Fault-Tolerant Convergence Routing

Bülent Yener
Department of Computer Science
Columbia University
New York, NY 10027

Inderpal Bhandari, Yoram Ofek and Moti Yung
IBM T.J.Watson Research Center
P.O.Box 704
Yorktown Heights, NY 10598

Abstract

This paper presents fault-tolerant protocols for fast packet switch networks with *convergence routing*. The objective is to provide, after a link or a node (switch) failure, fast reconfiguration and continuous host-to-host communication. *Convergence routing* is a variant of *deflection routing*, which combines, in a dynamic fashion, the on-line routing decision with the traffic load inside network. Unlike other deflection techniques, *convergence routing* guarantees that packets will reach or converge to their destinations.

This fault-tolerant solution is designed for a switch-based (i.e., arbitrary topology) LAN architecture called MetaNet. In this work, the original MetaNet's *convergence routing* scheme has been modified in order to facilitate the property that the packet header need not be recomputed after a failure and/or a reconfiguration. This is achieved by having, at the network interface, a translator that maps the unique destination address to a virtual address. The virtual addresses are stored at the packet header, and used for *convergence routing*, with *global sense of direction* over (i) a single spanning tree, and (ii) over two edge-disjoint spanning trees - for redundancy (fault tolerance) and greater efficiency.

1 Introduction

Recent advances in processor and communication technologies are enabling the evolution of local area networks (LANs) towards switch-based architectures. This work presents new protocols to obtain a fault-tolerant switch-based LAN, which is based around an architecture called MetaNet [8, 10].

Fault-tolerant computing implies correct execution of a specified algorithm or protocol (e.g., routing protocol) in the presence of faults. The goal of such computing is to make systems highly available and/or highly reliable [13]. In the context of networking, fault-tolerance is usually equated with reconfigurability. Traditional networks such as the Internet and the IBM Token-ring [4] have reconfiguration capabilities. Networks with reconfiguration capability suspend their normal operation during the reconfiguration procedure. This means that a failure can cause a protocol to stop even in parts of the network that are not directly affected by the fault. As a consequence, availability of the entire network can be affected. Furthermore, as the number of nodes in a network increases, so does the probability of failures and it follows that a high-speed network that cannot be either reconfigured quickly or tolerate failures may suffer from poor availability. In addition, host-to-host (or end-to-end) communication, between nodes that are active and reachable, cannot be suspended during reconfiguration, since some computation and communication environments have the stringent requirement of continuous availability.

The solution presented in this paper expands the basic features of the MetaNet architecture to include fault-tolerant properties such as continuous host-to-host communication during reconfiguration. In the original design of the MetaNet, while the fault-tolerance capability was recognized as an essential feature for arbitrary topology networks, explicit fault-tolerance was provided only for the recovery and stabilization of the fairness control mechanism [7]. The virtual embedding of ring or rings on the MetaNet facilitates the new *convergence routing* method, which is a variant of deflection routing along *global sense of direction* inside the network. In this paper, we solve the problem of fault-tolerant convergence routing by providing: (i) a simple mapping between the *globally unique address labels* and the sequential *virtual addresses* along the virtual ring embeddings, (ii) a fast local reconfiguration procedure, called *cycle closure* (of the virtual rings), and (iii) an algorithm which constructs *two redundant edge-disjoint virtual rings*, which is sufficient to recover from failures without suspending host-to-host communication.

This paper is organized as follows. We first discuss the fault model in Section 2. That discussion describes the support that we expect at the physical and data link layers to monitor and detect faults. This is followed by a basic description of the MetaNet architecture and the required enhancements for fault-tolerance, in Section 3. In particular, in that section, the use of locally-computed mappings, for translating the unique address labels to virtual addresses, is described. In Section 4, the reconfiguration issues are studied, and a simple algorithm is presented for a fast and local reconfiguration of a virtual ring which is embedded on a spanning tree. In Section 5, we consider two edge-disjoint virtual rings and show how redundancy is used for fault-tolerant *convergence routing*. The work is concluded in Section 6.

2 Fault Model

Network nodes and links often go down and come up again, causing disruptions in the system operation. The mechanisms to sense (detect) these phenomena, built into the low protocol layers of the network proto-

0-8186-6685-4/94 $4.00

cols, largely defines the fault model. In turn, the fault model fundamentally limits the reconfiguration strategy - since the efficiency of such a strategy depends on the accuracy of the model. Therefore, we proceed with a description of our assumptions on the fault-sensing mechanisms provided by a layer lower than the network layer.

Local fault-sensing mechanisms:
It is assumed that all links are bidirectional or full-duplex. A bidirectional link is considered to be DOWN if one of its directions is DOWN. It is assumed that each node can sense if a link or a nodes to which it is connected to is UP or DOWN. A link or a node is UP if it is perceived to behave "normally", and it is DOWN otherwise. There are several mechanisms of varying degrees of sophistication that can define the normal operational state of a node or a link. For example, if a node can communicate over a certain link with a neighbor, regardless of the error rate, this link is considered to be UP. Otherwise it is assumed to be DOWN. However, a more sophisticated scheme would consider the error rate and would assume that the link is DOWN if the rate exceeds a predefined threshold.

The fault-tolerant strategy presented in this work is compatible with these mechanisms. What is required is a mechanism that enables a node to determine whether a link or a node is UP or DOWN. This work primarily consider asynchronous, multiple link failures. Thus, a node failure can be treated as the failure of all its incident links.

Global fault-sensing mechanisms:
The MetaNet architecture has two global mechanisms that can be used for fault-sensing. The first is the *broadcast-with-feedback* algorithm [9], which enables a node to check the entire network by broadcasting a *test* message, and then waiting for all the feedback messages. A simple algorithm on the originating node will combine all the feedback messages. In this way the node can test whether or not the network is connected and whether or not the links used in its virtual rings are operational.

The second one is the fairness algorithm on spanning trees [7], which can be used to report on the network connectivity status. The algorithm has broadcast and merge phases which together constitutes a dynamic global cycle. The cycle is created by sending control signals to all the leaves, and these signals are then merged back to a dynamic root node. This mechanism has an upper bound on its cycle time, so absence of these signals for more than the time bound can be interpreted as a spanning tree failure.

3 Fault-Tolerant Convergence Routing

Convergence routing is a variant of *deflection routing* [2, 6]. It combines, in a dynamic fashion, the on-line routing decision with the traffic load inside the 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 it suffers no loss due to congestion.

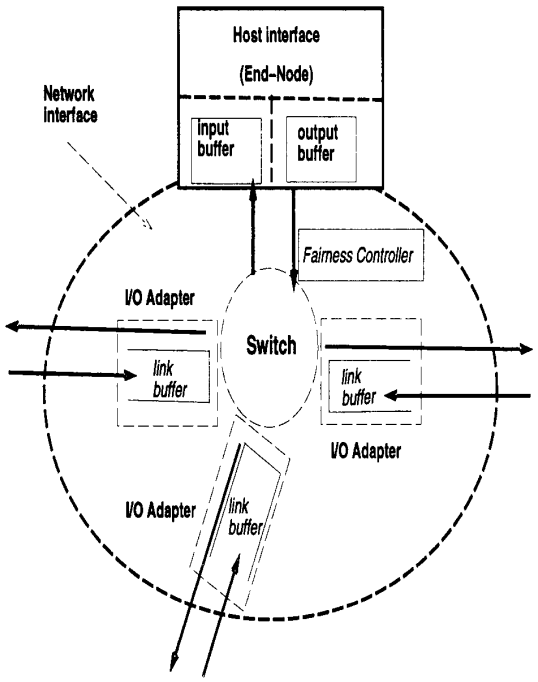

Figure 1: Node Structure and Network Interface

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.

3.1 Embedding of Unidirectional Virtual Rings

In any connected network with bidirectional links, it is always possible to have at least one global embedded ring, by embedding a virtual ring along a spanning tree, as shown in Figure 2. In this paper we concentrate only on that type of embedding structure, which is called a *tree embedded ring* (TER).

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 embeddings 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 tra-

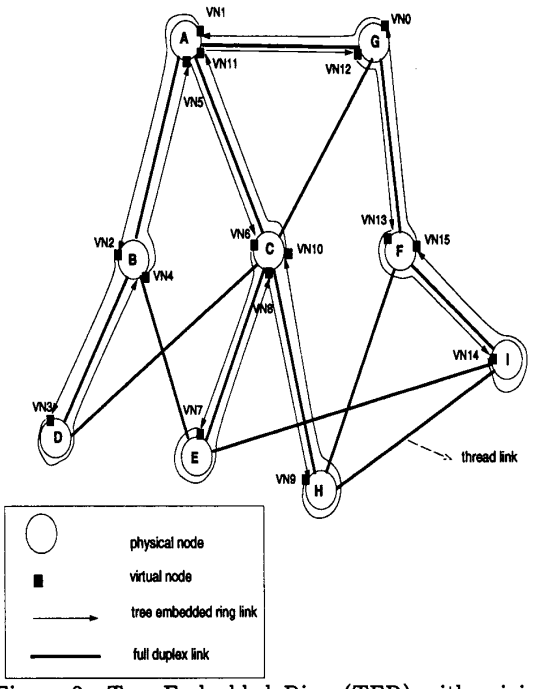

Figure 2: Tree Embedded Ring (TER) with origin Node G

verse a ring link only once. This is needed in order to prevent congestion and loss.

The virtual ring links are numbered sequentially from 0 to $m-1$, where m is the number of virtual nodes ($m = 16$, 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 ring should be closed, i.e., ring link $m-1$ (ring link 15, in Figure 2) should be connected to ring link 0.

We define two types of addresses on this network. Each node has its own **unique address label**, 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**. The virtual nodes receive 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 nodes), such that the distance between them, measured on the virtual embedded ring, is the minimum. Note that under this definition each direction of a thread link may be associated with different virtual nodes. For example, the thread link from C to G, in Figure 2, is connecting virtual nodes VN_{10} with VN_{12}, but in the other direction, the thread from G to C is connecting virtual nodes VN_0 and VN_6.

3.2 Convergence Routing with Global Sense of Direction

An important property of convergence routing is **global sense of direction**, which 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). Convergence 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 between source and destination is not fixed, but may change dynamically as a function of the load condition at the switches along the route. The convergence routing decision is based on computing the distance between two virtual nodes on the virtual embedded ring. Let m be the size of the virtual ring. To find the distance on the ring from virtual node u to virtual node v, the following simple calculation is performed:

$$DIST(u,v) := v - u \ (mod\ m) \qquad (1)$$

Similarly, the distance between two nodes is measured between 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 ring from node U to node V, the following calculation is thus used:

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

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 convergence routing paradigm:
The convergence 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 the distance from its destination at the next node (using Eq. 1) is *always* less than the distance from the current one.

Furthermore, we note that a 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 destination virtual node that is closest to the source virtual node. 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. Two methods are used:
(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 the packet may short-cut to VN_{10}, and if VN_{12} is idle the packet will jump to it using the thread link from VN_{10} to VN_{12}.

3.3 Local Assignment of the Virtual Addresses: LAVA

Fault-tolerant converge routing requires the following modifications to the basic routing algorithm explained above: (i) the packet header is changed to contain the unique address label of the destination, instead of the virtual address of the destination, and (ii) the design of the switch at every node includes a routing table with entries for the unique address labels of all nodes. We note that the unique address label assigned to each node is short; therefore, the number of entries in the routing tables can be small. For example, 8 or 12 bit address labels will require only 256 or 4096 entries in the routing table at the network interface.

The routing table, which resides in the network interface (see Figure 1), maps the unique address label in the header to the virtual addresses of the destination. The main motivation for this mapping is to avoid the necessity of recomputing the destination field at the packet header after each reconfiguration. Therefore, in the fault-tolerant design each node considers itself as the origin of the virtual ring, and assigns the virtual addresses to the nodes *locally* using the *same* underlying tree. As a result—in contrast with the original MetaNet design—*virtual addresses have only local significance in the fault-tolerant MetaNet.* For example, consider node C which has the virtual addresses $\{VN_6, VN_8, VN_{10}\}$ on node G, in Figure 2, and $\{VN_9, VN_{11}, VN_{13}\}$ on node F, in Figure 3. However, both G and F uses the same spanning tree structure for the routing computation.

Next we present the procedure (LAVA), executed at some node U, for local assignment of the virtual addresses to the nodes (see Figure 3.3), and then show that the *convergence routing with global sense of direction* of the original design is still preserved in the fault-tolerant convergence routing.

Routing with Local Assignment of Virtual Addresses

The distance computation is also realized by Equation 2 for the fault-tolerant convergence routing. Although the virtual nodes have only local meaning, the size of the virtual rings is the same on all nodes (for a network of n nodes it is $m = 2n - 2$). Since the same tree (i.e., same unique address labels arranged in the same lexicographic order from left to right) is used by all the nodes to compute their LAVA, the number of hops between two end points on the virtual ring is the same regardless of the origin of the virtual ring.

More precisely, consider the LAVA computed at node X. Let u_X, v_X be the virtual nodes corresponding to the left-most child of the nodes U and V, respectively. Similarly, let u_Y, v_Y be the corresponding virtual nodes with the LAVA at node Y. It is simple to observe that:

$$DIST(u_X, v_X) = DIST(u_Y, v_Y), \qquad (3)$$

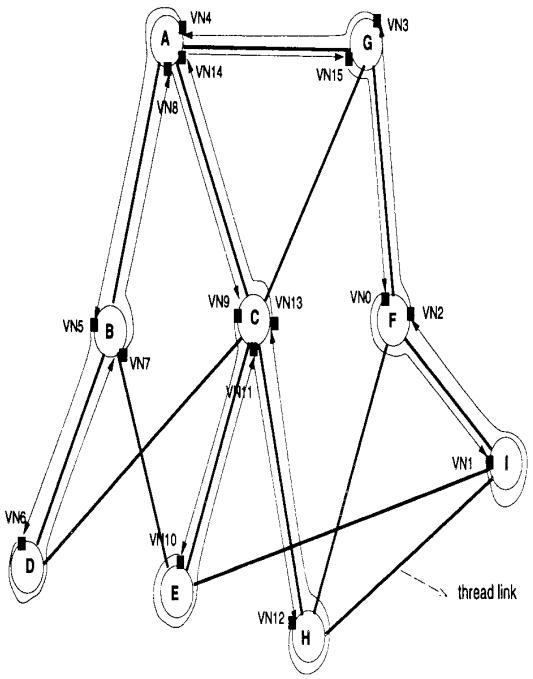

Figure 3: Tree Embedded Ring with origin Node F

Procedure *LAVA for Node U*

1. U considers itself as the root of the underlying spanning tree and orders the children of every node lexicographically from left to right using their unique address labels.
2. U traverses the tree in *depth-first order* starting from the port which leads to the left-most child and assigns the number 0 to this port. Increments the numbering every time an input port of a node is encountered during the traversal.
3. Each input port of each full-duplex link of the tree is encountered exactly once, and the number of time a node is traversed is the degree of that node in the tree.

Figure 4: Local Assignment of Virtual Addresses

since $v_X - u_X \ (mod\ m) = v_Y - u_Y \ (mod\ m)$. For example, the distance from node D to node I, (see Figure 2) using the LAVA at node G is $DIST(u_G, v_G) = 14 - 3 \ (mod\ 16) = 11$, whereas using the LAVA at node F it is $DIST(u_F, v_F) = 1 - 6 \ (mod\ 16) = 11$, as expected, both are identical.

4 Reconfiguration

In this section we consider a single virtual ring embedded on a spanning tree, which is assumed to be known to all the nodes, and examine two issues: (i) how to maintain *partial* host-to-host communication, by a fast and *local* reconfiguration of the virtual ring, and (ii) how to achieve a *global* reconfiguration to resume the normal operating conditions, in case of link failures. The global reconfiguration protocol has the unique property that it does not require the suspension of the partial host-to-host communication.

The first problem is addressed by a procedure called *cycle closure* to locally bypass the faulty virtual ring link. The objective is to obtain disjoint subrings for the connected nodes in order to ensure continuous host-to-host communication among them. The second problem, of global reconfiguration, can be solved by a distributed *spanning tree maintenance* procedure. The tree maintenance problem is well studied and many solutions have been described in the literature (e.g., [1, 11]). Any distributed protocol in which each node participates in the computation of a new spanning tree will serve our purpose.

4.1 Partial Host-to-Host Communication: Cycle Closure

When a link changes state from UP to DOWN, it will take some time before it is configured out of the network by the global configuration procedure. If it is a thread link, then the immediate impact of a failure can be ignored since the virtual ring remains connected. However, if the faulty link is a ring link, the virtual ring would be disconnected. This corresponds to two subtrees of the underlying spanning tree. A fast reconstruction of *virtual subrings* on the subtrees—to maintain host-to-host communication among the nodes that reside on the same subtree (i.e., partial host-to-host communication)—is essential. Since the network routing and control protocols rely on the existence of a virtual ring, this local reconfiguration for partial host-to-host communication is important for high availability until a complete reconfiguration is performed. The problem of how to preserve host-to-host communication among *all the active nodes* in the presence of link/node failure is addressed in Section 5. Next we present a procedure called cycle closure for maintaining partial host-to-host communication.

Cycle closure is based on partition of the nodes of an underlying spanning tree into two node-disjoint subrings. For example, suppose U detects a ring link failure with its neighbor V. Let u_j be a virtual node of U which is associated with the faulty ring link, and let u_i be the virtual node of U that is closest downstream to u_j. The algorithm cycle closure, executed at U, as given in Figure 4.1, shows how to connect u_j and u_i and specifies what information should be sent to other nodes in the network.

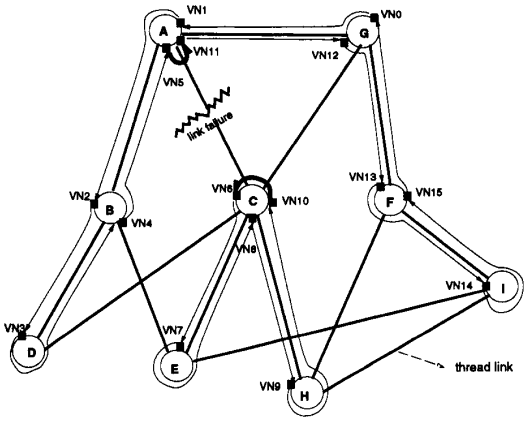

Figure 5: Cycle Closure

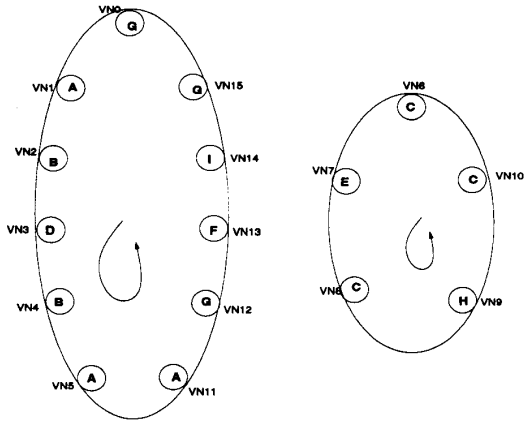

Figure 6: Node-Disjoint Subrings

Algorithm Cycle-Closure at Node U

1. connects u_j to u_i - construct a subring
2. generates a *link-failure* message which indicates the faulty link
3. initiates *fault-broadcast* by sending the link-failure message on all out-going ring links using reliable data link control (DLC) protocol

upon receiving the link-failure message, each node V executes the following operations

Procedure *fault-broadcast:*

1. routes the message reliably on all out-going ring links (without short-cuts or jumps), using a reliable data link control protocol
2. *prunes* the tree to determine the subtree—created by the faulty link—in which node V resides on
3. computes LAVA, and a new routing table on which the distance to the nodes outside the connected subtree is set as $+\infty$

Figure 7: Cycle Closure Algorithm

For each link failure that disconnects the underlying tree structure (i.e., a ring link failure), a link-failure message is generated. For example, in Figure 5, if the link between A and C is DOWN, then node A will perform cycle closure by connecting virtual nodes VN_5 to VN_{11}. Similarly, on node C, VN_{10} is connected to VN_6. The two subrings created by the cycle closure are shown in Figure 6. Nodes A and C send link-failure messages from virtual nodes VN_1, VN_{11}, and VN_6, VN_8, respectively.

We show in the following that once faults cease to occur, every node in the same subtree knows its local topology (i.e., the topology of the subtree it resides in). As a result, it follows that for each faulty ring link, there exists a virtual subring embedded on the underlying subtree. Thus, by accepting new traffic among the nodes that reside on the same subring, partial host-to-host communication is maintained.

Claim 1 *Algorithm Cycle-Closure maintains host-to-host communication between the nodes that reside in the same connected subtree.*

Proof: The goal of the proof is to show that each node knows correctly on what subtree it resides on. Consider two cases such that a node either (i) does not receive the link-failure broadcast message, or (ii) it receives multiple link-failure messages. The first case cannot be true unless the subtree—created by this link-failure—has also a link failure. This is true since the link-failure message is broadcasted over the subtree that this node resides on. Suppose that the subtree is broken into two components by a new link failure. In this case, the missing link-failure message (i.e., the message for the first link failure) is outdated, and can be overwritten by the new link-failure message. Thus, pruning the subtree according to the received link-failure message is sufficient for determining the correct subtree.

Note that since a reliable data link control protocol is used, we rule out the case that the link-failure message is lost.

To address the second case, we note that the receiving order of the multiple link-failure messages is not significant, since each link-failure message reports a new link failure which divides the subtree into two components. Consequently, regardless of the order of pruning, once a node receives no more link-failure message it knows the subtree in which it resides on, and thus partial host-to-host communication is ensured.

As we stated above, the objective of the cycle closure is to maintain the host-to-host communication among the nodes that reside in the same subtree. After computing LAVA, routing tables are updated. Setting the distances to the nodes—that are not in the same subtree—as $+\infty$, the routing operations are limited to the nodes that reside on the same subtree. Thus the convergence routing among them is ensured. □

Upon execution of the cycle closure, there are two types of packets on a virtual subring: the packets whose destination reside inside the subring (*local traffic*), and the ones with destination on another subring (*remote traffic*). We assume that each node that detects remote traffic discards it from that subring.

Algorithm Global Reconfiguration

1. **PHASE 1**: The root node of the new spanning tree sends (by a reliable DLC protocol) the *New-Tree* information to all the nodes using the new tree.
2. Each node upon receiving the *New-Tree* message
   2.1 sends the *New-Tree* to all its children
   2.2 acknowledges its parent node after receiving ACKs from all its children
   2.3 computes the new LAVA
   2.4 stops sending packets over its thread links
3. **PHASE 2**: The root node upon receiving the ACKs from all its children broadcast a *Down-load* message, over the *New-Tree*, to update the routing tables.
4. Each node upon receiving the broadcast message
   4.1 down-loads the new LAVA into its routing table
   4.2 acknowledges its parent node after receiving ACKs from all its children
   4.3 does not send packets over its thread links
   4.4 packets received on thread links are looped back
5. **PHASE 3**: The root node upon receiving the ACKs from all it children
   5.1 broadcasts that all nodes are already using the new tree
   5.2 thus, there is no need to loop back the packets received on thread links
   5.3 broadcast to all nodes that thread links can be used

Figure 8: Global Reconfiguration Algorithm

4.2 Global Reconfiguration: Spanning Tree Maintenance

In this section we consider how to construct a new global virtual ring on a tree, in order to resume global host-to-host communication. This is realized by a distributed algorithm for computing a new spanning tree (*New-Tree*) (e.g., [1, 11]). The protocol for global reconfiguration has three phases, as shown in Figure 4.1, and it does not require the suspension of the partial host-to-host communication.

The first phase starts with broadcast of the New-Tree structure from a root or leader node to all other nodes. Then there is bottom-up propagation of the acknowledgements (ACKs) starting from the leaves via all the intermediate nodes of the tree to the root node. During this phase each node also computes the new LAVA.

The second phase starts after the root is acknowledged that all the nodes have the same *New-Tree* information. Then the root broadcasts the *Down-Load* message using the new spanning tree. Each node upon receiving this broadcast message updates its routing table with new LAVA. After receiving ACK messages from his children on the tree, the node sends an ACK to its parent. Note that a node starts using its new routing table immediately. Thus, the routing tables at some nodes may be inconsistent (some with the old LAVA and some with the new LAVA). Consequently, the routing decision of some packets during this time may violate the convergence routing principle (i.e., distance from the destination of a packet is not guaranteed to decrease).

There are four cases to be considered for a packet transmitted on a link e (between nodes i, and j) during the execution of the global reconfiguration algorithm, assuming that i and j have inconsistent routing tables:

1. e is a thread link in i's routing table but a ring link in j's routing table
2. e is a ring link in i's routing table but a thread link in j's routing table
3. e is a thread link in both i's and j's routing tables
4. e is a ring link in both i's and j's routing tables

Regardless of which node has updated the routing table, the algorithm takes a pessimistic approach—in order to avoid congestion and packet loss—and prevents the use of thread links (step 2.4 and 3.3, in Figure 4.1).

Similarly, the packets that arrive on thread links of the new tree are looped back (step 3.4, in Figure 4.1). Thus, during the global reconfiguration, routing only via the ring links (i.e., case 4) is permitted.

In the third phase, the root node ensures that all nodes have executed the second phase successfully. Then it broadcasts to all nodes that the new LAVA tree is effective, and therefore, there is no need to loop back the packets that arrive on thread links. Lastly, the root node broadcast to all nodes that it is possible to use the thread links.

First, note that in the above procedure, the host-to-communication is not suspended during the update of the new routing tables - only the use of the thread links has been suspended. Second, there will be no packet loss during this inconsistent period, but only the convergence of the packets to the destination may be delayed until the end of the reconfiguration protocol.

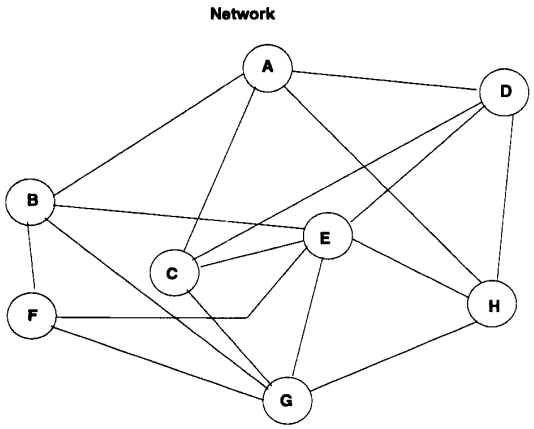

Figure 9: Network

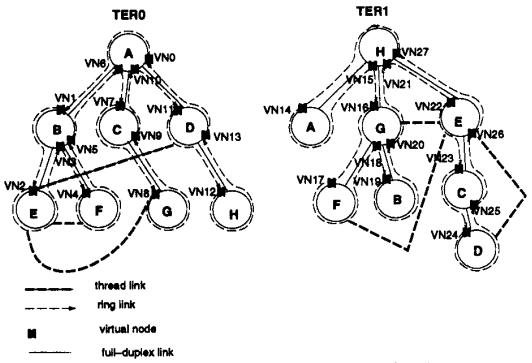

Figure 10: Edge-Disjoint Virtual Rings

5 Fault-Tolerance with Multiple Virtual Rings

In Section 4 we presented a method for maintenance of the host-to-host traffic on a tree embedded virtual ring under multiple link failures. The host-to-host communication, in that section, was maintained only among the nodes that resided on the same subtree after the failure. In this section we introduce a more powerful method which ensures the maintenance of communication between all non-faulty (*active*) nodes while tolerating node failures. Therefore, this method can provide *complete* host-to-host communications, and it is based on redundant virtual ring embeddings. In particular, we consider two *edge-disjoint virtual rings*, constructed on two edge-disjoint spanning trees in the network. The problem of finding two edge-disjoint spanning trees of a given network (called the 2-Tree problem) is studied in [15].

5.1 Two Edge-Disjoint Spanning Trees and Fault-Coverage

Two edge-disjoint spanning trees can tolerate link faults. For example, consider the network in Figure 9 which contains two edge-disjoint trees $Tree_0$ and $Tree_1$.

Let TER0 and TER1 be the virtual rings embedded over these trees, respectively (see Figure 10). Suppose, the link (A,C) goes down in the TER0. Then the nodes C and G become unreachable to the rest of the nodes on TER0. However, the traffic to/from nodes C and G can be routed on TER1.

In addition to the capability of handling link failures, two edge-disjoint spanning trees can tolerate some node failures. Specifically, we distinguish the leaf nodes in the trees from the non-leaf (intermediate) nodes, since failure of a leaf-node does not disconnect the rest of the tree. In other words, a tree with a failed leaf-node is considered to be non-faulty since it is still the *spanning tree of all active nodes.* Therefore, we consider three cases:
(1) the faulty node is a leaf in both trees,
(2) the faulty node is a leaf in one of the spanning trees, and
(3) the faulty node is a non-leaf in *both* trees, which we call a *critical node.* In the first case, a *graceful degradation of performance* is provided by the cycle closure algorithm which simply excludes this node from both of the trees. Thus, the impact of this node failure on the network is minimal and host-to-host communication among all non-faulty nodes will continue.

In the second case, it is possible to maintain complete and continuous host-to-host communication over one of the spanning trees, which still spans all the active nodes.

In the third case, both virtual rings will be disconnected and recomputation of the new spanning trees of the active nodes is necessary (using the techniques described in [15]). Thus, the failure of a critical node cannot be tolerated using two edge-disjoint spanning trees, and is not in the scope of this paper.

In the rest of this work, we consider only the second case. First we define *fault-coverage* as the ratio between the number of *critical nodes* and the total number of nodes. Then we present heuristics based on construction of two edge-disjoint spanning trees to minimize the number of critical nodes which will maximize the fault-coverage of the network.

5.2 Computational Results for Fault-Coverage

In this section we present computational results to determine an experimental bound on fault-coverage of two edge-disjoint spanning trees. More specifically, first we generate random networks of various sizes and connectivity, and then construct two edge-disjoint spanning trees on these networks and then compute the *fault-coverage.*

Four connectivity property in a network is sufficient for existence of two edge-disjoint spanning trees [5, 14, 12]. Thus, we use a well known result due to Bollobás [3] to ensure the connectivity of the random networks used in this section. We present two algorithms for constructing two edge-disjoint spanning trees on the random networks.

The first algorithm builds a depth first search (DFS) tree, and then removes it from the graph, and then builds a breadth first search (BFS) tree. We denote the trees of the first algorithm by DFS & BFS. The second algorithm constructs two BFS-like trees (denoted by BFS & BFS) for an heuristic minimization of the critical nodes by *maximizing the number of leaves.* Our objective is to compare these two algorithms as a function of network size and network connectivity, and obtain computational results on their fault-coverage. In Table 1, the ratio of the critical nodes over the total number of nodes is shown as a function of various network sizes for both algorithms. Each row of this table summarizes the average values obtained over 500 random networks of the same size and with connectivity 4. In order to determine the statistical value of these averages, the standard deviation has been computed. Evidently, the second algorithm, BFS & BFS, which induces more leaves on each tree, leads to much better fault-coverage.

In Table 2, the comparison is made as a function of the minimum degree, which is also, with very high probability, the connectivity [3]. Thus, the network size is kept constant 50 nodes while the connectivity is increased. It is shown that as the connectivity increases the fraction of critical nodes decreases and the fault-coverage increases (i.e., the probability of both trees being affected by the same node failure goes to zero).

5.3 Maintenance of Complete Host-to-Host Communication

In this section we assume that the faulty node is a leaf in at least one of the two spanning trees (i.e., cases 1 and 2 in Section 5.1), and present a fault-tolerant protocol that maintains continuous host-to-host communication between all non-faulty nodes. In the fault model considered here it is possible to tolerate multiple node failures, provided that after each node failure there exist a non-faulty tree - a spanning tree of all active nodes. (Note that this is the case if the faulty nodes are leaves in the *same* non-faulty tree). However, for simplicity we consider a single node failure and proceed with the following example.

Suppose node G, in Figure 10, which is a leaf node on the first spanning tree (i.e., $Tree_0$), becomes faulty. Since G has only one link on this tree, the failure of G is equivalent to the failure of link (C, G). Hence, upon detecting this failure, node C prunes $Tree_0$ to obtain $Tree_0'$ by executing the cycle-closure procedure. Consequently, a new virtual ring—that contains all the active nodes—is constructed, and complete host-to-host communication is ensured among the active nodes over the pruned tree, $Tree_0'$.

However, the impact of this failure on the TER1 is more drastic since the spanning tree $Tree_1$ is broken to three components (as a result of the removal of all the ring edges incident to G). In this case, each non-faulty neighbor of G on TER1 executes the cycle-closure algorithm, thus yielding to multiple subrings on which partial host-to-host communication can continue. However, the status of the remote traffic on TER1 (i.e., the packets whose destination is on another subring) needs to be addressed. There are three possible methods: (1) to discard those packets, (2) to send these packets to upper layers to be stored temporarily, and then use TER0 to route them to their destination, and (3) to try to route those packets to TER0 by using **jump** and **short-cut** operations.

Algorithm Global Host-to-Host
Let U be a faulty leaf node on TER0 and let node V be the neighbor of U on TER0.
Upon detecting the failure of U, V performs the following operations:

1. execute cycle-closure on TER0 to prune U from the underlying $Tree_0$
2. send a broadcast message on the pruned tree to inform all the nodes that U is down

Each active node W upon receiving such a broadcast message performs the following operations:

- **On the TER0:** (i) W sends (and accepts) no traffic to U, and (ii) W does not route any traffic to TER1 from TER0
- **On the broken TER1 (with multiple components):** (i) if W is adjacent to U on $Tree_1$, then it performs a cycle-closure, and (ii) W maintains partial host-to-host communication on its connected component on TER1, as was shown in Section 4.1.

Figure 11: Maintenance of the Complete Host-to-Host Communication

In Figure 5.3 we summarize the simple protocol for ensuring continuous global host-to-host communication in the presence of a failure.

6 Conclusions

This work describes modifications and algorithmic enhancements needed in the network interface of the MetaNet architecture in order to provide fault-tolerant convergence routing. Our fault-tolerant solution has the property that the packet header need not be recomputed after reconfiguration. In other words, the actual destination address that is used inside the network for routing remains the same, regardless of the network structure. (Most fast packet switch architectures do not have this property.)

The objective of the fault-tolerant protocols is to maximize the availability of host-to-host communication, in the case of failures. First, a protocol for maintaining partial host-to-host communications was presented. This protocol was based on a new mechanism called cycle closure (i.e., local isolation of failures in order to minimize the network's down time).

Next, in order to maintain continuous host-to-host communication among all active nodes, after a link or a node failure, a protocol based on two edge-disjoint spanning trees, for redundancy, was proposed. In this technique, the host-to-host communication will continue after any single link failure, and after any single node failure, which is not a *critical node* (i.e., not an internal node in both spanning trees). Simulation of the heuristics, to minimize the number of critical nodes, shows that the ratio between the number of critical nodes to the total number of nodes can be very small, which implies high fault coverage with this technique.

References

[1] Y. Afek, S. Kutten, and M. Yung. Memory efficient self stabilizing protocols for general networks. In *The 4th International Workshop on Distributed Algorithms*, pages 15–28. Spriger-Verlag, Lecture Notes in Computer Science, No. 486, September 1990.

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

[3] B. Bollabás and A. Thomason. Random graphs of small order. *Annal. of Discrete Mathematics*, 28:47–97, 1985.

[4] 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.

[5] J. Edmonds. Edge disjoint branchings. *in Combinatorial Algorithms (R.Rustin, Ed)*, pages 91–96, 1972.

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

[7] 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.

[8] 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.

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

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

[11] R. Perlman. An algorithm for distributed computation of a spanning tree in an extended LAN. In *9th Data Commun. Symp., Whistler Mountain, British Columbia*, pages 44–53, September 1985.

[12] Y. Shiloach. Edge-disjoint branching in directed multigraphs. *Information Processing Letters*, 8:24–27, 1979.

[13] D. P. Siewiorek. Architecture of fault-tolerant computers. *IEEE Computer Magazine*, 17(8):9–18, 1984.

[14] R.E. Tarjan. A good algorithm for edge-disjoint branching. *Information Processing Letters*, 3:51–53, 1974.

[15] B. Yener, Y. Ofek, and M. Yung. Topological design of loss-free switch-based lans. *Submitted for publication*, 1994.

		Average % of the Critical Nodes		Standard Deviation	
# of Nodes	# of Edges	DFS & BFS	BFS & BFS	DFS & BFS	BFS & BFS
12	47.26	27.6%	8.0%	1.17	0.87
14	55.69	27.8%	8.2%	1.21	0.90
18	73.33	27.3%	8.7%	1.38	1.11
20	81.30	27.6%	9.3%	1.44	1.13
22	91.37	27.8%	9.2%	1.49	1.23
24	100.06	27.9%	9.2%	1.56	1.23
26	106.89	27.7%	9.2%	1.75	1.30
30	128.89	26.7%	8.9%	1.82	1.34
40	178.88	26.7%	9.3%	2.10	1.63
50	226.93	26.6%	9.1%	2.44	1.90
60	281.21	26.2%	9.3%	2.60	1.98
70	332.22	25.9%	9.1%	2.87	2.14
80	387.75	25.6%	9.0%	3.05	2.26
90	442.84	25.3%	8.7%	3.18	2.44
100	507.23	24.7%	8.4%	3.44	2.57

Table 1: Fault-coverage comparison for minimum degree 4 and various network sizes. The ratio is between the number of critical nodes and total number of nodes.

		% of the Critical Nodes		Standard Deviation	
Min. Degree	# of Edges	DFS & BFS	BFS & BFS	DFS & BFS	BFS & BFS
4	226.93	26.6%	9.1%	2.44	1.90
6	303.17	22.2%	6.5%	2.00	1.50
8	385.10	19.2%	4.6%	1.88	1.34
10	468.86	16.9%	3.4%	1.59	1.19
12	558.16	15.2%	2.6%	1.55	1.04
14	642.28	13.7%	2.0%	1.48	0.95
16	745.82	12.4%	1.6%	1.39	0.83
18	846.28	11.3%	1.3%	1.23	0.76
20	945.13	10.5%	1.1%	1.11	0.69
22	1061.86	9.7%	0.95%	1.15	0.65
24	1179.44	9.1%	0.77%	1.02	0.61
26	1316.18	8.5%	0.72%	0.97	0.58
28	1447.18	7.8%	0.66%	0.93	0.54
30	1598.04	7.3%	0.57%	0.86	0.53

Table 2: Comparison of the algorithms as a function of expected connectivity (min degree) for 50 node networks