Concurrent Asynchronous Broadcast on the MetaNet *

Yoram Ofek, † Bülent Yener, ‡ Moti Yung, §

Abstract

The problem solved in this work is how multiple nodes in a network with an arbitrary topology can broadcast concurrently, in an asynchronous manner, to all other nodes. Asynchronous means that the nodes do not co-ordinate their broadcast, and therefore, it is possible that all nodes will start to broadcast at the same time. Simultaneous broadcast by many nodes can cause traffic congestion, which can result in a traffic loss. The main property of the broadcast algorithms presented in this work is that under any arbitrary broadcast pattern there will be **no packet or cell loss** due to internal traffic congestion.

The routing mechanism used by the broadcast algorithm is a variant of *deflection routing*, which means that a node makes an on-line routing decisions based on the local flow of traffic (i.e., internal load conditions). Unlike other deflection techniques the MetaNet routing is along a *global sense of direction*, which guarantees that packets will reach their destinations. Thus, we call this method *convergence routing* (previous deflection algorithms did not guarantee deterministic routing convergence, i.e., a cell/packet can be deflected indefinitely inside the network). As a result of the convergence property, the deflection routing used in this work is the only one with broadcast capability.

*Early version appeared in *IEEE INFOCOM'91* [20]

†IBM T.J.Watson Research Center, P.O.Box 704, Yorktown Heights, NY 10598, ofek@watson.ibm.com

‡Department of Computer Science, Columbia University, NY NY 10027, yener@cs.columbia.edu

§IBM T.J.Watson Research Center, P.O.Box 704, Yorktown Heights, NY 10598, moti@watson.ibm.com

1 Introduction

In this work we present broadcast and broadcast-with-feedback algorithms on the recently suggested **MetaNet** architecture [19, 18, 21], a LAN with an arbitrary topology. The broadcast and broadcast-with-feedback algorithms are functionally equivalent to the broadcast on today's LANs, say token-ring or Ethernet [3, 16]. The broadcast algorithms are completely loss free under asynchronous (bursty) access method. I.e., a node does not have to coordinate its broadcast with other nodes, and as a result, many nodes may broadcast simultaneously. The time complexity of a typical broadcast on the MetaNet (measured when it is internally not over-loaded) is $O(log\ n)$ - while on a ring based LAN the complexity is n.

The underlying MetaNet is a novel network architecture which can be viewed as a local area network with an arbitrary topology and spatial bandwidth reuse (i.e., packets are removed by their destinations and concurrent transmissions are possible). On this architecture it is possible that on one hand a node can try to transmit asynchronously, without reservation, as much as it can, and on the other hand the network access and flow control will ensure no loss and fair access to the network. The switching over this network requires only a single buffer on the receiving sides of the full-duplex links. The transmission via the receiving buffer can be "cut through", i.e., the incoming packet can be sent to the next link before the entire packet has arrived (unlike "store-and-forward"). All those properties together are found today only on LANs with linear, ring or bus, topologies.

As a general topology LAN/MAN architecture in contrast with traditional linear topologies, the MetaNet has the following obvious advantages: *performance* (adapting to any traffic load requirements by adding links and switches to the topology), *fault-tolerance* (graceful degradation as long as the network is connected, since the control mechanisms (e.g., routing) do not rely on any regular pattern of the topology), and *geographic availability* (the ability to use efficiently the existing link infrastructure).

In a high-speed distributed network environment, the reliance on global knowledge of the traffic pattern in the network may not be possible, since the changes may be too fast. In distributed data network design, such as in a LAN, one may assume the worst-case scenario where at any given time, each user may want to transmit to any other user or even broadcast to all other users. This is, indeed, the common assumption in the LAN design (e.g., IEEE 802 and FDDI). As a result, reservation management which implies the setting of a connection with a given bandwidth to every pair of nodes is not realistic. Since then, for a network of n nodes, we would need $O(n^2)$ connections with only small fraction of the bandwidth assigned to each connection, which would result in an inefficient operation. Thus, one of the main characteristics of a LAN is that it supports connection-less traffic with asynchronous (uncoordinated) access to the network, while guaranteeing fairness (on rings) and loss-free-ness (i.e., that a packet is not lost once it has entered the network). We provide these exact characteristics on the general topology MetaNet.

Furthermore, in LANs, the control functions provided at an intermediate node (e.g., routing decisions) do not cause long delays since they are performed in a fast and relatively simple (hardware-oriented) fashion. Thus, in the MetaNet design, at each node, a special purpose switching component, which we call *network interface*, performs those functions. This is done without the interference of the higher-layers that may be a slower (software) component which we call *end-node*. (See Figure 1 and [25] where a similar node model was suggested.) Special-purpose hardware components are necessary to overcome the gap between transmission speed (in fiber-optics media) and general-purpose computer processing time (in software). The MetaNet provides basic control functions (such as routing, access control, and flow control) using *local state information* in the *network interface*.

The MetaNet architecture can be viewed as built upon the ring-topology local-area MetaRing architecture [5, 4, 17]. The MetaNet extends, modifies, and generalizes the MetaRing properties to a general network with an *arbitrary topology*. Our starting point, the MetaRing, is a full-duplex buffer insertion (single buffer per link) ring network with mechanisms that guarantee fairness (i.e., do not deny network services) and spatial bandwidth reuse (i.e., provide concurrent transmissions in contrast to the traditional token-ring). The MetaRing network was prototyped at the IBM T. J. Watson Research Center in 1989. This prototype supports the transmission of variable size packets at 100 Mb/s link speed with an aggregate throughput of 700 Mb/s. A Gigabit version of the MetaRing had been implemented as part of the IBM participation in the Aurora testbed which was part of the NSF/DARPA Gigabit Networking Program.

Related work:
The routing on the MetaNet is a variant of *deflection routing* [2, 13]. It combines in a dynamic fashion the on-line routing decision with the traffic load inside network (for example, the routing decision may depend on flow-control aspects like whether or not the serial transmission link is idle). Deflection routing typically minimizes intermediate buffering requirements and suffers no loss due to congestion. Unlike previous deflection techniques the MetaNet routing is along a *global sense of direction*, which guarantees that packets will reach their destinations. Thus, we call this method *convergence routing* (previous deflection algorithms did not guarantee deterministic routing convergence, i.e., a cell/packet can be deflected indefinitely inside the network). As a result of the convergence property, the deflection routing used in this work is the only one to have broadcast capability.

Broadcast in store-and-forward packet switch networks with routing tables was studied extensively in the past. Next we summarize a survey work by Dalal and Metcalfe [7]. The simplest broadcast is called *separately addressed packets*, where the packet is duplicated at the source and sent to each destination separately. The second method is *multi-destination addressing* where each packet carries a variable length destination list. An intermediate node gets a destination list, if the destinations should be routed via different outgoing links, the node copies the packet and splits the destination list into smaller lists which it attaches to the copies according to the outgoing links. The next mechanisms in their paper are based on tree structures. One mechanism is *spanning tree forwarding* where broadcasting is done along the tree only. The nice property of this method is that the global structure makes it optimal in messages, it uses the routing tables directly where tree links are marked as special broadcast destinations. To reduce the propagation time the fourth method *source based forwarding* has a BFS spanning tree per each source and messages carry their source which identifies the forwarding tree. The disadvantage is the increase in routing broadcast tables.

The last method surveyed in the paper is *Hot Potato forwarding* which is related to Baran's [2] deflected (hot potato) routing method. A node gets a message and it forwards it to *all* other neighbors, which by itself generates network flooding. To prevent flooding, a few methods are suggested. The first is a hop-count field which causes messages to be discarded once the network diameter is reached, while the second one is sequence number field attached to the message, and each node remembers the numbers of recently forwarded messages for some time interval in which repeated messages are discarded; it may be hard though to estimate for how long a number has to be remembered. The disadvantage in this method is overloading the network with packets, while the advantage is the reliability of this method and its simplicity (e.g., no queue management).

The new method presented in the paper [7] is *reverse path forwarding.* It is based on the routing table mechanism and is source dependent. The node forwards the message to all other neighbors if it believes that the source of the message is on the best path from the source to itself, otherwise the packet is discarded. This algorithm combines hot-potato and source based forwarding.

Deering and Cheriton [8] extend the above broadcast mechanisms and other existing routing mechanisms to internetworking and extended LAN environments. They describe a single spanning-tree method and variants of reverse path forwarding (which they build upon the Bellman-Ford routing algorithm), as well as a method using the new Arpanet routing [22] for shortest path broadcast.

The primitive of *propagation of information with feedback* (PIF) [24] can be viewed as an extension of source based forwarding, or as a flooding which generates the source tree dynamically. The feedback messages are combined from the children and forwarded back to the parent starting from the leaves up to the source. In our broadcast-and-feedback, we do not use flooding, rather, we employ the self-routing mechanism to create dynamically (according to the instant traffic load) a source tree, and feedback messages from the leaves are not combined.

In an interesting study Melen and Turner [15] suggested two basic broadcast methods for high-speed packet switch networks. One based on acquiring explicit permit or token prior to transmission, while the other is based on contention-based arbitration where nodes compete when they observe that it is possible that they may win, and the network decides which node wins via several distributed arbitration mechanisms.

Recently, a general topology LAN called *Autonet* [23] was proposed by DEC System Research Center, the main differences between it and the MetaNet architecture are in the control mechanisms of the two networks. Autonet broadcast is not integrated into the regular traffic, since it can cause deadlocks; so in order to avoid it, the back pressure mechanism is disabled, which requires the use of much larger link buffers (to avoid packet loss). On the MetaNet, the asynchronous broadcast-with-feedback algorithms were integrated with no side-effects (as it is shown in this work).

In contrast with all the above, our broadcast does not need arbitration and is integrated into the fair and loss-free routing/flow-control mechanisms. We do not buffer in intermediate nodes, so we are similar to the classical deflection, however, we do not cause flooding, and we use simple hardware mechanisms to forward the broadcast messages dynamically relying on some global structure (like a spanning tree). Comparing to all existing works, our broadcast-with-feedback is the only one we know of which is sensitive to the current traffic load conditions, and can adapt to them based solely on local information.

2 MetaNet Basic Operations

In this section, we describe some of the basic operation principles of the underlying architecture; more details can be found in [19].

2.1 The Node Structure

As previously said, the routing on the MetaNet is a variant of *deflection routing* [2, 13]. It combines, in a dynamic fashion, the on-line routing decision with the traffic load inside network. Deflection routing typically minimizes intermediate buffering requirements and it suffers no loss due to congestion. The switching principles described in this section are typical to deflection routing networks.

The network topology is assumed to be arbitrary and all links are bidirectional or full-duplex. Each node has one or more full-duplex I/O adapters connected to full-duplex links, as shown in Figure 1. On the receiving side of each link there is a link buffer (LB) which can store one maximum size packet. The switch is part of the network interface and can transfer packets from the link buffers to the host or to the outgoing (output) links, and from the host to the outgoing links.

2.2 Input, Output and Switching

The **host interface**, as shown in Figure 1, is the connection of the **network interface** with the outside world (which is the **end-node**). It contains the output buffers for the traffic to the network and the input buffers for the traffic from the network. As part of the control algorithm the following is done. When the node has a packet to transmit it puts it in the output buffer and decides over which output link the packet should be sent. If the link is idle (i.e., not busy) the packet is sent from the output buffer to the link via the switch. When a packet destined to the node arrives it will be sent to

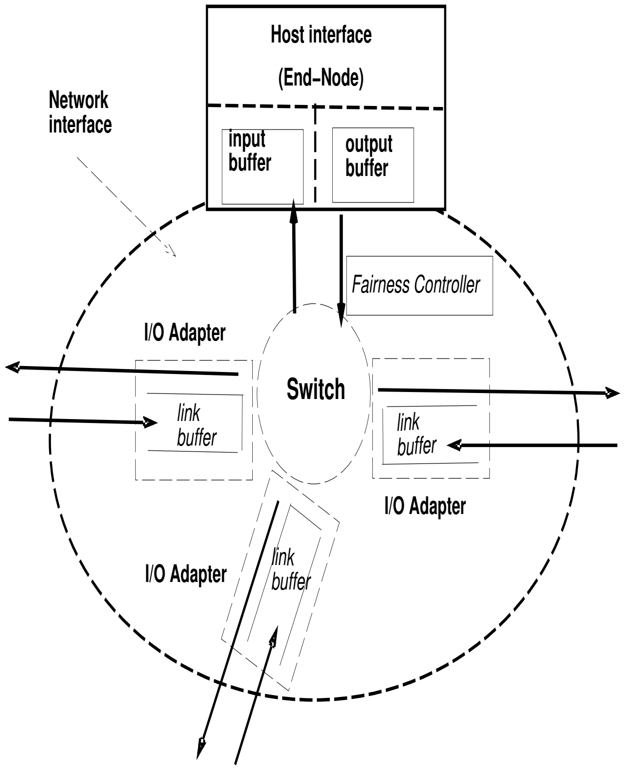

Figure 1: Node Structure and Network Interface

the host interface input buffer via the switch.

The asynchronous switch operation:

- A packet forwarding via the switch can be done only to an output link that is idle.
- Thus, only one packet can be forwarded to an output link, at any given time, either from an input link or from the output buffer.
- A packet that is to be sent via an idle output link, may start to be transmitted before the entire packet has arrived at the input link buffer, i.e., the packets can "cut-through" the switch from input to output [11].
- Multiple simultaneous packet forwarding are possible: (i) from different input links to the host, (ii) from the host to different idle output links, and (iii) from one input link to multiple idle output links for broadcast and multicast operations. These three forwarding properties can be implemented by using a crossbar switch.

2.3 Outcome: Asynchronous Switch Complexity

The MetaNet switch operation has the property that the following three rates are the same: (i) the transfer rate from the input adapter to the switch, (ii) the transfer rate from the switch to the output adapter, and (iii) the serial link transmission rate. In other words, the rate (in bits per second) at which data can be transferred to/from the output/input side of the I/O adapter is the same as the serial transmission rate. We call this property *balanced I/O transmission*, which is also a typical property of other deflection routing designs (e.g., [13]).

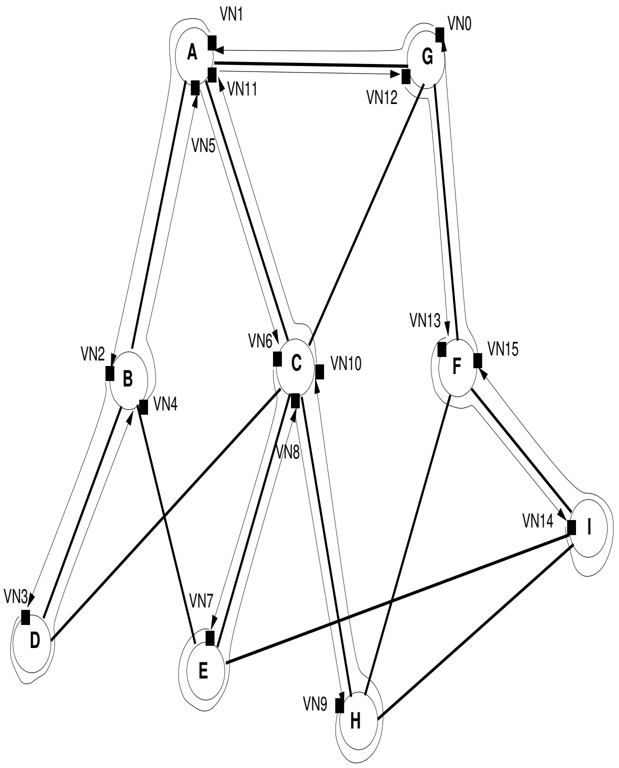

Figure 2: Tree Embedded Ring (TER)

This property significantly simplifies the design and implementation of the I/O adapter. Existing switch designs for fast connection networks (over fixed routes) with output buffers do not have this property (e.g., [1, 10]). In these designs the rate at which data can be transferred via the switch into a single output adapter can be much higher (e.g., 10 times or more) than the serial link transmission rate. As a result, these designs are more complex, especially for high-speed (gigabit/sec) links.

2.4 Embedding of Unidirectional Rings

The links of the network are divided into two types: (i) **ring links** and (ii) **thread links**. The ring links are part of a **virtual embedded ring** and the thread links are all the rest, as shown in Figure 2. One or more virtual embedded rings can be constructed in the network, under the following restrictions:

1. At least one of the virtual ring embedding should be *global*. Namely, the virtual embedded ring should traverse each node at least once, in order to facilitate routing from any node to any other node in the network. (It is possible to combine global and partial ring embeddings, which we do not discuss here.)

2. Each direction of a link can be embedded only once in a virtual ring, i.e., the embedded ring can traverse a ring link only once. This is needed in order to prevent congestion and loss.

Under the above restrictions, on any bidirectional connected network it is possible to have at least one global embedded ring. 2. Furthermore, on a networks with a Hamiltonian cycle, it is possible to embed two global virtual rings in opposite directions.

In this paper we concentrate only on one embedding structure. A traversal ring on a tree that is called tree embedded ring (TER), as shown in Figure 2. The virtual ring links are numbered sequentially from 0 to $m-1$ ($m = 16$, in Figure 2). The starting point can be chosen arbitrarily and the numbers are incremented by one each time a node is traversed. Therefore, a node that is traversed more than once is associated with more than one number. The traverse should be closed, i.e., ring link $m-1$ (ring link 15, in Figure 2) is connected to ring link 0.

We define two types of addresses on this network: physical and virtual. Each node has its own unique identifier (ID) or physical address, denoted by a capital letter A, B, C etc., as shown in Figure 2. The number associated with each ring link constitutes a **virtual node** (VN) or a **virtual address**. Thus a virtual node receives the embedding numbering: VN_0, VN_1,... and so on. A node is associated with one or more virtual nodes, for example in Figure 2, node A has three virtual nodes: VN_1, VN_5 and VN_{11}, and node F has two virtual nodes: VN_{13} and VN_{15}.

We define a **thread link** between two physical nodes, as if it connects two virtual nodes (on two different physical nodes), such that, the distance between them, measured on the virtual embedded ring, is the smallest. Note that under this definition each direction of a thread link may associate with two different virtual nodes. For example, the thread link from C to G, in Figure 2, is connecting virtual nodes 10 with 12, but in the other direction, the thread from G to C is connecting virtual nodes 0 and 6. (In [19] we considered a more general definition of a thread connection.)

This defines the embedding, next let us review how the structure supports some basic network control functions.

2.5 Convergence Routing with Global Sense of Direction

An important property of the MetaNet is the *global sense of direction*, which supports the MetaNet deflection routing. This routing mechanism is called *convergence routing* and it is different from all other deflection techniques as it assures that: (i) a packet reaches its destination, unless a failure occurs, and (ii) a packet can traverse each direction of a link at most once (and the number of hops it takes is bounded deterministically).

Deflection routing involves on-line switching/routing decisions which are based upon the current congestion or flow condition at the switch. As a result, the route from a given source and destination pair is not fixed, but may change dynamically as a function of the load condition at the switches along the route. The routing decision on the MetaNet is based on computing the distance between two virtual nodes on the virtual embedded ring. To find the distance on the ring from virtual node u to virtual node v, the following simple equation is used:

$$DIST(u, v) \ := \ v - u \ (mod \ m) \tag{1}$$

The distance between two nodes is measured with respect to the two virtual nodes that are the closest to one another on the virtual embedded ring (similar to the thread definition). To find the distance on the virtual ring from node U to node V, the following equation is used:

$$DIST(U, V) \ := \ min_{\forall(u \in U, v \in V)} \{v - u \ (mod \ m)\} \tag{2}$$

The distance from node D to node F, in Figure 2, is $10 = 13 - 3(mod\ 16)$ (since it is smaller than $12 = 15 - 3(mod\ 16)$), but the distance from node F to node D is $4 = 3 - 15(mod\ 16)$.

The routing paradigm:

The MetaNet routing paradigm is that *a packet should always be routed towards the destination in the global sense of direction* (by using the virtual embedded ring). In other words, the packet is routed such that its distance from its destination at the next node (using Eq. 1) strictly decreases compared to the current one.

The packet enters the virtual embedded ring via the virtual node that is closest to the destination. The destination field in the packet header contains the virtual node that is closest to the source. Both the source virtual node and the destination virtual node are computed by Eq. 2.

Routing methods:

The simplest method of routing is by following the virtual ring, which guarantees that the packet reaches its destination. This simple method is, of course, not very efficient. Therefore, the routing mechanism at every intermediate node tries to decrease the distance to the destination as much as possible. We use two methods:
(1) **short-cut** to a virtual node on the same node that is closer to the destination in the global sense of direction, and
(2) **jump** on a thread link from a virtual node (on one node) to a virtual node (on a neighbor node) that is closer to the destination.

For example, in Figure 2, a packet arrives at VN_6 and its destination is VN_{14} (node I). If VN_{10} is idle (i.e., the ring link associated with this virtual node is not busy) the packet may short-cut to VN_{10}, and if VN_{12} is idle the packet may jump to it using the thread link from VN_{10} to VN_{12}.

Internal flow control:

In our design the virtual embedded ring operates as a buffer insertion ring [12, 9]. Under the buffer insertion mechanism the traffic already on the ring has priority to continue on it, e.g., the virtual ring traffic into VN_2 has priority over any other traffic to continue on the virtual embedded ring using the ring link from VN_2 to VN_3.

Each time a packet arrives at a virtual node the switch controller tries to find the best available short-cut or jump to a virtual node that brings the packet closest to its destination. But since a packet at a virtual node has priority to continue on the outgoing ring link from this virtual node, lossless-ness is guaranteed if neither a short-cut nor a jump is available.

The objective of the convergence routing on the MetaNet is that the routing from source to destination will be as close as possible to shortest path routing. In order to achieve this the probability of short-cuts and jumps should be maximized, which requires that the network will not be internally over-loaded. One of the fairness algorithms described in [18, 4, 14] can ensure that the network does not get internally over-loaded even when it is externally over-loaded.

2.6 External Access Control

As was previously stated, the virtual embedded ring is a buffer insertion ring. Therefore, following the known insertion buffer access principle, a node can start a packet transmission at any time, from virtual node VN_i, as long as VN_i's link buffer (LB) is empty, i.e., the out-going ring link is idle. If a ring traffic arrives when VN_i is in the middle of a packet transmission, then the ring traffic will be stored in the link buffer (if neither a short-cut nor a jump is available), until this packet transmission is completed. The node cannot transmit anymore until the link buffer becomes idle again, i.e., a non-preemptive priority is given to the ring traffic. If the node is idle, the ring traffic cuts-through the link buffer.

Let us exemplify how a packet enters the network from the external node via the virtual node that is the closest to its destination (using Eq. 2). In Figure 2, a packet that should be sent from node B to node H, enters the virtual embedded ring via VN_4. When the ring link is busy the external packet waits until its designated ring link becomes idle, and this must happen due to the fairness mechanism. Once the packet has entered the virtual ring it travels, with or without short-cuts and jumps, until it reaches its destination and is then removed from the network.

3 The Broadcast Protocol

In this section we present three broadcast algorithms for the MetaNet, one is a simple broadcast, the second is combined with feedback (or echo) to the source for fast acknowledgement, and the third is broadcast with minimum number of messages and time. The following algorithms exploit and preserve the intrinsic properties of the MetaNet. The algorithms are suitable for either partial or global embedding and for a network with or without threads. On partial embedding, the transmission will reach only a subset of the network nodes and can be viewed as a multicast operation.

The algorithm starts with a single broadcast message, that can be duplicated into multiple messages along the way in order to speed-up the operation.

The broadcast should satisfy the following conditions:

1. Each physical node on the embedded ring should be reached at least once.
2. The broadcast message should be copied once into each physical node on this embedded ring, regardless of the number of times it was traversed by this broadcast.
3. Each virtual node on the embedded ring should be traversed at most once.
4. Under light load condition the broadcast should reach all the nodes in time proportional to the propagation delay along the diameter of the network (which can, optimally, be $\log n$).

The first two are necessary for correctness; while the last two are required for performance (reduced time and messages).

3.1 Informal Description

A broadcast packet is marked as such by its source node. The packet header has two basic parameters: its source virtual node identification (VN_{SRC}), and its destination virtual node identification (VN_{DST}), The VN_{SRC} identifies the first ring link this packet is transmitted on, whereas VN_{DST} indicates the point where the packet should be removed from the virtual embedded ring and set initially as $VN_{SRC} - 1(mod\ m)$. Thus, the broadcast packet header format is: $P^{BRD}(VN_{SRC}, VN_{DST})$. The source field in the header remains unchanged as the packet is transmitted over the network, but the destination field is changed when the packet is duplicated.

The default path of the packet is along the embedded ring. However, in order to speed-up the broadcast coverage of the network, the packet is duplicated into multiple packets whenever it is possible, and each duplicate is transmitted concurrently. A packet that arrives at a virtual node can be duplicated in two ways: (i) by a **short-cut** to another virtual node in the same physical node and (ii) by a **jump** via a thread to a virtual node on another physical node. The broadcast algorithm performs three basic functions:
(1) when to receive or remove a packet,
(2) when and how to duplicate a packet, and
(3) how to modify the destination field in the header.

Figure 3 is an illustration of how the algorithm works. Assume that the broadcast packet is sent from virtual node s, and therefore, the initial destination in this case is $s - 1(mod\ m)$, and the initial packet header is $P^{BRD}(s, s-1(mod\ m))$. When this packet arrives to virtual node j while virtual node k on this physical node is idle, the packet is duplicated and will continue on two disjoint segments of this virtual embedded ring. The packet that makes the short-cut to k will remain the same, $P^{BRD}(s, s-1(mod\ m))$, and the packet that continues on the ring will be changed to $P^{BRD}(s, k-1(mod\ m))$.

For example, in Figure 2 a broadcast packet originated at virtual node 15 of node F ($P^{BRD}(15, 14)$) can be duplicated by making a jump from node G to virtual node 6 on node C. After this jump

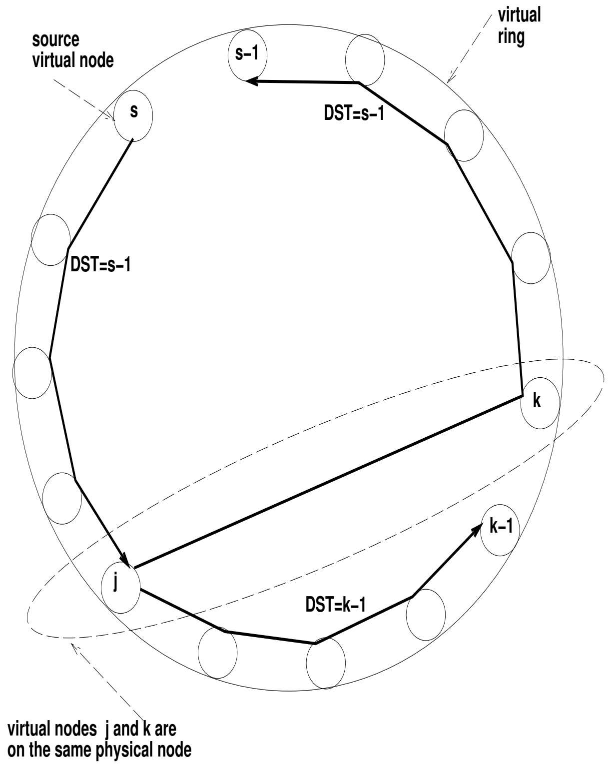

Figure 3: Broadcast Duplication Example

duplication the header of the packet that continues from node G is modified to $P^{BRD}(15, 5)$, and the header of the packet that made the jump remains unchanged - $P^{BRD}(15, 14)$.

The packet is received by a node only when it arrives to the virtual node that is the closest to its source. In the example of Figure 3, since the source virtual node is s, the packet is received into the host interface of a node when the broadcast arrives to the virtual node that is the closest to s on the virtual ring. In Figure 2 the broadcast originated at virtual node 15 of node F will be received, for example, by node A via VN_1, by node C via VN_6, and by node B via VN_2.

The number of times a broadcast packet is duplicated on a node depends on two factors: (i) the destination range and (ii) the short-cuts or jumps availability (whether or not the corresponding links are idle or busy). The method in which multiple duplications are handled is by trying to duplicate first to the closest available virtual node. After each duplication the current destination is modified and this operation is repeated until no more short-cuts or thread jumps are possible.

3.2 The Broadcast Algorithm

Notations:

- $VN_0, ..., VN_{m-1}$ are the m virtual nodes in this embedded ring arranged in a sequential order from 0 to $m - 1$.
- $\vec{X}$ and $\vec{Y}$ are logical names of physical nodes X and Y. The logical name consists of all virtual node names associated with that physical node.
- $P^{BRD}(VN_{SRC}, VN_{DST})$ is the broadcast packet header. Initially, the destination is the virtual

node right before the source virtual node. For example, if the source is s then the destination is $s-1(mod\ m)$, as shown in Figure 3.

- $VN_j \in \vec{X}$ is the virtual node the broadcast packet has arrived at.
- $T\vec{R}_{\vec{X}}$ is the set of all the threads connected to node X.
- $\vec{Y} \in T\vec{R}_{\vec{X}}$ is a thread that connects node X with node Y.
- $V\vec{N}_j$ is an ordered list of virtual nodes that a broadcast packet can be duplicated to, either by a short-cut or a jump via thread. This list contains only virtual nodes that reside on the virtual embedded ring between VN_j and VN_{DST}. The virtual nodes are arranged in this vector, such that, the first virtual node is the closest to the destination virtual node, and so on. In the case of threads, this vector will contain the name of the virtual node that is closest on the embedded ring to VN_j.

The algorithm for handling a broadcast packet that has arrived at a virtual node VN_j, on a physical node, is composed of the following basic steps:

1. **Compute $V\vec{N}_j$.** This step computes the vector of all virtual nodes that a broadcast packet received at VN_j can be duplicated, i.e., virtual nodes which are in the destination range of this broadcast packet.
2. **Receive.** Determines whether or not to receive the packet. The packet is received only once by each physical node, when it arrives at the virtual node that is closest to the source virtual node of this broadcast.
3. **Duplicate.** Broadcast packet is duplicated to a virtual node in $V\vec{N}_j$ when it is available. The header of a duplicated packet remains unchanged; the destination field of the original packet is modified after the duplication, in order to reflect that it has to cover a smaller segment of the virtual ring. After the modification is done, this packet can be duplicated again to another virtual node that is in the range of the modified destination field.

Broadcast Algorithm 1 for $VN_j \in \vec{X}$ after receiving $P^{BRD}(VN_{SRC}, VN_{DST})$:

1 **COMPUTE $V\vec{N}_j$ STEP**
1.1 $V\vec{N}_j = \emptyset$
1.2 For all $x \in \vec{X}$, if x is between VN_j and VN_{DST}, then add x to $V\vec{N}_j$
1.3 For all $\vec{Y} \in T\vec{R}_{\vec{X}}$:
1.3.1 Let y_{min} be y s.t. $min_{y\in\vec{Y}}\{y - VN_{SRC}(mod\ m)\}$
1.3.2 if y_{min} is between VN_j and VN_{DST}, then add y_{min} to $V\vec{N}_j$
1.4 Order $V\vec{N}_j$ s.t. the first element is the closest one to VN_{DST}, and so on
2 **RECEIVE STEP**
2.1 Let x_0 be x s.t. $min_{x\in\vec{X}}\{x - VN_{SRC}(mod\ m)\}$
2.2 If $VN_j = x_0$, then receive the packet

2.3 If $VN_j = VN_{DST}$, then remove the packet
and exit the algorithm

3 **DUPLICATE STEP**

3.1 If $\vec{VN_j} = \emptyset$ Goto 3.5

3.2 If first $d \in \vec{VN_j}$ is idle, then do

3.2.1 Send $P^{BRD}(VN_{SRC}, VN_{DST})$
via virtual node d

3.2.2 $VN_{DST} = d - 1$

3.3 $\vec{VN_j} = \vec{VN_j} - \{d\}$

3.4 If $\vec{VN_j} \neq \emptyset$ Goto 3.2

3.5 Send $P^{BRD}(VN_{SRC}, VN_{DST})$ via VN_j
and exit the algorithm

3.3 Broadcast-with-Feedback

Next we describe the algorithm for broadcast-with-feedback or echo (BF). The objective of this algorithm is to inform the source of the broadcast message that the packet has reached all the nodes in the network.

If our network was a simple ring, then this kind of feedback is very simple since the broadcast packet can continue one more hop from the destination node to the source node, and then be removed from the ring. In our broadcast algorithm the situation is not that simple any longer, since a single broadcast packet can be duplicated into many packets. Thus, the source should receive the feedback regarding the successful coverage of the network by multiple feedback of messages.

The broadcast algorithm with the multiple messages can be viewed as a tree (generated dynamically) in which its root is the broadcast source node, and the duplication points are the vertices of the tree. The nodes in which the broadcast packets are removed from the tree are its leaves. In the following algorithm, when a message is removed by a leaf node, a short feedback message will be sent back to the source.

This feedback control message will contain the header and the broadcast identifier as it was determined by its source. In the following description we ignore this identifier, since it is administrated locally by the source, therefore, we assume that when a source receives a feedback message it knows to what broadcast message it belongs.

The feedback message uses the source and destination fields in the header, but in the reverse order, i.e., the source field becomes the new destination and vice versa. (The self-routing property of the MetaNet will enable this packet to get back to its source.) The additional information needed by the source in order to determine whether or not the broadcast was successful is to add to the feedback message the first virtual node which a duplicated packet has traversed.

This additional information for the feedback algorithm is kept in a **pseudo source field**, $P - SRC$, in the header. Thus, in the following algorithm the broadcast header structure is $P^{BRD}(VN_{SRC}, VN_{DST}, VN_{P-SRC})$. The feedback packet that is sent back from a leaf is a simple packet with a single destination. In the header of this packet the source and destination fields of the received broadcast packet were switched, and it has the following format: $P(VN_{DST}, VN_{SRC}, VN_{P-SRC})$. When the packet is originated at its source, the pseudo source and source fields are the same, i.e., $VN_{P-SRC} = VN_{SRC}$.

BF Algorithm for $VN_j \in \vec{X}$ after receiving $P^{BRD}(VN_{SRC}, VN_{DST}, VN_{P-SRC})$:

1 **COMPUTE $\vec{VN_j}$ STEP**

(Repeat steps 1.1 to 1.4 of algorithm 1)

2 **RECEIVE STEP**
2.1 x_0 be x s.t. $min_{x \in \vec{X}}\{x - VN_{SRC}(mod\ m)\}$
2.2 If $VN_j = x_0$, then receive the packet
2.3 If $VN_j = VN_{DST}$, then do
2.3.1 Remove the packet
2.3.2 Send self-routing simple feedback message:
$P(VN_{DST}, VN_{SRC}, VN_{P-SRC})$
to the originating source
and exit the algorithm
3 **DUPLICATE STEP**
3.1 If $\vec{VN_j} = \emptyset$ Goto 3.5
3.2 If first $d \in \vec{VN_j}$ is idle, then do
3.2.1 Send $P^{BRD}(VN_{SRC}, VN_{DST}, d)$
via virtual node d
3.2.2 $VN_{DST} = d - 1$
3.3 $\vec{VN_j} = \vec{VN_j} - \{d\}$
3.4 If $\vec{VN_j} \neq \emptyset$ Goto 3.2
3.5 Send $P^{BRD}(VN_{SRC}, VN_{DST}, VN_{P-SRC})$
via VN_j, and exit the algorithm

The following is the broadcast check algorithm which receives all the feedback responses and confirms whether or not this broadcast was successful. It is easy to see from the algorithm that each feedback comes from a message that has covered a segment of the virtual ring between VN_{P-SRC} and VN_{DST}. After receiving all the feedback messages, every virtual node between 0 and $m-1$ should be in one of the segments.

Broadcast Check Algorithm after receiving $P(VN_{DST}, VN_{SRC}, VN_{P-SRC})$:

1 Add all virtual nodes between
VN_{P-SRC} and VN_{DST} to $\vec{FDBK}$
2 If $\vec{FDBK}$ contains all the virtual nodes
between 0 and $m-1$, then this broadcast
was completed successfully

3.4 Correctness Proof

We first prove the correctness of the broadcast algorithm and then show the correctness of the feedback extension.

Lemma 1 (Coverage): *A broadcast packet will reach each virtual node exactly once.*

Proof:
The proof is by a simple induction on the number p of packets. Initially, the packet sent from the originating source, VN_{SRC}^{ORIG}, with the destination $VN_{SRC}^{ORIG} - 1(mod\ m)$. If this packet is not duplicated (i.e., $p = 1$), then all virtual nodes are covered. Consider one duplication of the packet at some virtual node j to virtual node k (i.e., $p = 2$, see Figure 3). Then the virtual ring segment from VN_{SRC}^{ORIG} to j has been covered. The coverage of the remaining of the virtual ring will be realized by the two packets, such that, the first one will cover the segment from j to $k - 1(mod\ m)$ and the other will cover from k to $VN_{SRC}^{ORIG} - 1(mod\ m)$. Clearly, after this duplication the virtual ring will be completely covered.

Assume that after the pth duplication of a packet the virtual ring is still completely covered. Then at the $p+1$th duplication a packet with destination f is duplicated from virtual node d to e. The coverage of the virtual ring segment between d and f will be divided by the two packets: one will cover from d to $e-1(mod\ m)$ and the other will cover from e to f. Again, after this duplication the virtual ring will be still completely covered.
Q.E.D.

Lemma 2 (Safety): *A broadcast packet is copied once by each physical node.*

Proof:
The *coverage lemma* above ensures that all virtual nodes are covered. From statements 2.1 and 2.2 in the previous two algorithms, we know that a broadcast packet can be copied only by the virtual node that is the closest to its source, and therefore, is copied only once.
Q.E.D.

Next we prove a lemma on the feedback extension of the algorithm.

Lemma 3 (Feedback): *If a broadcast was completed successfully, the set of feedback messages informing on the coverage of the embedded ring, from all its leaf nodes, contains all the virtual nodes.*

Proof:
The pseudo source is the point on the virtual embedding where the packet was duplicated. Again by induction, initially, the pseudo source is equal to the source, and therefore, if the message is not duplicated to any virtual node, the feedback information will provide full virtual ring coverage.

Assume that after the ith duplication of a packet the feedback information still provides full virtual ring coverage. Then at the $i+1$th duplication a packet with destination f and pseudo source c is duplicated from virtual node d to e. The feedback information of the virtual ring segment between c and f will be divided by the two packets: the duplicated packet will have a pseudo source e and destination f and the other will have the feedback information from pseudo source c to destination $e-1(mod\ m)$. Thus, the combined feedback information remains the same, i.e., from c to $e-1(mod\ m)$ and from e to f.
Q.E.D.

3.5 Minimizing the Broadcast

In the previous broadcast algorithms all virtual nodes were covered exactly once by the broadcast. This is not really necessary since what is needed is that on every physical node the virtual node that is the closest to the source would be covered (the virtual node where the packet is copied into the node).

In this section, we present a modification of the previous algorithm, which reduces the number of links needed to be traversed in every broadcast. If the virtual ring was embedded on a tree, the number of virtual nodes traversed under the previous two algorithms is always $2(n-1)$ (n is the number of physical nodes). In the following algorithm under light load conditions the number of links traversed by a broadcast is only $(n-1)$.

The modification needed in the algorithm is simple. Consider the step 3.2.2 of the algorithm, if the destination virtual node, VN_{DST}, and the current virtual node, VN_j, are on the same physical node; then the destination value is decreased by one, $VN_{DST} = VN_{DST} - 1$. It is clear that the destination is always further away from the source, than the current virtual node. Since, the packet will not be copied from that destination there is no point in reaching that virtual node.

In order to maintain the correctness of the feedback algorithm, the original value of the destination, VN_{DST}, will be copied into a **pseudo destination field**, VN_{P-DST}, in the header. This field will be used in the feedback check at the source. The packet header format in the minimum broadcast algorithm is $P^{BRD}(VN_{SRC}, VN_{DST}, VN_{P-SRC}, VN_{P-DST})$. Initially, the two pseudo fields and the corresponding real fields are the same.

Minimum Broadcast Algorithm for $VN_j \in \vec{X}$ after receiving $P^{BRD}(VN_{SRC}, VN_{DST}, VN_{P-SRC}, VN_{P-DST})$:

1 **COMPUTE $\vec{VN_j}$ STEP**
(Repeat steps 1.1 to 1.4 of algorithm 1)
2 **RECEIVE STEP**
2.1 x_0 be x s.t. $min_{x \in \vec{X}}\{x - VN_{SRC}(mod\ m)\}$
2.2 If $VN_j = x_0$, then receive the packet
2.3 If $VN_j = VN_{DST}$, then do
2.3.1 Remove the packet
2.3.2 Send self-routing simple feedback message:
$P(VN_{DST}, VN_{SRC}, VN_{P-SRC}, VN_{P-DST})$
to the originating source
2.3.3 Exit the algorithm
2.4 If $VN_{DST} \in \vec{X}$, then $VN_{DST} = VN_{DST} - 1$
3 **DUPLICATE STEP**
3.1 If $\vec{VN_j} = \emptyset$ Goto 3.5
3.2 If first $d \in \vec{VN_j}$ is idle, then do
3.2.1 Send $P^{BRD}(VN_{SRC}, VN_{DST}, d, VN_{P-DST})$ via virtual node d
3.2.2 $VN_{DST} = d - 1$
3.2.3 $VN_{P-DST} = d - 1$
3.3 $\vec{VN_j} = \vec{VN_j} - \{d\}$
3.4 If $\vec{VN_j} \neq \emptyset$ Goto 3.2
3.5 Send $P^{BRD}(VN_{SRC}, VN_{DST}, VN_{P-SRC}, VN_{P-DST})$ via VN_j
3.6 Exit the algorithm

When the feedback packets are received by the broadcast source, in order to check full coverage, the node will use the pseudo source and destination fields in the header.

Broadcast Check Algorithm after receiving $P(VN_{DST}, VN_{SRC}, VN_{P-SRC}, VN_{P-DST})$:

1 Add all virtual nodes between VN_{P-SRC}
and VN_{P-DST} to $\vec{FDBK}$
2 If $\vec{FDBK}$ contains all the virtual nodes between
0 and $m-1$, then this broadcast
was completed successfully

In the following lemma we show that minimum broadcast is achieved for the case that there are only short-cut duplications. Using jump duplications further increase the number of messages, speeding-up the broadcast procedure, but may use slightly more than $n-1$ links.

Lemma 4 (Minimum Broadcast): *If a broadcast message from some virtual node i makes all possible short-cut duplications, on a spanning tree with n nodes, then this broadcast message is sent over $n-1$ links.*

Proof:
Assume that the broadcast starts at a virtual node which is at the root of a spanning tree. We will

show by induction that following two properties for minimum broadcast hold:
(P1) the distance of any broadcast message from the root can only increase, and
(P2) the destination virtual node and the next virtual node of a broadcast message are always on the same physical node.
Immediately from the tree structure, if these two properties hold, the broadcast is sent over $n-1$ links.

Initially, there is a packet at virtual node i and its destination is $i-1(mod\ m)$ at distance 1 from the root. There are two cases:
(i) this is a leaf node, which means that P1 and P2 hold, and
(ii) this is not a leaf node, which means that the first duplication is to the virtual node closest to the destination (statement 3.2 in the algorithm), i.e., it goes to the physical node of virtual node $i-1$, and again P1 and P2 hold.
If the first duplication was to virtual node i', the destination is modified to $i'-1$ (statement 3.2.2). Then if another duplication is possible it is performed to the virtual node that goes to the physical node of virtual node $i'-1$ (statements 3.3 and 3.2), which means again that P1 and P2 hold.
If no more duplication is possible, the broadcast message is forwarded to the physical node of its destination, and P1 and P2 hold.

Assume that a broadcast message arrives to a virtual node u at distance k from the root and its destination virtual node is v. Both u and v are on the same physical node, and therefore, the destination is decremented by one to $v-1$ (statement 2.4). Since no jumps were made, and given the tree structure, all possible short-cuts will be made to virtual node on the virtual ring segment between u and $v-1$. Next the broadcast message is duplicated and forwarded to distance $k+1$. This scenario is basically the same as in the initial step, and therefor, after all possible short-cuts were taken the two properties, $P1$ and $P2$, will hold.
Q.E.D.

4 Performance Study

In this section, the performance of the minimum broadcast algorithm is studied for two embedding structures: (i) the tree embedded ring (TER) and the (ii) simple embedded ring (SER), which is based on a unidirectional Hamiltonian cycle, as shown in Figures 4 and 5.

The minimum broadcast algorithm is executed once for each physical node, by taking that node as the source of the initial broadcast message. Thus, the performance of the algorithm is studied over N different runs. The performance measures for this study are the *time complexity* and the *message complexity*, which are averaged over the number of physical nodes (N=15 in our example).

4.1 Controlling the Duplication Probability

In the MetaNet the asynchronous access is regulated by a predefined *quota* given to each node in every *global* or *local* fairness cycle. The size of the *quota* controls the internal flow in the network and determines the *availability* of a link or alternatively the duplication probability using a *short-cut* or a *jump*. High *quota* increases the number of packets in the network which in turn increases the busy period of a link, and therefore, decreases the duplication probability. Thus, the external access *quota* is the control parameter for the MetaNet internal flow control.

Four different fairness algorithms can be used for allocating quotas and controlling the duplication probability - P:

1. Global on a virtual embedded ring - using the MetaRing's SAT algorithm [6].
2. Local on a virtual embedded ring - using the MetaRing's request (REQ) and grant (GNT) control signals [4].

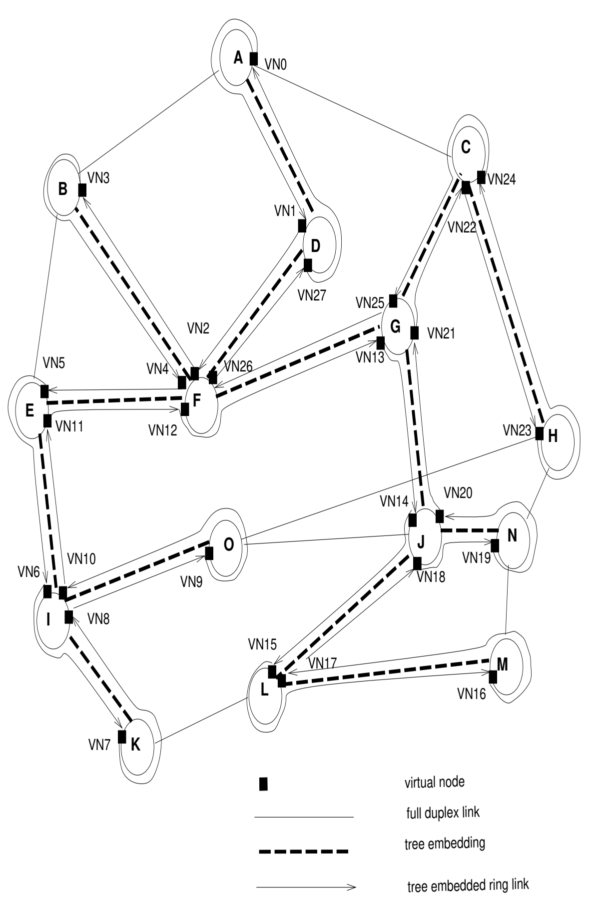

Figure 4: A Tree Embedded Ring

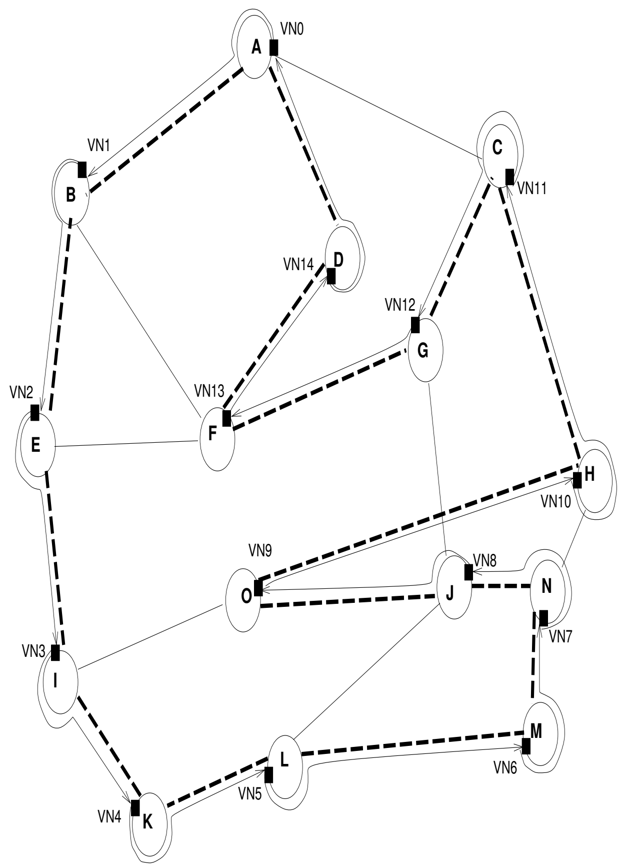

Figure 5: A Simple Embedded Ring

3. Global on a tree - using the MetaNet's SAT-BRD and SAT-MRG control signals [18].

4. Local on a tree - using the MetaNet's request (REQ) and grant (GNT) control signals [14].

4.2 Description of the Simulation

As stated above, the broadcast algorithm is originated once from each physical node. The algorithm is simulated with synchronous time steps such that at each step of the algorithm a broadcast message is sent across one link from one node to another. Note that the duplication processing time is included at the end of each time step. The number of time steps until all nodes receive a copy of the broadcast message determines the *time complexity* of the algorithm.

In addition, the duplicates of the original message, made during the execution of the algorithm, is counted to express its *message complexity.* These two measures are computed for each node and then their average is taken.

During the simulation of the algorithm, an ordered list of virtual nodes $V\vec{N}_j$ is computed at virtual node VN_j that receives a broadcast message. The ordering is determined according to the distance, computed on the virtual ring, from VN_j to the destination, such that, the closest virtual node to the destination is the first one in the list. The ordering implies a priority assignment for duplication of the broadcast message arrived at VN_j to the virtual nodes in $V\vec{N}_j$. A probabilistic decision is made to determine whether or not a duplication to a virtual node in $V\vec{N}_j$ is possible (note that this kind of assumption is often made in the performance study of other deflection networks). Since the number of duplicates generated by the algorithm is a function of the *availability*, P, the following typical values are considered:

1. The worst case is when P=0, which means that duplications are not possible. In this scenario the probability of finding an idle link is 0, i.e., no short-cuts or jumps are possible.

2. The average case is when P=0.5, which means that duplications are possible in half of the short-cuts and jumps.

3. The best case is when P=1, which means the the network is lightly loaded, and therefore, broadcast message duplications using short-cuts and jumps are always possible.

4.3 Comparison of the Two Embedding Structures

In Table 1, the time and space complexity measures are summarized for different *availability* values, P_J for jumps and P_S for short-cuts.

	Simple Embedded Ring		Tree Embedded Ring	
Probability	Time Complexity	Message Complexity	Time Complexity	Message Complexity
$P_J = P_S = 0$	14	1	20.6	1
$P_J = P_S = 1/4$	12.1	2	12.8	5
$P_J = P_S = 1/3$	11.2	2	11.7	5
$P_J = P_S = 1/2$	9.5	3	6.9	8
$P_J = P_S = 2/3$	8.2	4	6.7	9
$P_J = P_S = 3/4$	7.3	4	6	9
$P_J = P_S = 1$	5	5	5	14
$P_J = 2/3, P_S = 1/3$	8.2	4	10.9	6
$P_J = 1/3, P_S = 2/3$	10.9	2	9	9
$P_J = 3/4, P_S = 1/4$	7.3	4	12.3	6
$P_J = 1/4, P_S = 3/4$	11.8	2	6	10

Table 1: Comparison of the embeddings

Time complexity:
The time complexity of the algorithm is shown in the first column of each embedding structure in Table 1. Evidently, under the light load assumption (i.e., high availability), the time complexity of the broadcast algorithm is about the diameter of the network. However, as the load in the network increases, the time complexity approaches the size of the virtual embedded ring. Since this size is smaller for the SER structure, under heavy traffic conditions, it may provide a better performance results.

Message complexity:
The message complexity is shown in the second column of each virtual ring embedding in Table 1. The number of broadcast messages increases as the internal load decreases. Since the SER structure supports *jump* operations only, the message complexity on this embedding is smaller than for TER. The TER structure supports both short-cuts and jumps, and therefore, there are more messages, which can provide a shorter time complexity, than SER, as shown in Table 1.

The total number of times a broadcast message is transmitted over a link, or the *communication complexity*, is $N-1$ for SER and is between $2(N-1)$ and $N-1$ for TER. Therefore, when $P_J = P_S = 1$ in the TER case the message complexity is 14, but the communication complexity for minimum broadcast will be only about N. This means that each copy of the broadcast message is transmitted over small number of links.

5 Discussion

We have presented the MetaNet architecture and supplied it with the important broadcast-with-feedback mechanism. This architecture can serve as both a high-speed connection-less general network and as an arbitrary topology LAN.

The broadcast on the MetaNet is dynamic and the actual pattern of each broadcast message depends on the instant load inside the network. Furthermore, the broadcast-with-feedback operation is completely loss-free due to congestion inside the network. This together with the very low bit error rate on today's high-speed fiber optic networks will ensure a successful completion of broadcast operations, from arbitrary number of sources, in almost all cases.

The performance of the broadcast algorithm is compared in two embedding structures and verified computationally that under light load the time complexity is bounded by the diameter of the network. Under heavy internal load conditions the time complexity of the broadcast algorithm can approach the size of the virtual embedded ring. However, since the internal traffic load condition is regulated and controlled by a global or local fairness mechanism, the heavy load condition is not likely to occur. This property is of great interest and is subject of further study.

In addition, the performance study shows that the message complexity and the time complexity are inversely related. The message complexity is captured by the number of duplicates of the original broadcast packet which is demonstrated as a function of the load in the network (i.e., *availability* of the links). The time complexity is measured as the maximum number of time steps it takes to deliver a broadcast message to all the nodes.

Broadcast has various applications.
(1) The broadcast with its feedback mechanism can be used to implement various synchronization and coordination tasks as part of the network control and management procedures.
(2) The broadcast can be used to compute a *global sensitive functions*. Suppose each node stores a variable and we want to add-up all these values, this is an example of a semi-ring operation, exclusive-or of bits, addition, minimum, maximum, and so on. Using the fact that a broadcast can wake-up all nodes and then a broadcast-with-feedback mechanism which sum-up the values.

References

[1] H. Ahmadi and W. E. Denzel. A survey of modern high-performance switching techniques. *IEEE J. on Selected Areas in Comm.*, SAC-7(7):1091–1103, September 1989.

[2] P. Baran. On distributed communication networks. *IEEE Trans. on Communications Systems*, CS-12(1-2):1–9, March 1964.

[3] W. Bux, F. H. Closs, K. Kummerle H. J. Keller, and H. R. Mueller. Architecture and design of a reliable token-ring network. *IEEE J. on Selected Areas in Comm.*, SAC-1(5):756 – 765, November 1983.

[4] J. Chen, I. Cidon, and Y. Ofek. A local fairness algorithm for the MetaRing and its performance study. *Proc. IEEE GLOBECOM'92*, pages 1635–1641, 1992.

[5] I. Cidon and Y. Ofek. MetaRing: A full-duplex ring with fairness and spatial reuse. *INFOCOM'90*, pages 969–981, 1990.

[6] I. Cidon and Y. Ofek. MetaRing - a full-duplex ring with fairness and spatial reuse. *IEEE Trans. on Comm.*, COM-41(1):110–120, January 1993.

[7] Y.K. Dalal and R.M. Metcalfe. Reverse path forwarding of broadcast packets. *Comm. of the ACM*, 21(12):1040–1048, April-June 1978.

[8] S.E. Deeering and D.R. Cheriton. Multicast routing in datagram internetworks and extended lans. *ACM Trans. on Computer Systems*, 8(2):85–110, May 1990.

[9] E. R. Hafner, Z. Nenadal, and M. Tschanz. Integrated local communications - principles and realization. *Hasler Review*, 8(2):34–43, 1975.

[10] A. R. Jacob. A survey of fast packet switches. *Computer Communication Review*, 20(1):54–64, January 1990.

[11] P. Kermani and L. Kleinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks*, 3:267–286, September 1979.

[12] M. T. Liu and D. M. Rouse. A study of ring networks. *Proc. IFIP WG6.4/University of Kent Workshop on Ring Technology Based Local Area Networks*, pages 1–39, September 1983.

[13] N. F. Maxemchuk. Routing in the manhattan street network. *IEEE Trans. on Communications*, COM-35(5):503–512, May 1987.

[14] A. Mayer, Y. Ofek, and M. Yung. Local fairness in general topology networks with convergence routing. *INFOCOM'95*, 1995.

[15] R. Melen and J. Turner. Distributed protocols for access arbitration in tree-structured communication channels. *IEEE Trans. on Communications*, COM-39(3):416–425, March 1991.

[16] R. M. Metcalfe and D. R. Boggs. Ethernet: Distributed packet switching for lacal computer networks. *Communications of the ACM*, 19(7):395–404, July 1976.

[17] Y. Ofek. Overview of the MetaRing architecture. *Computer Networks and ISDN Systems*, 6-8:817–830, 1994.

[18] Y. Ofek and M. Yung. Efficient mechanism for fairness and deadlock-avoidance in high-speed networks. *The 4th International Workshop on Distributed Algorithms*, pages 192–212, September 1990.

[19] Y. Ofek and M. Yung. Principles for high speed network control: lossless-ness and deadlock-freeness, self-routing and a single buffer per link. *9-th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 161–175, August 1990.

[20] Y. Ofek and M. Yung. Asynchronous lossless broadcast-with-feedback on the MetaNet architecture. In *IEEE INFOCOM*, pages 1050–1063, April 1991.

[21] Y. Ofek and M. Yung. Routing and flow control on the MetaNet: an overview. *Computer Networks and ISDN Systems*, 6-8:859–872, 1994.

[22] J.M. McQuillan I. Richer and E.C. Rosen. The new routing algorithm for the arpanet. *IEEE Trans. on Communications*, 28(5):711–719, May 1980.

[23] M. D. Schroeder, A. D. Birrell, M. Burrows, H. Murray, R. M. Needham, T. L. Rodeheffe E. H. Satterthwaite, and C. P. Thacker. Autonet: a high-speed, self-configuring local area network using point-to-point links. *IEEE J. on Selected Areas in Communications*, 9(8):1318–1335, October 1991.

[24] A. Segall. Distributed network protocols. *IEEE Trans. on Information Theory*, 29(1):23–35, January 1983.

[25] J. Turner. New directions in communications (or which way to the information age?). *IEEE Communications Magazine*, 24(10), October 1986.