*In Proceedings of the ISCA 13th International Conference on Parallel and Distributed Computing Systems*

# Region Broadcasting in *k*-ary *m*-way Networks

Muhammed Mudawwar and Rania Mameesh
Computer Science Department
The American University in Cairo

mudawwar@aucegypt.edu
rmameesh@cs.aucegypt.edu

**Abstract**

*k*-ary *m*-way networks are especially suitable for broadcasting because of the shared nature of *m*-way channels. A flit placed on an *m*-way channel can be accepted concurrently by adjacent routers and node interfaces during a given clock cycle. This paper discusses two modified Minimum Spanning Tree (MST) broadcast algorithms that are designed to work efficiently in *k*-ary *m*-way networks. The algorithms are the MST-safe and MST-replica. The two tree-based algorithms are deadlock free and can be safely mixed with dimension-order routing for unicast messages. The performance of *k*-ary *m*-way mesh and hypercube networks is evaluated using the MST-Safe and MST-Replica algorithms and compared with existing unicast-based broadcast algorithms, which are Recursive Doubling and MST-unicast. The results clearly favor the MST-safe and MST-replica over the unicast-based broadcast algorithms.

**Keywords:** Region broadcasting, *k*-ary *m*-way networks, MST-safe, MST-replica, MST-unicast, Recursive Doubling.

## 1 Introduction

The communication operations among nodes of an interconnection network can be either point-to-point or collective, depending on whether exactly two or more than two processes participate in the communication. Collective operations, such as *replication*, are operations that involve the movement of global data [2], [10]. Multicast and broadcast are examples of collective communication.

Perhaps the most essential collective communication routine is broadcast, in which one source sends a message to all the nodes in the network. The broadcast operation can be generalized to allow one source node to send a message to a rectangular region of nodes, rather than to all nodes. This is referred to as *region broadcasting* in this paper. Region broadcasting is a special case of multicasting, in which a source node sends a message to an arbitrary collection of destination nodes. Region broadcasting can be implemented as a collective communication primitive in a *k*-ary *m*-way network quite easily. A large number of parallel applications can benefit from this operation.

Broadcasting can be unicast-based. A source node sends a message to a subset of the destinations. Each node holding a copy of the message forwards it to another subset of the destinations that have not yet received it, and so on until the message is delivered to all destination nodes. An example of a unicast-based broadcast algorithm is the Recursive Doubling algorithm [10], [11].

Broadcasting can be also tree-based. A source message gets replicated at intermediate routers to reach nodes along different routing paths. The same message is delivered to all the nodes on a given routing path. An example of a tree-based broadcast algorithm is the Minimum Spanning Tree (MST) for meshes [2], [3].

This paper extends the work in [5] and [6] by adding a broadcast facility to a *k*-ary *m*-way router. Tree-based broadcast algorithms make use of the shared nature of *m*-way channels because they allow a message to be sent in more than one direction in a given cycle. The MST is a tree-based broadcast algorithm but it is not deadlock free. Two modified MST broadcast algorithms, MST-safe and MST-replica, are discussed in this paper. These two algorithms are deadlock-free in mesh and hypercube topologies and can degenerate to dimension-order routing for unicast messages.

## 2 *k*-ary *m*-way Interconnection Networks

A *k*-ary *m*-way network is a multi-dimensional mesh or torus structure constructed using *m*-way routers and channels [5]. An *m*-way (called also *multiway*) channel is a physical channel *shared* by a maximum number, *m*, of routers or processors. It is the physical wiring of *m* links. An *m*-way router interfaces two *m*-way channels only, irrespective of the network topology or dimension. It has a constant degree 2. An *m*-way router defines the operation of an *m*-way channel. At any clock cycle, only one of the *m* routers (or processors) linked to an *m*-way channel can drive the channel. However, all *m* routers (and processors) can concurrently read the channel. The factor *k* is the number of *m*-way channels along each dimension. In practice different values of *k* can be assigned to different dimensions.

An example of a 4-ary 5-way mesh network is shown in Figure 1. This is a 2-dimensional mesh with 16 nodes and 24 routers. Each 5-way channel wires 4 routers to a node. A channel is identified by a number *c*. A processor node linked to channel *c* is identified as $P_c$. A router connected to channel *c* are identified as routers $R_{c,x}$ when it is along the

positive *X* dimension, and as router $R_{c,y}$ when it is along the positive *Y* dimension. Although one node is shown connected to each channel, it is possible to link multiple nodes without increasing the number of routers.

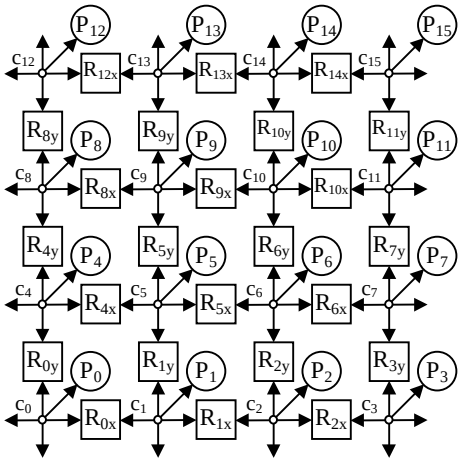


Figure 1: 4-ary 5-way Mesh

### 2.1 Router Structure

A router for *k*-ary *m*-way networks is depicted in Figure 2. A router has two channel interfaces, two channel arbitrators, and two groups of buffers with allocation/ mapping units, routing logic, and buffer arbitrators. The *directionalities* of the two groups of buffers are DIM+ and DIM–, where DIM is the dimensionality of the router. The directionality is used to identify a buffer set when selecting a driver for a channel or when accepting message flits. This identification should be unique across an *m*-way channel.

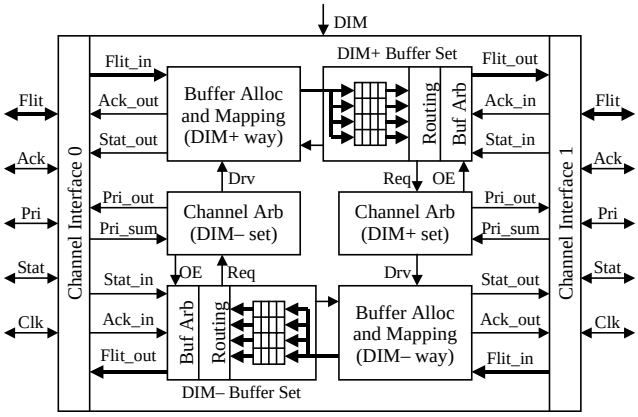


Figure 2: Internal Structure of a Router

A physical channel consists of data, control, and arbitration lines. The *Flit* lines carry one flit of a message. The *Ack* lines are used to acknowledge the transfer of a flit and to report the full status of the receiver buffer. The priority lines, *Pri*, are used for arbitration and carry the wired-OR sum of output priorities of requesting drivers. The *Stat* lines carry the availability and full status of receiver buffers. The *Clk* line is used to synchronize the operation of an *m*-way channel. For more details, see [5].

### 2.2 Message Format

A message consists of a header flit followed by an arbitrary number (possibly zero) of body flits, followed by a tail flit. A *Tag*, identifying the kind of flit, is generated at the sending node and transmitted with each flit. Four tags are used: *Header* (*H*), *Body* (*B*), *Tail* (*T*), and *Invalid* (*I*). This is depicted in Figure 3. A valid tag is a request to transfer a flit across an *m*-way channel. In addition to the tag, the driver's buffer number, *Buf*, is transmitted with every flit. This number identifies the driver's buffer and can change on every router a message reaches. The tag and driver's buffer number are control information sent with every flit.

| H | Buf | Ways | Class | Region | Len | … |
|---|---|---|---|---|---|---|
| B | Buf | Data | | | | |
| B | Buf | Data | | | | |
| B | Buf | . . . | | | | |
| T | Buf | Data | | | | |

Figure 3: Message Format

The header flit carries additional control information. It carries the routing *Ways*, which specifies all the directions of a header flit when it is transmitted across an *m*-way channel. The routing *Ways* is an *m*-bit field, where each bit is associated with one direction. The routing *Ways* is required to transfer header flits, but is not required to transfer body or tail flits. The *Class* field identifies the buffer class or the subset of buffers that can be allocated when a header flit is received. It is used to guide buffer allocation and to avoid deadlocks for some routing algorithms. The *Region* field specifies a rectangular region of processor nodes. It can be encoded absolutely as the addresses of the low and high corner nodes of a rectangular region. Alternatively, it can be encoded relative to a source node as increasing and decreasing offsets along all dimensions. Unicast messages specify regions with one destination node. The length field, *Len*, encodes the number of data bytes in the body and tail flits that follow the header flit. The length field can be used to allocate storage at the destination node as soon as a header flit is received. It can be also used to ensure the correctness of the tags and to establish a limit on the length of a message.

### 2.3 Buffer State

Each buffer in a buffer set has associated state information, as depicted in Figure 4. The allocation bit, *A*, indicates the allocation status. The full bit, *F*, indicates the full status. The driver number, *Drv*, indicates the driver set from which the flits of a message are received. The driver's buffer number, *Buf*, specifies a buffer in a buffers set. *Drv* and *Buf* locate the previous buffer along the routing path. The front pointer, *Fptr*, points to the front entry in a buffer. The rear pointer, *Rptr*, points to the rear entry. The receiver's full status, *RF*, indicates whether the receiver buffer of a message has a full status.

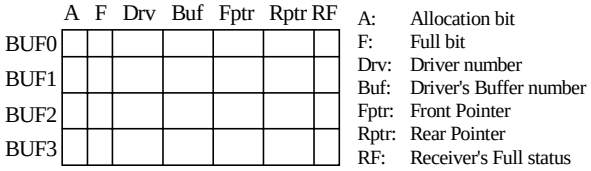


Figure 4: Buffer associated state information

### 2.4 Transferring a Flit

At the beginning of a clock cycle, a driver puts a header flit on an *m*-way channel. The header flit includes the header tag, *H*, the driver's buffer number, *Buf*, and the routing *Ways* that specify the routing directions. All buffer allocation and mapping units across an *m*-way channel examine the header flit. However, some of them will accept the header flit, depending on the routing *Ways*. Once accepted, a free buffer is allocated for the header flit. Therefore, a header flit is transferred to multiple receivers in a single cycle. The receivers concurrently allocate buffers for the header flit. The allocated buffers may have different numbers. However, all of them identify the same driver, *Drv*, and the same driver's buffer number, *Buf*.

In some cases, not all the receivers in the specified routing *Ways* may have free buffers. Some receivers may allocate a buffer. Others will not. An acknowledgement, *Ack_out*, indicates whether the allocation is successful or not. The wired-AND acknowledgement, *Ack*, of all *Ack_out*, is put on the shared channel. A header flit is not transferred until all the required buffers are allocated. Therefore, a driver may place a header flit on a channel more than once, until all the required buffers are allocated At that point, the header flit is transferred to all the receiver buffers.

When a driver places a body or a tail flit on a channel (*Tag* = *B* or *T*), it does not include the routing *Ways* as part of the flit. All allocation and mapping units across a channel examine the flit that carries the driver's buffer number, *Buf*. They also receive the current driver number, *Drv*, from the channel arbitrator. All allocation and mapping units are searched in parallel by content for a match with *Drv* and *Buf*. If a match occurs and the allocation bit is set, the corresponding buffer allocation and mapping unit will accept the body or tail flit. Otherwise, it will reject it.

## 3 Broadcast Algorithms

The broadcast algorithms, described in this section, support broadcasting not only to the whole network, but also to a rectangular region of the network as defined in the message header. The source node is assumed to be part of the region. Otherwise, a message will have to be sent first to any node in the region, and then a second message will be broadcast to all the nodes in the region.

### 3.1 The Minimum Spanning Tree (MST) Algorithm

The MST is a tree-based broadcast algorithm in which a router in dimension *i* is allowed to send a message to all routers in dimensions ≥ *i*. Figure 5 shows the MST broadcast algorithm in a 4x4 region. Circles denote processor nodes, small squares denote channels, and arcs denote buffers. The black circle is the source node and the white ones are the destination nodes. Buffers exist in routers as well as in node interfaces. A source node injects a header flit to be received concurrently in buffers along four directions (X+, X–, Y+, Y–). The four routing directions are encoded as bits in *Ways* in the header flit. The *Region* field in the header flit initially encodes the 4x4 region depicted in Figure 5. The *Region* field is then modified at each receiver router to reflect the sub-region that a message should reach along its direction. For example, when an X+ buffer receives a header flit, it can route it to the next buffers along the X+, Y+, Y–, and node ejection buffer. However, it cannot route it along the X– direction. If a Y+ buffer receives a header flit, it can route it along the Y+ direction and to a node ejection buffer, but not along the X+, X–, or Y– directions. A body or a tail flit is transferred when all the specified routing directions are ready to receive it.

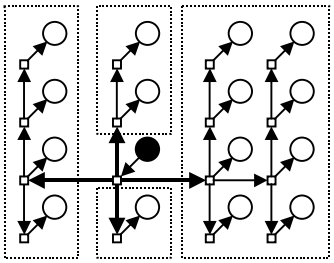

Figure 5: MST Broadcasting in a 4x4 Region

### 3.2 Deadlock Formation in MST

Many deadlock situations occur when the MST algorithm is used. These deadlock situations can be obtained by recording the deadlock state of a network under simulation. Detailed descriptions of these deadlock situations are recorded in [4]. The causes of these deadlocks can be summarized as follows:

**a. Reserving buffer resources is not ordered**. This happens when a header flit is to be transferred to buffers along multiple routing ways, say X+, Y+, Y–, and the local node, and that no buffer in the X+ routing way is available, but they are available in the other routing directions. If buffers are NOT allocated (or reserved) in an orderly fashion, say Y+ is allocated before X+, then this was found to be a cause for deadlock formation. Therefore, buffer allocation across an *m*-way channel must be ordered.

**b. Advancing a header flit in some allocated buffers**. This happens when a header flit requires the allocation of buffers along multiple routing ways, but some of these routing ways do not have available buffers. If a header flit is allowed to advance from a driver buffer to a receiver buffer along some routing ways while keeping a copy in the driver's buffer to be transferred to the other ways later, then this was found to be a cause of deadlock formation. Therefore, a header flit must not advance until the buffers in all the routing ways are allocated.

**c. Buffer size is less than the message length**. In other words, the wormhole routing technique is also believed to be a cause of deadlocks for an MST broadcast algorithm. If one branch of an MST tree becomes blocked, then all other branches will become blocked as well. Therefore, the size of buffers must be big enough to store a whole message or packet [2], [3]. In other words virtual cut-through must be used rather than wormhole routing. This last statement, however, requires further investigation.

### 3.3 The MST-Safe Algorithm

The MST-Safe algorithm is a modified version of MST algorithm. Safety here means that the algorithm prevents deadlocks by satisfying the following 3 conditions:

**a. Buffer reservation is ordered.** If buffers along all routing ways are available then they can be allocated concurrently. However, if not all buffers are available then they should be reserved in an orderly fashion: X+ then X– then Y+ then Y– … then an ejection buffer of a local node. More than one buffer can be reserved concurrently as long as the order of reservation is not violated.

**b. A header flit is advanced when the buffers along all routing ways are allocated**. If the buffers along all routing ways are available then they can be allocated concurrently and the header flit can be advanced to all of them in one cycle. Otherwise, some of the buffers may be reserved according to condition *a*, but the header flit should not advance.

**c. Virtual cut-through rather than wormhole routing**. This means that buffers are large enough to hold a message or a packet.

### 3.4 The MST-Replica Algorithm

This is another modified version of the MST algorithm. It satisfies conditions *a* and *c* of the MST-safe algorithm. However, it provides more flexibility for condition *b*. If a header flit is to be sent along multiple routing ways, which are not all available, then the header flit is allowed to proceed in some of the routing ways, and a replica message is produced to allow the message to proceed along the remaining routing ways.

An example that illustrates the MST-replica algorithm is depicted in Figure 6. In this example, a header flit in buffer *buf* is to be sent to all the nodes in regions 1 and 2. The routing ways are X+, Y+, Y–, and the local node. If a buffer is available along the X+, Y–, and local node directions, but all buffers are allocated along the Y+ direction then the message header in *buf* is advanced only to the X+ direction to be broadcast to region 2, while a replica message is produced for region 1. Advancing the header along the X+ direction only, but not along the Y– direction, will preserve the ordering of buffer allocation. Since Y+ buffers are not available (dashed arrow in Figure 6), we cannot allocate buffers along the Y– direction or in the node ejection buffer. The replica message will be transmitted independently to region 1 when a Y+ buffer becomes available. Message replication requires hardware support and can be implemented internally to a router by allocating a new buffer, replicating the message flits, and splitting a region into two regions in the header flits of the original and replicated message. Alternatively, message replication can be implemented at a node interface, but again with special hardware support. Message replication increases the network traffic.

The worst case of MST-replica is when a replica message is created for each routing direction. In this case, MST-replica degenerates to an MST-unicast algorithm.

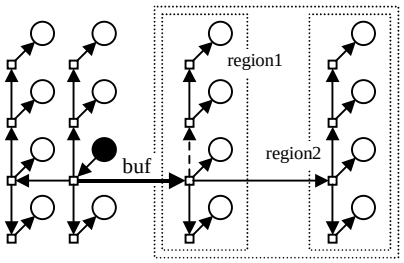


Figure 6: MST Replica in a 4x4 Region

### 3.5 The MST-Unicast Algorithm

Broadcasting can also be implemented using multiple unicast messages to all the destinations. This is called unicast-based broadcasting. The advantage of this approach is that it requires no special support from the hardware. The disadvantage is that it produces more traffic.

The MST-Unicast algorithm is a unicast-based broadcast algorithm that works almost the same as MST. Instead of putting a message flit once on a channel and the message flit is transferred to all routing directions in a single cycle, copies of the same message are generated at a node and each copy is transferred at different cycles to different directions. A message is replicated at each intermediate node and forwarded to the nodes in the remaining sub-regions. Dimension-order routing is used to route messages. Therefore the algorithm is deadlock free in a mesh structure.

### 3.6 Recursive Doubling (RD) Algorithm

The RD algorithm makes each node holding a copy of a message responsible for a partition of a row or column. The node divides its partition in half and sends a copy of a message to the node in the other half that occupies the same relative position [10]. Figure 7 shows the RD algorithm in an 8x8 region. When a message is first generated, the source produces six different messages each to a different destination and with a different region. Each of these messages will be delivered to their destination using XY-deterministic routing. After receiving a message that requires forwarding, a node copies it and injects the copy back into the network. When each of the new messages reaches its destination, new copies are produced and sent to other destinations until the whole region receives the message.

As with other unicast-based broadcast algorithms, the number of generated and injected messages in the RD algorithm is equal to the number of nodes in a region. Another disadvantage is that the generated messages might pass by other nodes that have not received the message yet on their way to their destinations. However, the message is not delivered to these other nodes until later. The RD algorithm is deadlock free since it uses dimension-order routing to send messages.

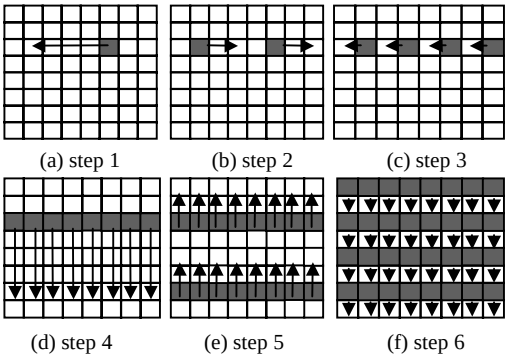


Figure 7: RD Broadcast in an 8x8 Mesh [10].

## 4 Network Simulation and Performance

To measure the performance of broadcast algorithms implemented in *k*-ary *m*-way networks, a number of mesh and hypercube networks have been simulated varying few parameters in every run. The simulator is a C++ program that simulates *k*-ary *m*-way networks at the flit level. A flit transfer over an *m*-way channel is assumed to take place in one cycle. The simulator can be configured to support different parameters and can generate various statistics.

The latency incurred by a message is measured from the time of generation at a source node until its tail is ejected at a destination node. Latency is measured from the generation time to include the time incurred by the message at the source queue. Traffic is measured as the percentage of utilization of channels. A channel is utilized during a clock cycle if it is used to transfer a flit successfully. The injection rate of a node is the percentage of channel cycles used to inject a flit successfully into the network. The ejection rate is the percentage of channel cycles used to eject a flit successfully from the network. The average traffic, injection, and ejection rates are taken over all channels and nodes in the network over a period of time.

### 4.1 Effect of the Broadcast Algorithm

The purpose of this experiment is to show the performance of the tree-based algorithms that take advantage of the shared nature of *m*-way channels versus broadcast algorithms that are unicast-based. The tree-based algorithms are MST-Safe and MST-Replica. The unicast-based algorithms are MST-unicast and RD. Two medium-sized networks are simulated, an 8x8x8 mesh and a 9D hypercube. The number of buffers in each buffer set along each routing direction is four. All messages carry 64 bytes of data and consist of 5 flits (1 header + 4 data). Each flit is assumed to carry 16 bytes of data [5].

The performance of a 3D mesh is shown in Figures 8 and 9. Figure 8 shows the average latency and ejection rate, which is a measure of throughput, and Figure 9 shows the network traffic. Figures 10 and 11 are for a 9D hypercube. The figures reveal that the tree-based broadcast algorithms, MST Safe or MST Replica, clearly outperform the unicast-based ones, MST Unicast and RD. The latency of a tree-based algorithm is much lower than the latency of a unicast-based broadcast algorithm.

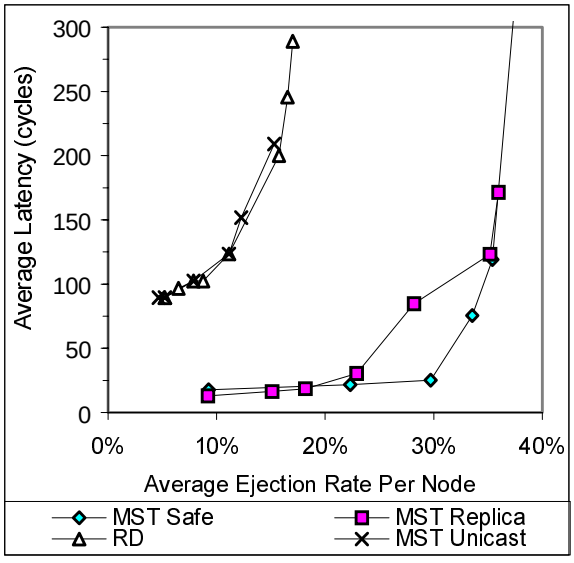


Figure 8 : Latency and Ejection Rate in an 8x8x8 Mesh

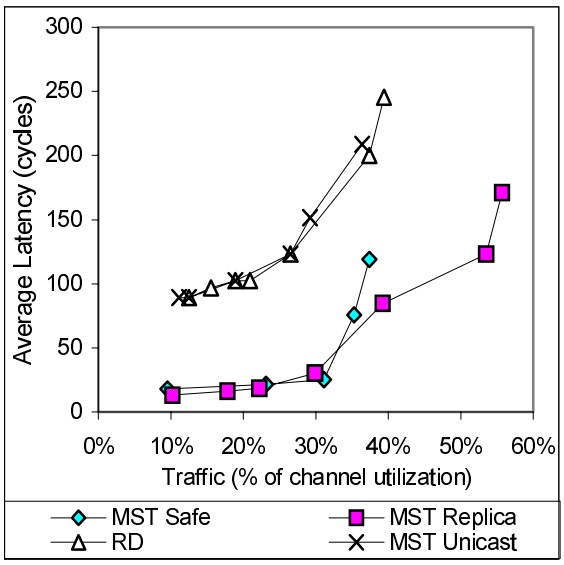


Figure 9: Latency and Traffic in an 8x8x8 Mesh

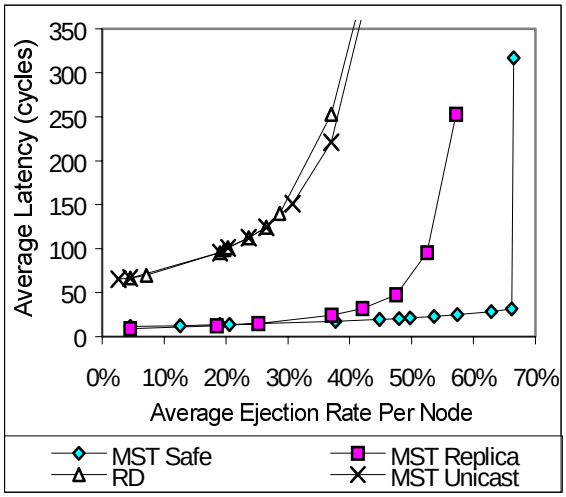


Figure 10: Latency and Ejection Rate in a 9D Hypercube

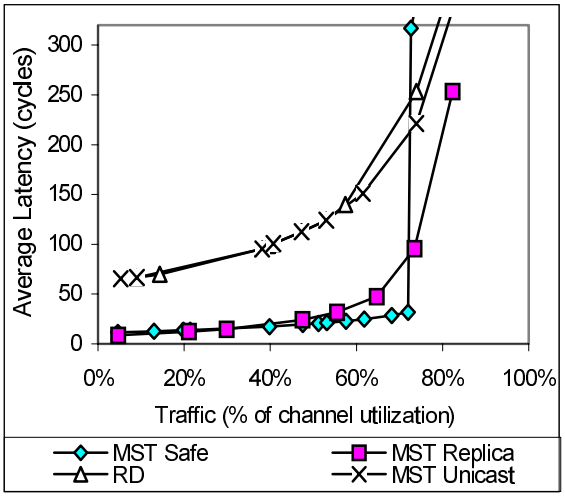


Figure 11: Latency and Traffic in 9D Hypercube

Figures 8 through 11 also reveal that although the network traffic for a tree-based broadcast algorithm is comparable to that of a unicast-based broadcast algorithm, the throughput of a tree-based algorithm is much better because it saturates at higher ejection rates. The MST unicast and the Recursive Doubling algorithms have almost identical performance. The MST safe algorithm is clearly the best and results in the lowest latencies and highest throughputs at saturation.

### 4.2 Network Performance under a mix of Messages

In this experiment, the performance of 3D mesh network with 16x8x8 channels and nodes is simulated under a different mix of broadcast and unicast messages. The mix is 5% broadcast and 95% unicast in a first run, while it is 95% broadcast and 5% unicast in a second run. The broadcast algorithm used is MST Safe, and the unicast algorithm is dimension-order routing. The number of buffers is 4 along each routing direction. Each message consists of 5 extra wide flits that include 64 bytes of data.

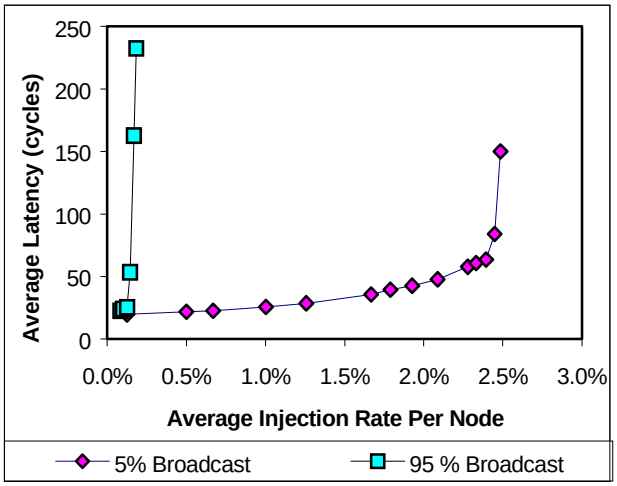


Figure 12: Injection Rate in a 16x8x8 Mesh with a different mix of unicast and broadcast messages

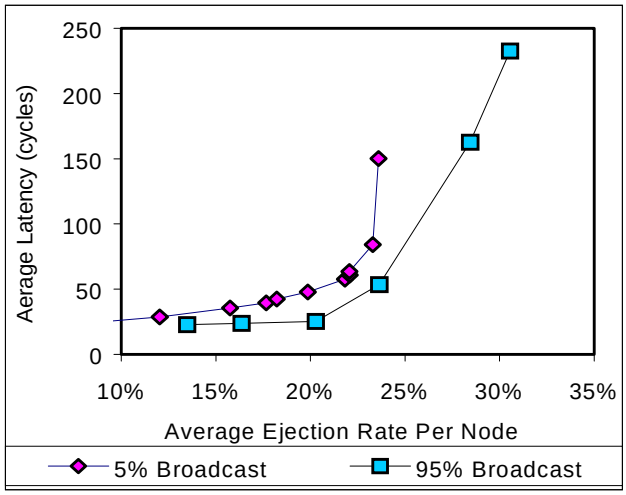


Figure 13: Ejection Rate in a 16x8x8 Mesh with a different mix of unicast and broadcast messages

Figures 12 and 13 show the difference between injection and ejection rates under a different mix of broadcast messages and unicast messages. When the percentage of broadcast messages increases then less messages will be injected but more messages will be ejected. The injection rate is, however, more sensitive to broadcasting than the ejection rate.

## 5 Conclusion and Further Research

Broadcast algorithms that take advantage of the shared nature of an *m*-way channel perform better than those who do not exploit that characteristic. This is why MST Safe and MST Replica perform better than MST unicast and RD. The average latency is much less in MST Safe than in the unicast-based algorithms because messages do not have to be re-injected into the network after being ejected. Although the network traffic is comparable, the average ejection rate is the highest in MST Safe because a channel transaction is used to transfer a flit to multiple directions, rather than to single directions.

The traffic in a mesh network reached only 35% under saturation for the MST Safe algorithm. We are currently trying to improve the traffic to better utilize the shared channels. Improving the traffic will improve the injection and ejection rates. Further research in this direction is to improve the MST Safe algorithm to permit its use in a torus structure, and to relax the requirement of virtual cut-through routing to wormhole routing (Condition *c* in Section 3.3).

## References

[1] W. J. Dally and H. Aoki, Deadlock-Free Adaptive Routing in Multicomputer Networks using Virtual Channels, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 4, pp. 466-475, April 1993.

[2] J. Duato, S. Yalamanchili and L. Ni, Interconnection Networks: An Engineering Approach, *IEEE Computer Society Press*, 1997.

[3] X. Lin and L. M. Ni, Deadlock-Free Multicast Wormhole Routing in Multicomputer Networks, *Proceedings of the 18th International Conference on Computer Architecture*, pp. 116-126, May 1991.

[4] R. Mameesh, Region Broadcasting in Multiway Channel Networks, Master Thesis, Computer Science Department, The American University in Cairo, January 2000.

[5] M. F. Mudawwar, A Switch-Free Router for k-ary m-way Networks, *in Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, June 2000.

[6] M. F. Mudawwar, Multiway Channels in Interconnection Networks, *in Proceedings of the ISCA 12th International Conference on Parallel and Distributed Computing Systems*, pp. 506-513, August 1999.

[7] T. M. Pinkston and S. Warnakulasuriya, On Deadlocks in Interconnection Networks, *Proceedings of the 24th International Symposium on Computer Architecture*, pp. 38-50, June 1997.

[8] L. Schwiebert and D. N. Jayasimha, A Universal Proof Technique for Deadlock-Free Routing in Interconnection Networks, *Proceedings of the Symposium on Parallel Algorithms and Architectures*, pp. 175-185, July 1995.

[9] J.-Y. Tien, C.-T. Ho, and W.-P. Yang, Broadcasting on Incomplete Hypercubes, *IEEE Transactions on Computers*, Vol. 42 No. 11, pp. 1393-1398, November 1993.

[10] Y.-J. Tsai and P. K. Mckinley, An Extended Dominating Node Approach to Broadcast and Global Combine in Multiport Wormhole-Routed Mesh Networks, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, No. 1, pp. 41-57, January 1997.

[11] Y.-J. Tsai and P. K. Mckinley, A Broadcast Algorithm for All-Port Wormhole-Routed Torus Networks, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 7, No. 8, pp. 876-885, August 1996.

[12] J. Wu, Safety Levels-An Efficient Mechanism for Achieving Reliable Broadcasting in Hypercubes, *IEEE Transactions on Computers*, Vol. 44, No. 5, pp. 702-706, May 1995.