distances between any two nodes in such networks are shown in Figure 11(b).

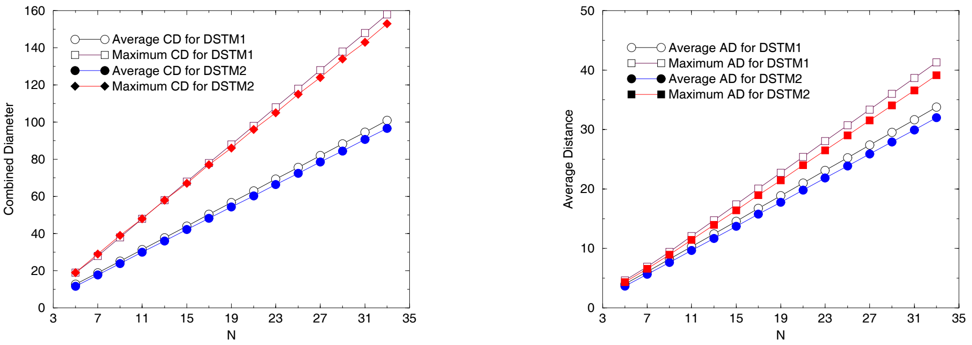
If the fault information is known to neighboring nodes only, fault tolerance can be achieved as follows. A message is routed normally until the faulty link is encountered. The router R at which the message last arrives then routes the message as normal except that, instead of routing the branch of the message to the faulty link, it routes the branch to the local processor. Node R then completely consumes the entire message, re-calculates the header flits according to the good tree as if R is the source and re-injects the message to the good tree. Note that two startup penalties can be incurred in this approach for some messages.



(a) Combined diameter

(b) Average distance

FIG. 11. Combined diameter and average distance for DSTM–1 and DSTM–2 in $N \times N$ torus networks with a single link fault and global fault information.



(a) Combined diameter

(b)Average distance

FIG. 12. Combined diameter and average distance for DSTM–1 and DSTM–2 in $N \times N$ torus networks with a single link fault and local fault information.

In this approach, message transmission will not be deadlocked in a single tree. The good tree can always be relied on to deliver the message when a faulty link is encountered. Since only a single link is faulty, messages can be routed only from one tree to the other but not vice versa. This allows deadlock freedom to be maintained.

The combined diameters for DSTM–1 and DSTM–2 in single-link-fault networks with local fault information are listed in Figure 12(a), shown with the average and the maximum value among 500 trials. The average distances between any two nodes in such networks are shown in Figure 12(b).

To summarize the results of Figures 11 and 12, combined diameters and average distances for the DSTM constructions are increased only modestly when a single link is faulty compared to the fault-free case and the values remain below those produced by Tarjan's algorithm with no faults. The increase is slightly higher for local information than for global information but distances remain quite reasonable even if only local fault information is used.

8. CONCLUSION

In this paper, the DSTM multicast algorithm using edge-disjoint spanning trees was presented. The DSTM algorithm provides deadlock-free multicast routing fully compatible with unicast by making use of a modified up-down routing algorithm. Multicast capability has been successfully added to up-down routing with the scalable header scheme proposed in this paper. Compared with multicast algorithms using a single spanning tree with cross links, DSTM uses more links as tree links and better balances traffic in the network. This produces simulation results that outperform the best single tree approach in 2-dimensional torus networks by up to a factor of two. In addition to its performance benefits, DSTM is able to tolerate single link faults.

An interesting question is how well-suited is DSTM for network topologies other than torus. The minimum number of links required to construct two edge-disjoint spanning trees is $2(N-1)$ (the torus has $2N$). Because two-dimensional mesh networks (without wrap-around connections) have fewer than $2(N-1)$ links, they are not able to produce multiple edge-disjoint spanning trees.

However, DSTM can be extended to k-ary n-cubes with $n > 2$. These networks represent higher-dimensional versions of the 2-D torus. We have been able to construct 3 edge-disjoint spanning trees in k-ary 3-cube networks, and we conjecture that n edge-disjoint spanning trees can be found in an arbitrary k-ary n-cube. $k = 2$ is an interesting special case. Wrap-around connections are redundant in this case and deleting them yields the hypercube topology. It is known that $\lfloor \frac{n}{2} \rfloor$ edge-disjoint spanning trees can be constructed in an n-dimensional hypercube [3]. Adding wrap-around connections would then allow $2 \cdot \lfloor \frac{n}{2} \rfloor$ edge-disjoint spanning trees, proving our conjecture true for $k = 2$ and n even. We are currently studying how to generalize our tree construction and selection methods for arbitrary k-ary n-cubes.

The concept of making use of multiple edge-disjoint spanning trees is sufficiently general for other regular or irregular networks with an algorithm to provide the construction of a maximum number of spanning trees in the network. Current vendors usually provide routers or network interface cards with a fixed number of ports, say P. A maximum of $P/2$ disjoint spanning trees could then be constructed. The actual number is likely to be much smaller in practice because of unused ports and irregular interconnection patterns. An interesting open problem is to find a topology that permits the maximum number of edge-disjoint spanning trees given the number of nodes, the number of routers, and the value of P.

APPENDIX A

Header Encoding Algorithm

We provide a header encoding scheme for multicast in a network with a binary spanning tree. This scheme includes a one time pre-processing step at network configuration time and a calculation executed at a source node before a message is injected into the network. The source node calculation generally follows depth-first search with two parameters, one being the node currently considered and the other being the destination list for this node. The calculation starts with the root node and a complete destination list. The destination list is then partitioned into sub-lists, one for each child of the root. Each child's destination list contains exactly those destinations that are contained in the sub-tree rooted at that child.

The same calculation is recursively executed at each child node with its own destination list.

When the network is configured, a spanning tree is constructed following a post-order tree traversal. An integer number is incrementally assigned to each node during the traversal. This number is henceforth referred to as the PN number of that node. Each node N_i maintains “low” and “high” fields recording the coverage of its subtree. The node with PN number of “low” is the leftmost child in N_i’s subtree. The node with PN number of “high” is N_i itself. Any node with a PN number between “low” and “high” resides in N_i’s subtree (see the example in Figure2). This one time pre-processing step takes time $O(N)$ in a network with N nodes and is needed only at network configuration time. The information of the tree is then distributed to all routers and stored in their memory. When a multicast is present at a source node, the source node first sorts the destination list according to the PN numbers. Algorithm 1 is then used by the source node to generate the header flits, with the root node and the sorted destination list as inputs.

ALGORITHM 1 ($Header\_Calculation$ $(current, dests)$).

```
Input:   $current$ = current node
         $dests$ = destination list sorted according to the PN numbers;
Output:  $header$ = list of header flits, global variable, single header flit has the format
         of [$Node$, “$C$”, “$O_1$”, “$O_2$”, “$O_3$”, “$O_4$”, “$S$”];
begin
  SPLIT_LOCALLY = FALSE;
  if ($dests.length() = 1$ ) OR ($current = dests.last()$) then
    add flit [$dests.last()$, “$1''$”, “$0''$”, “$0''$”, “$0''$”, “$0''$”, “$0''$”] to the end of the header;
    remove $dests.last()$;
    if ($dests$ is empty) return;
    else SPLIT_LOCALLY = TRUE;
    end if
  end if
  for $i = 1$ to $4$ do                                   /* partition the destination list */
    begin
     if ($dests$ is not empty and port $O_i$ connects node $current$ to its child node $C_i$)
     then
       if ($node[dests.last()].PN \leq node[C_i].high$)
         create integer list $dests\_C_i$ = $dests$ and clear list $dests$;
       else              /* partition list dests and keep new list $dests\_C_i$ sorted */
         create empty integer list $dests\_C_i$;
         while ($dests$ is not empty and
             $node[dests.first()].PN \leq node[C_i].high$)
             move $dests.first()$ from list $dests$ to the end of list $dests\_C_i$;
       end if
      end if
    end if
  if ( not SPLIT_LOCALLY) and (exist more than one non-empty $dests\_C_i$) then
    add header flit [$current$, “0”, “0”, “0”, “0”, “0”] to the end of the header;
    SPLIT_LOCALLY = TRUE;                                 /* split at current node */
    end if
    if ( SPLIT_LOCALLY)
        set $O_i = 1$ for every non-empty $dests\_C_i$ in last added header flit;
    for $i = 1$ to $4$ do
        if (exist non-empty $dests\_C_i$) call $Header\_Calculation$ ($C_i$, $dests\_C_i$);
end.
```

Now we consider the complexity of our header encoding scheme for a multicast with D destinations in a network with N nodes. During network configuration, two $O(N)$ operations are performed as pre-processing steps for all subsequent communication in the network. The two steps correspond to constructing a binary spanning tree and traversing the tree in post order. When a multicast message is present at a source node, the destination list is first sorted according to PN numbers which takes $O(D \log D)$ time. The recursive procedure in Figure0 is then executed to compute the header flits. With a complete binary tree and randomly-distributed destinations, this algorithm has a complexity of $O(D \log N)$ on average, with at most D computations in each level and a total of $\lceil \log(N+1) \rceil$ levels in the tree. As a special case of multicast, a broadcast message in a network with a complete binary spanning tree takes $O(N \log N)$ time for its header encoding.

Note that different tree constructions vary the number of levels of the tree while the distribution of destinations affects the number of computations in each level. For example, the DSTM construction introduced in Section 4 produces two binary trees with $2\sqrt{N} - 1$ levels instead of $\log N$ levels in a complete binary tree. Two distributions of the destination nodes simplify the algorithm of Figure 0. First, any computation with a destination list covered by a single child's subtree has a complexity of $O(1)$ at the current node. Second, any computation with a single destination has a total complexity of $O(1)$, either for a single destination branch or a unicast. Therefore, unicast has a header encoding complexity of $O(1)$. In Section 5, we analyze the complexity of the header encoding scheme for the trees produced by our spanning tree construction.

As for the storage requirement of this header encoding method, the tree information stored at each node takes size of $O(N)$. The procedure of Figure 0 requires a storage of $O(D)$. All these requirements may be met using local memory at the source nodes. No on-chip memory is required in the routers.

APPENDIX B

Proof of Theorem 5.1

The proof is done by induction.

First consider 5×5 torus as shown in Figure 1, i.e. $\sqrt{N} = 5$, nodes N_{11} and N_{33} are the starting nodes for T_1 and T_2 respectively. The diameter $D = 2\sqrt{N} - 1 = 9$ can be easily proved by exhaustive counting.

Second, assuming diameter $D_i = 2\sqrt{N_i} - 1$ for $\sqrt{N_i} \times \sqrt{N_i}$ torus network for odd number $\sqrt{N_i} = 2i + 1$ (i is an integer), we prove that, diameter $D_{i+1} = 2\sqrt{N_{i+1}} - 1$ for $\sqrt{N_{i+1}} \times \sqrt{N_{i+1}}$ torus network, where $\sqrt{N_{i+1}} = 2i + 3$, i.e., the diameter increases by 4 with each increase of 2 in network size.

To bring the result from $\sqrt{N_i} \times \sqrt{N_i}$ torus network with $\sqrt{N_i} = 2i + 1$ to an expanded network, the old topology should be held in the expanded network, which leads to the old trees also be held. The expansion can be done by adding two rows and two columns in the manner shown in Figure 2.

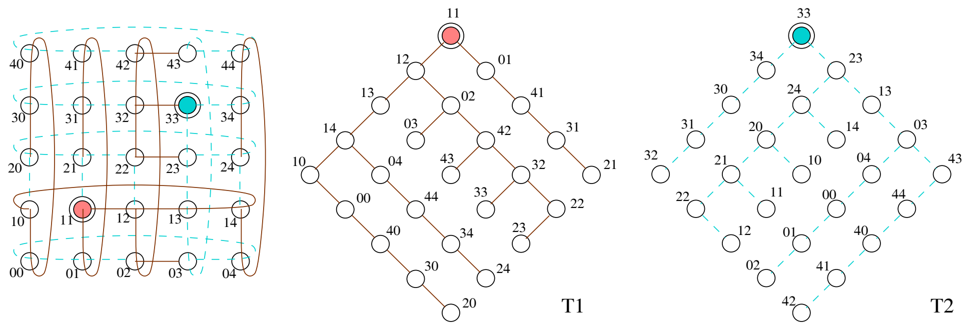


FIG. 1. 5x5 torus network and its corresponding spanning trees.

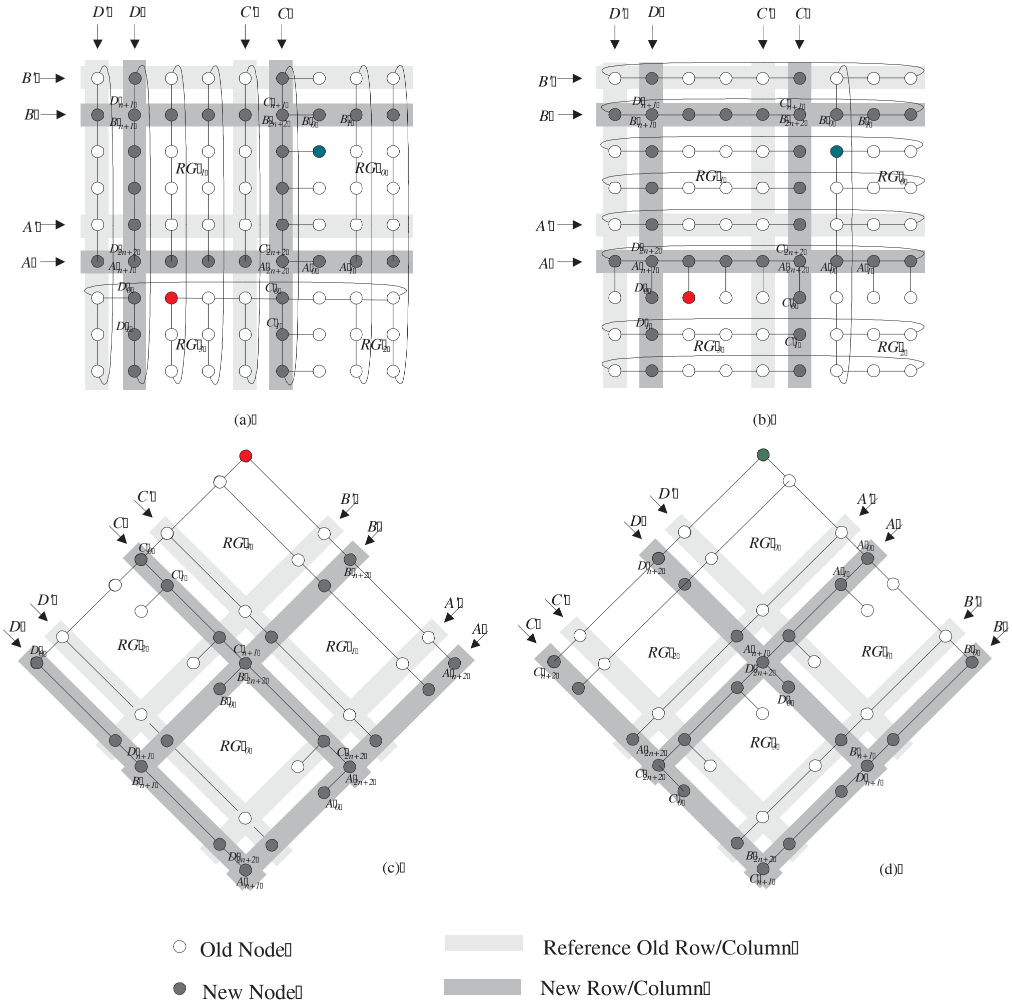


FIG. 2. Expansion of $\sqrt{N_i} \times \sqrt{N_i}$ torus network to $\sqrt{N_{i+1}} \times \sqrt{N_{i+1}}$ network and the corresponding spanning trees.

A $\sqrt{N_{i+1}} \times \sqrt{N_{i+1}}$ torus network with $\sqrt{N_{i+1}} = 2i + 3$ can be expanded from a $\sqrt{N_i} \times \sqrt{N_i}$ torus network where $\sqrt{N_i} = 2i + 1$ by adding two rows and two columns.

In our proof, row A and row B are added adjacent to R_1 and R_2 in the $+y$ direction respectively. Column C and column D are added adjacent to R_1 and R_2 in the $-x$ direction respectively. The added nodes are referred to as *new* nodes. Nodes that were present in the original network are referred to as *old* nodes. The expanded trees are shown in Figure 2(c) and (d), from where we can see the expansion is done on the old topology by inserting four dark shaded regions A, B, C and D. We evaluate the distance between nodes in the expanded network for four different cases.

Case 1. Distance between old nodes and old nodes.

Starting from any old node, we need to traverse at most 3 dark shaded regions to reach any old nodes. This observation holds for both trees. Therefore, at most 3 hops are added between any two old nodes. The increase of the distance is no more than 3.

Case 2. Distance between new nodes and old nodes.

From Figure 2(c) and (d), we select a reference old node in light shaded region A', B', C', D' for each new node in dark shaded region A, B, C, D respectively. The reference nodes are geographically adjacent to the new nodes along either x or y direction. Every new node except four intersection nodes is able to find a reference old node and that the new node is either one hop away from that old node or able to reach their least common ancestor old node by one more hop. With this one extra hop and the result from Case 1, every new node except four intersection nodes are able to reach any old nodes with at most 4 more hops.

Case 3. Distance between four intersection nodes and old nodes.

New nodes located at the intersection of the four dark shaded region adjacent to new nodes in both x and y directions. Therefore no reference old node can be found. However, node C_{n+1} at the intersection of row B and column C belongs to CRNS of T_1. Node A_{n+1} at the intersection of row A and column D belongs to CRNS of T_2. Therefore, they are able to reach all other nodes within $2\sqrt{N_{i+1}} - 1$ hops. For each of the rest two intersection nodes, i.e. node B_{n+1} at intersection of row B and column D and node A_{2n+2} at intersection of row A and column C, we obtained a pair of formulas for the distances between the intersection node and old nodes, one for each tree. These pairs of formulas showed a compensation relationship. When the path in one tree is quite long, the other tree gives a shorter path. These formulas show that the distance is no greater than $2\sqrt{N_{i+1}} - 1$ in all cases.

Case 4. Distance between new nodes and new nodes.

In a way similar to Case 3, we also prove the diameter condition is satisfied.

Therefore, that the combined diameter for DSTM–1 is no more than $2\sqrt{N} - 1$ is proved for odd number $\sqrt{N} = 2i + 1$. For even number $\sqrt{N} = 2i$, combined diameter for the base case of $\sqrt{N} = 4$ can be easily proved to be 7 by exhaustive counting as we did in step one. With the same expansion and induction, same result can be proved for even number $\sqrt{N} = 2i$. Therefore, the combined diameter of DSTM–1 is no more than $2\sqrt{N} - 1$ is proved.

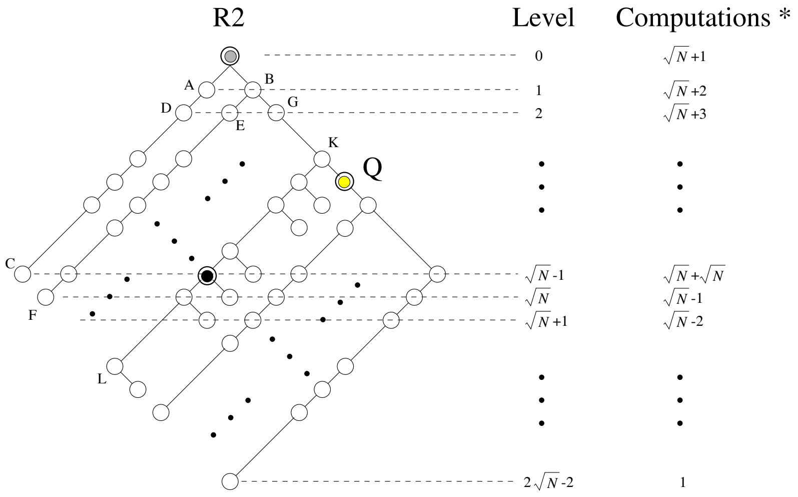
APPENDIX C

Proof of Theorem 5.3

We start the proof of header encoding complexity from the worst case, i.e., broadcast in a $\sqrt{N} \times \sqrt{N}$ torus. Before the calculation is started from the root node, the complete destination list is first sorted according to nodes' PN numbers, which takes $O(N \log N)$

complexity with N destinations for the broadcast. Algorithm 0 is then executed for the two spanning trees constructed by DSTM.

Figure 1 shows one of the edge-disjoint binary spanning trees, i.e. T_2, constructed by DSTM–1 in a $\sqrt{N} \times \sqrt{N}$ torus. We prove the broadcast header encoding complexity is of $O(N)$ with this tree using Algorithm 0. As similar analysis applies to tree T_1 of DSTM–1 and all other trees produced by DSTM construction, we prove the broadcast header encoding complexity for DSTM is of $O(N)$.



* Not include the computations at nodes on branch KL.

FIG. 1. Analysis of header encoding complexity of DSTM broadcast.

Figure 1 shows T_2 as a binary spanning tree with a total number of $2\sqrt{N} - 1$ levels. The number of computations in each level is listed to the right of each level. Being a broadcast, destinations list includes all nodes in the network, sorted according to their PN numbers. At level 0, root node R_2 takes one computation to strip off itself from the destination list, and $\sqrt{N} - 1$ computations to move the $\sqrt{N} - 1$ destinations on the branch AC to child A's destination list. After this, since the rest of the destinations can be completely covered by child B's subtree, one computation is taken to assign the destination list to child B. Therefore, the number of computations at level 0 is $\sqrt{N} + 1$, as listed to the right of level 0.

At level 1, since branch R_2C is a linear array, node A takes one computation to strip off itself from A's destination list. The rest of the list is assigned to child D. While node B repeats the computation of node R_2, $\sqrt{N} + 1$ computations are taken to partition B's destination list. A total number of $\sqrt{N} + 2$ computations is therefore needed at this level.

At level 2, there are two linear arry branches DC and EF and a splitting node G. A total number of $\sqrt{N} + 3$ is needed for nodes at this level, with 1 computation for each linear array and $\sqrt{N} + 1$ computations for the splitting node.

The number of computations in each level forms a equal-difference series until level $\sqrt{N} - 1$ with a computation number of $\sqrt{N} + \sqrt{N}$.

Starting from level $\sqrt{N}$, all branches are linear arrays and the number of computations in each level forms a equal-difference series from $\sqrt{N} - 1$, $\sqrt{N} - 2$ to 1 at the lowest level.

Except for the above computations, nodes on branch KL have additional computations because of their non-linear array subtrees. The computation numbers are respectively $2\sqrt{N}$, $2\sqrt{N}-2$, $2\sqrt{N}-4$, $\cdots$, 4, 2 for nodes on each level of branch KL.

The overall number of computations then equals to

$$\begin{aligned}\text{Computations} = &\underbrace{(\sqrt{N}+1)+(\sqrt{N}+2)+\cdots+(\sqrt{N}+\sqrt{N})}_{\sqrt{N}}\\ &+\underbrace{(\sqrt{N}-1)+(\sqrt{N}-2)+\cdots+2+1}_{\sqrt{N}-1}\\ &+\underbrace{(2\sqrt{N}-2)+(2\sqrt{N}-4)+\cdots+4+2}_{\sqrt{N}}\\ = &\,3N-\sqrt{N}.\end{aligned}$$

Therefore, broadcast header encoding complexity of DSTM–1 with T_2 is of $O(N\log N+N)$.

With regular topology, we know that T_1 of DSTM–1 has lighter right branches. Header encoding function described in Algorithm 0 can then be modified to search the destination list from the end of the list instead of the front. Same broadcast header encoding complexity can then be achieved onT_1, completely symmetric to T_2. Therefore, the broadcast header encoding complexity of DSTM–1 is of $O(N\log N+N)$.

The differences among DSTM trees don't bias the argument on the complexity analysis of DSTM–1. Therefore, broadcast header encoding complexity of DSTM is of $O(N\log N+N)$.

Generally, when a multicast message is present, the complete destination list is first sorted according to nodes' PN numbers. This is a $O(D\log D)$ procedure given D destinations. The execution of Algorithm 0 then takes no more than $O(N)$ complexity for the same reason as in broadcast with $D\leq N$. The fact that the multicast header encoding complexity with D destinations of DSTM is of $O(D\log D+N)$ is then proved.

REFERENCES

1. B. Alspach, J. Bermond and D. Sotteau, "Decomposition into cycles I: Hamiltonian decomposition," in Hahn, Sabidussi, and Woodrow, editors, *Cycles and Rays,* vol. 301 of *NATO Science Series: C Math. and Phys. Sci.,* pp. 9–18, 1989.
2. J. Aubert, "Decomposition de la somme Cartesienne d'un cycle et de l'union de deus cycles Hamiltoniens en cycles Hamiltonians," *Disc. Math.,* vol. 38, pp. 7–16, 1982.
3. B. Barden, R. Libeskind-Hadas, J. Davis and W. Williams, "On Edge-Disjoint Spanning Trees in Hypercubes," *Inf. Proc. Lett.*, vol. 70, pp. 13–16, Apr. 16 1999.
4. J. Bruck, "On Optimal Broadcasting in Faulty Hypercubes," *Disc. Appl. Math.,* vol. 53, pp. 3–13, 1994.
5. C.M. Chiang and L. M. Ni, "Multi-Address Encoding for Multicast," *Proc. of the Parallel Computer Routing and Communication Workshop,* (PCRCW94), May 1994.
6. C.M. Chiang and L. M. Ni, "Encoding and Decoding of Address Information in Multicast Message," *Proc. of the 1994 Int'l Computer Symposium Conference,* pp. 1092–1097.
7. M. Gerla, P. Palnati and S. Walton, "Multicasting protocols for high-speed, wormhole-routing local area networks," *Computer Communication Review,* vol. 26, pp. 184–193, Oct. 1996.
8. I. Havel, "On Hamiltonian Circuits and Spanning Trees of Hypercubes," *Casopis Pro Pestovani Metematiky,* vol. 109, pp. 135–152, 1984.
9. S. Johnsson and C.-T. Ho, "Optimum Broadcasting and Personalized Communication in Hypercubes," *IEEE Trans. Comp.,* vol. 38, pp. 1249–1268, Sept. 1989.
10. R. Libeskind-Hadas, K. Watkins, and T. Hehre, "Fault-Tolerant Multicast Routing in the Mesh with No Virtual Channels," *Proc. of HPCA '96,* pp. 180–190.

11. R. Libeskind-Hadas, D. Mazzoni and R. Rajagopalan, "Tree-Based Multicasting in Wormhole-Routed Irregular Topologies," *Proc. of IPDPS '98,* pp. 244–249.

12. R. Libeskind-Hadas, D. Mazzoni and R. Rajagopalan, "Tree-Based Multicasting in Wormhole-Routed Irregular Topologies," Technical Report HMC-CS-97-02, Harvey Mudd College, Aug. 1997.

13. X. Lin, P. McKinley and L.M. Ni, "Deadlock-Free Multicast Wormhole Routing in 2D-Mesh Multicomputers," *IEEE Trans. on Parallel and Distributed Systems,* vol. 5, pp. 793–804, Aug. 1994

14. L.M. Ni, "Should Scalable Parallel Computers Support Efficient Hardware Multicast?", *Proc. of the 1995 ICPP Workshop on Challenges for Parallel Processing,* pp. 2–7, 1995.

15. D.K. Panda, S. Singal, and P. Prabhakaran, "Multidestination Message Passing Mechanism Conforming to Base Wormhole Routing Scheme," *Proc. of the Parallel Computer Routing and Communication Workshop,* (PCRCW'97), pp. 131–145, 1994.

16. J. Roskind and R. Tarjan, "A Note on Finding Minimum-Cost Edge-Disjoint Spanning Trees," *Math. of Oper. Res.*, vol. 10, pp. 701–708, Nov. 1985.

17. R. Sivaram, D. K. Panda and C. B. Stunkel, "Multicasting in Irregular Networks with Cut-Through Switches using Tree-Based Multidestination Worms," *PCRCW'97,* pp. 35–48, June 1997.

18. S. Song, "Towards a simple construction for Hamiltonian decomposition of the Hypercube," *Disc. Math. and Theoret. Comp. Sci.,* vol. 21, pp. 297–306, 1995.

19. C. Stunkel, R. Sivaram, and D.K. Panda, "Implementing Multidestination Worms in Switch-Based Parallel Systems: Architectural Alternatives and their Impact," *Proc. 24th Annual Int'l Symp. on Computer Arch.,* pp. 50–61, 1997.

20. Y.-C. Tseng, D.K. Panda, and T.-H. Lai, "A Trip-Based Multicasting Model in Wormhole-Routed Networks with Virtual Channels," *IEEE Trans. on Parallel and Distributed Systems,* vol. 7, pp. 138–150, Feb. 1996.

21. H. Wang and D. M. Blough, "Tree-Based Multicast in Wormhole-Routed Torus Networks," *Proc. 1998 Int'l Conf. on Par. and Dist. Proc. Tech. and Appl.*, pp. 702–709.

22. H. Wang and D. M. Blough, "Construction of Edge-Disjoint Spanning Trees in the Torus and Application to Multicast in Wormhole-Routed Networks," *Proc. 1999 Int'l Conf. on Parallel and Distributed Computing Systems,* pp. 178–184, 1999.

23. J.S. Yang and C.T. King, "Efficient Tree-Based Multicast in Wormhole-Routed 2D Meshes," Proc. 3rd Int. Symp. on Parallel Architectures, Algorithms and Networks, pp.494–500, 1997.