A Resilient Communication Structure for Local Area Networks

Amr El Abbadi*

Thomas Räuchle

TR 84-653

December 1984

(Revised April 1985)

Department of Computer Science
Cornell University
Ithaca, New York 14853

* This author was supported by NSF Grant No. MCS82-10356.

A RESILIENT COMMUNICATION STRUCTURE FOR LOCAL AREA NETWORKS

Amr El Abbadi*, Thomas Räuchle

Department of Computer Science

Cornell University, Ithaca, New York, 14853

(607) 256 8597

ABSTRACT

Reliable communication is crucial to the correct functioning of distributed systems. We propose a multi-ring communication structure and a reconfiguration algorithm that tolerate multiple link failures before the network divides into more than one partition. In case of partitioning, each partition is reconfigured to allow communication among the sites within the partition. The algorithm handles recovery of links and merges partitions once links become operational again. The algorithm itself is fault-tolerant, and it is fully distributed and does not require global knowledge about the status of the network at any one site.

*This author was supported by NSF Grant No. MCS82-10356.

1. Introduction

Reliable communication is crucial in a distributed system. For local area networks, various communication topologies that exhibit different degrees of reliability have been proposed. Broadcast networks such as Ethernets [Met76] and token rings [Far72] are the most prominent local area topologies. The reliability of such networks depends on their ability to mask failures.

Failures range from noise in the communication medium, to interface failures, to failures of segments of the communication system. There are standard techniques for error detection and correction to overcome the problem of noisy communication channels. These techniques are independent of the communication topology. In Ethernet like networks, failures of the interface only affect the host connected to that interface, while the failure of a segment of the cable causes the entire communication network to stop functioning. In ring networks, the failure of a segment of the medium restricts communication to one-way communication between a subset of the sites in the ring; halted interfaces also lead to such ring discontinuity. Multiple failures result in partitioning of the network into subnetworks called *partitions*. The sites in each partition can communicate with each other but not with sites in another partition.

Various approaches to increase the reliability of communication systems have been proposed. Replicating ethers has been proposed to increase reliability of broadcast networks [Bab84]. In [Wol79], reliability is increased by adding one additional communication ring between sites. Schröck [Sch84] extends this technique by imposing a logical loop structure on top of two rings.

In this paper, we present a communication structure and a reconfiguration algorithm that achieves high fault-tolerance and handles partitioning and merging of partitions. Our approach is based on the fiber optical loop proposed by Schröck[Sch84]. It tolerates failures of communication segments and interfaces, and guarantees communication between each pair of sites that is connected through a functional communication path.

The sites in the system are ordered as a ring. Each neighboring pair of sites (s_i, s_j) is connected by k links. The reconfiguration algorithm tolerates $2k - 1$ link failures in the worst case before the network partitions into subnets. In case of partitioning, each partition is reconfigured and again tolerates up to $2k - 1$ failures before further partitioning occurs. The algorithm also handles the recovery of links and ensures that partitions are merged correctly.

The algorithm does not require global knowledge about the network and is itself tolerant to failures during its execution. Furthermore, extensions of the network do not require changes to the algorithm.

The paper is organized as follows. In Section 2 we describe the basic communication structure and an algorithm that tolerates one link failure before partitioning occurs. The algorithm is then extended to handle link recovery and partition merging. Section 3 describes an extension to the communication structure and an algorithm to tolerate multiple link failures before partitioning. Section 4 concludes the paper with a summary of the results.

2. The Basic Double-Ring Approach

In this section we present the physical and logical topology of the communication network and then give a reconfiguration algorithm that tolerates one link failure before partitioning occurs.

2.1. The Communication System

Sites access the communication network by connecting to a *switch*. The switches are ordered in a circle and each pair of neighboring switches is connected through a bidirectional *link*. A link consists of two unidirectional *edges*, directed opposite to each other. Thus, the physical communication system forms two unidirectional rings, arranged as depicted in Figure 1a. Messages on the inner ring, called the *receiving ring*, flow in the direction opposite to messages on the outer ring, called the *sending ring*. Message transmission time on a non-faulty edge is bounded by a constant T; hence the time to send a message around a ring is bounded by nT, where n is the number of segments in the ring.

Logically, the network is configured into a *loop* as depicted in Figure 1b. One of the switches is designated as *head*, the switch immediately preceding the head on the sending ring is designated the *tail*. The head generates a continuous stream of empty message frames on the sending ring. To send a message, a site acquires an empty frame on the sending ring, inserts the data and destination address into the frame, and passes the frame on to the next switch on the sending ring. The access protocol is similar to the slotted ring protocol in [Pie72]. Messages on the sending ring are relayed until they reach the tail. The tail receives messages on the sending ring and sends them back on the receiving ring. A message is received by its destination on the receiving ring. Note that the link between the head and the tail is not used for communication. This redundancy

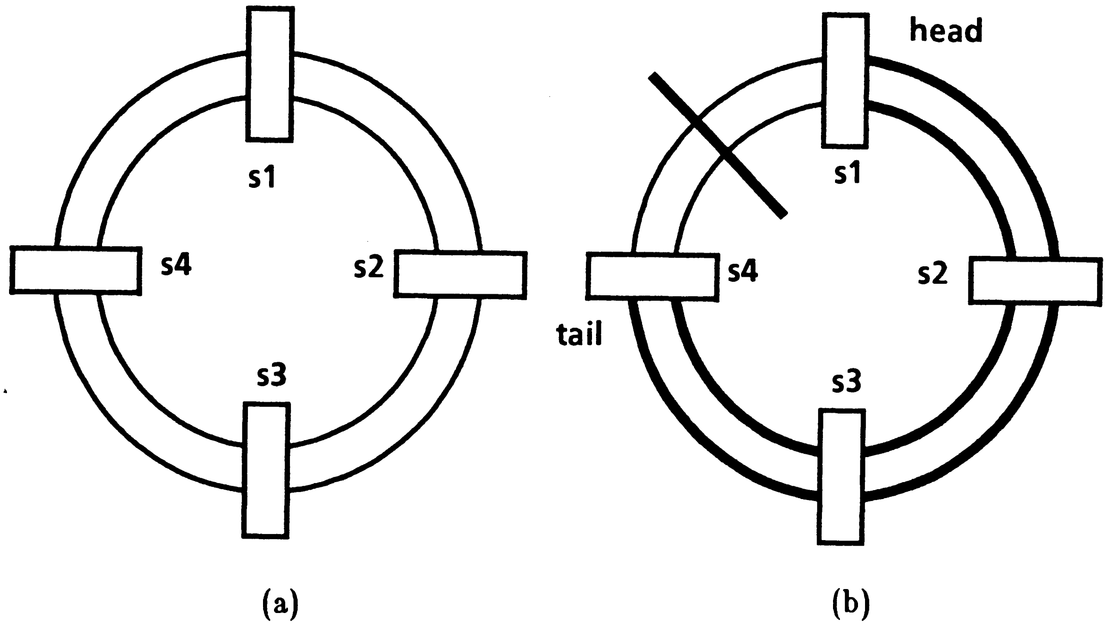


Figure 1: Physical and Logical Configuration

allows for link failures to be tolerated as described below.

2.2. The Algorithm

Edges fail by not delivering any messages, losing some messages, or delivering messages too slowly. These failures cause the message stream to be interrupted; thus edge failures can be detected. If the message stream on an edge is interrupted for more than T units of time, the switch at the receiving end of the edge detects the failure. The switch then changes to *failure mode* and executes the reconfiguration algorithm. The algorithm is based on an algorithm given in [Sch84].

In failure mode, a switch s_i first propagates the fact that a failure has been detected. It does so by sending a stream of failure messages, denoted m_f, on both its links.[1] A timer is started when the first failure message is sent. A failure message takes at most T time units to reach a neighboring switch. If this neighbor is not already in failure mode, it will change to failure mode and start sending its first failure message, which also takes at most T time units to reach back to s_i. Thus, if s_i does not receive m_f messages on a link by $2T$ time units after it sent on the link, it declares the link faulty and stops sending over it. Note that a link is faulty if at least one of its edges is faulty. If a link fails during the execution of the protocol, the stream of m_f messages is interrupted. The switch at the incoming end declares that link failed. After faulty links have been detected, switches assume the role of head or tail as follows. If the link that contains the incoming sending edge is faulty, the switch assumes the role of head, if the link with the incoming receiving edge is faulty, it assumes the role of a tail. Each head then starts transmitting empty message frames on the outgoing sending edge. Upon receiving empty message frames on the sending edge, other switches enter normal mode of operation again.

Note that in case of simultaneous link failures, the network will be partitioned into several subnetworks. The algorithm ensures that each subnetwork is configured into a loop; hence communication within each partition is guaranteed. A shortcoming of the algorithm is that it does not handle the case of links recovering or the case of partitions merging. This implies that upon startup of the network, the loop structure has to be established through external means. In the next section we give an extension of the algorithm that handles link recovery and partition merge.

2.3. Link Recovery and Merging of Partitions

To handle the recovery of links, it is necessary to detect when links become functional again. This is accomplished through the following protocol, executed periodically by the tail switches in each partition. The tail switch sends out a probe message, denoted m_p, on the outgoing edge of the faulty link.[2] Whenever a head switch receives a message m_p on a "faulty link", it sends the message back on the other edge of the same link. If the originating station receives m_p back, the link has recovered and the merge protocol can be initiated.

---

[1] Sending on a link means sending on the outgoing edge of the link.

[2] Note that a link is declared faulty once one of its edges is faulty, so sending on a faulty link might be possible.

To merge partitions, the same reconfiguration protocol as for failures is executed, i.e., failure and recovery of links are treated uniformly as "change of configuration." Once a switch detects recovery of a link, it starts sending m_f messages and thus forces its neighbors into failure mode and the reconfiguration algorithm is executed as described above.

This algorithm however fails to establish a head and a tail when all links in the network are operational. In this case, all switches in the network will remain in a state in which they continuously send and receive failure messages on both links. None of the switches has an adjacent faulty link, thus no switch will assume the role of head or tail, and the system will livelock.

To detect this state, a predetermined switch, say s_1, keeps track of the time after it sent the first m_f message. After receiving m_f messages for $2NT$ time units, s_1 will stop sending failure messages and assume the function of head. Consequently, the stream of m_f messages for its neighbor s_n becomes interrupted and s_n will assume the role of the tail. To guarantee correctness of the algorithm, N must be at least n, the number of switches in the network. Choosing $N > n$ allows for extending the network without reprogramming s_1.

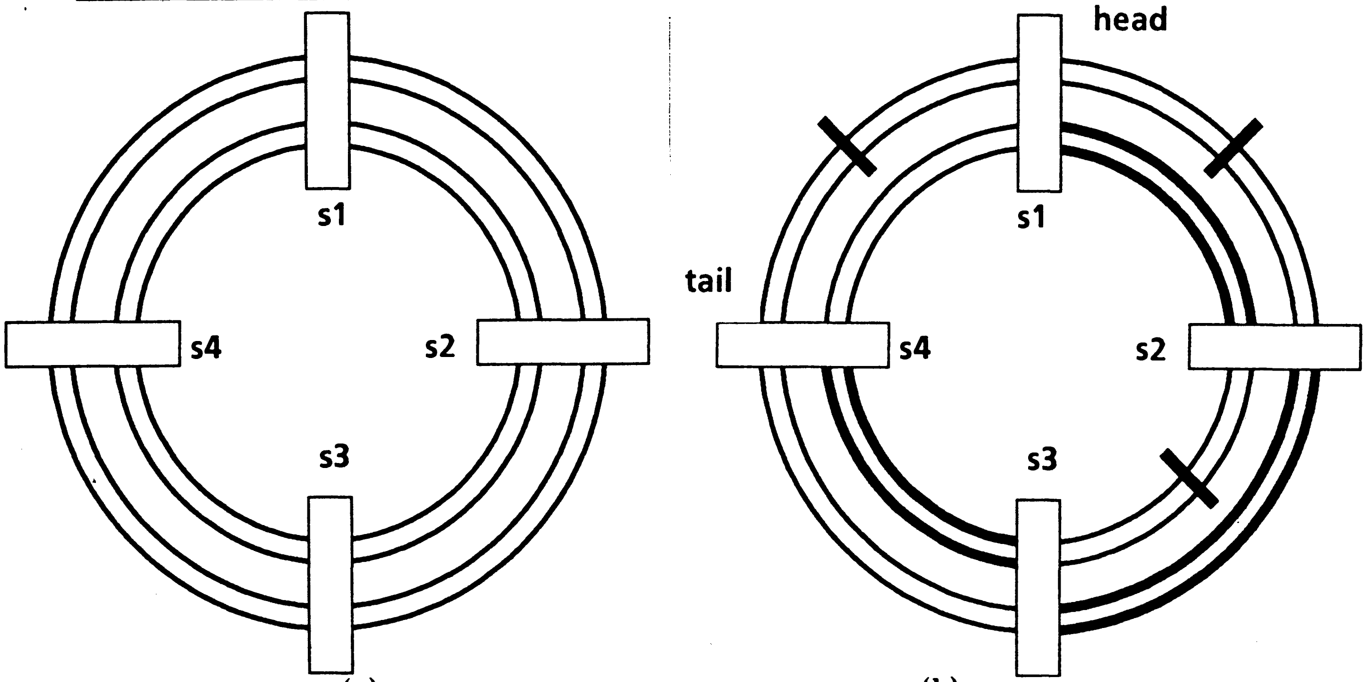
As a consequence of the above modification, the network can be started with n partitions, each containing a single switch in failure mode that sends m_f on both links. Upon termination of the algorithm, the network will be configured in a loop with head s_1 and tail s_n.

The correctness of the algorithm follows from Lemmata 2 and 3 in Section 3.

3. A Multi-Ring Structure that Tolerates Multiple Failures

In this section we present a reliable communication system, the *multi-ring*. In the multi-ring, each pair of neighboring switches in the circle is connected through k links. The communication system consists of k pairs of rings, where the rings in each pair are directed opposite to each other. Figure 2 shows a communication system with 2 pairs of rings. The introduction of additional pairs of communication rings to connect the switches increases redundancy and allows for more fault-tolerance. We extend the previous algorithm to reconfigure a network with k pairs of rings into a single loop. In the worst case, the algorithm tolerates $2k - 1$ link failures before the system is divided into two partitions.

In normal mode of operation, there is a *current loop* that is used for message transmission. In the example in Figure 2b the current loop is depicted by bold



(a) (b)

Figure 2: Multi-Ring Network

edges. As in the simple case, the head generates a continuous stream of message frames on the sending ring of the current loop. The tail receives messages on the sending ring and sends them back on the receiving ring.

Each switch s_i maintains the two sets in_i and out_i. For each i, in_i contains all links l_j, $1 \leq j \leq k$ adjacent to s_i, such that l_j is non-faulty and its sending edge is directed towards s_i. Out_i contains all links l_j, $1 \leq j \leq k$, such that l_j is non-faulty and its sending edge is directed away from s_i. After s_i detects a change in configuration (failure or recovery of a link), it changes to failure mode and starts sending failure messages m_f on all links in $in_i \bigcup out_i$. This causes its neighbors to change to failure mode and send failure messages. A link is deleted from in_i or out_i whenever s_i detects that the link failed. The switch then stops sending on this link. A link is added to the appropriate set when s_i detects that it is operational again. If links fail during the execution of the algorithm, the appropriate in and out sets reflect this change. If no failures occur for a sufficiently long time, in_i contains the same links as $out_{(i-1) mod\ n}$ for all i.

Failure messages are being sent until one of the two following conditions occurs. Either, there exists a switch s_j with $in_j = \phi$, or, the *in* sets of all switches are non-empty. In the first case, s_j assumes the role of a head. It follows from the observation above, that $out_{(j-1) mod\ n} = \phi$, and hence $s_{(j-1) mod\ n}$ will assume the role of a tail. In the second case, a predetermined switch, say s_1, assumes the role of the head after receiving failure messages for $2NT$ time units. It stops sending on its *in* set, and hence s_n will assume the role of the tail. Note that if the system is partitioned, each partition will contain a head and a tail.

Once a head has been determined, it starts sending empty message frames on the innermost link of its *out* set. A site receives a message over a link from its *in* set, and forwards it on the innermost link of its *out* set. The tail bounces messages back on the innermost link of its *in* set.

The algorithm tolerates $2k - 1$ link failures in the worst case before partitioning occurs. Failure of any switch corresponds to the failure of k links. In particular, the correctness of the algorithm does not depend on s_1 being operational.

3.1. Proof of Correctness

In this section we first prove that the network will in the worst case tolerate $2k - 1$ link failures before any partitioning occurs. We then prove that each partition contains exactly one head and one tail.

The following lemma shows that the network tolerates $2k-1$ failures.

Lemma 1:

If the number of link failures in the network is less than $2k$, then the algorithm terminates with a loop that connects all switches.

Proof:

Assume that two partitions have formed with fewer than $2k$ failures such that two sites s_i and s_j cannot communicate. There must be two sites s_p and s_q, $p \neq q$, such that any message sent between s_i and s_j must pass through s_p or s_q and $in_p = in_q = \phi$. A set in_i is empty, if and only if, k links have failed. Thus, there must have been at least $2k$ failures; a contradiction. □

Lemmata 2 and 3 prove that upon termination of the algorithm, there is exactly one head and one tail in each partition. Note that a fully functional network forms a partition containing all switches.

Lemma 2:

Upon termination of the reconfiguration algorithm there is at least one head and at least one tail in each partition.

Proof:

We give the proof for the tail, the proof for the head is analogous. Assume there exists some partition that contains no tail. First consider the case where there is no switch with an empty *out* set. This implies that failure messages m_f are being sent continuously to all switches. But according to the algorithm, after $2NT$ time units of receiving m_f messages on a receiving edge, s_1 assumes the role of a head and does not send messages on its *in* set. Hence $out_n = \phi$ and s_n will be tail. Secondly, consider the case where there is at least one switch with an empty *out* set. This switch will become a tail $2T$ time units after its *out* set became empty. In both cases we obtain a contradiction to the assumption that there is no tail in a partition. Therefore each partition contains at least one tail. □

Lemma 3:

Upon termination of the reconfiguration algorithm, there is at most one head and at most one tail in each partition.

Proof:

We prove that there is at most one head, the proof for the tail is analogous. For the sake of contradiction, assume there is a partition that contains more than one head. First, consider the case where all *in* sets are non-empty. In this case, all switches are continuously receiving m_f messages. After $2NT$ time units, only s_1 will decide on being head. This contradicts the assumption that there is more than one head in a partition.

Secondly, consider the case where the system has suffered from link failures, and hence may be partitioned. Let $P = (s_i, s_{i+1}, \cdots, s_l)$ be the smallest partition with more than one head where s_i is a head and s_l is a tail. Assume there is at least one switch s_j, $i < j \leq l$ such that $in_j = \phi$, i.e., s_j is also a head. According to the algorithm, a switch that does not receive m_f on its in set stops sending m_f on these links and becomes a head. Since $in_j = out_{(j-1) \bmod n}$, the switch $s_{(j-1) \bmod n}$ becomes a tail. Therefore, s_j is not in the same partition as s_i, i.e., P consists of subpartitions P_1 and P_2. P_1 and P_2 either contain at most one head or they contain more than one head. In the first case, there is no partition with more than one head, in the second case there is a partition smaller than P that contains two heads. In both cases we obtain a contradiction to P being minimal with more than one head.

Thus, there is at most one head in each partition. □

4. Conclusions

We presented a new fault-tolerant communication structure, the multi-ring and a fault-tolerant reconfiguration algorithm. The algorithm reconfigures a network connected through k double rings to allow communication between all sites. It tolerates $2k - 1$ link failures in the worst case before partitioning occurs. If more than $2k - 1$ failures cause the network to partition, the algorithm ensures communication within each partition. In terms of network connectivity, this compares favorably with the resilient broadcast network proposed in [Bab84], which tolerates $k - 1$ link breakdowns in the worst case before partitioning. The kinds of failures our system tolerates are interface failures, link breakdowns, lossy links and slow links.

After adding the ability to detect link recovery, we achieved a uniform treatment of link failures and recovery. Unlike the approach presented in [Wol79], our algorithm treats all failures uniformly; furthermore, it merges partitions once links become operational again. Consequently, the network can now be started with each switch in a singleton partition. After termination of the algorithm, communication will be possible between all sites in the network.

The logic required in the switches is very simple and can thus be made very reliable without high cost. The algorithm is fully distributed and depends only on the ability to communicate between neighboring switches in the network. Thus, there is no single central switch on whose reliability the reliability of the network depends.

5. Acknowledgements

We would like to thank Thomas Joseph and Özalp Babaoğlu for their comments on an earlier version of this paper.

6. References

[Ake84]

S. B. Akers, B. Krishnamurthy, "Group Graphs as Interconnection Networks", *GE Technical Information Series 84crd156,* July 1984

[Bab84]

Ö. Babaoğlu, R. Drummond, "Communication Architectures for Fast Reliable Broadcasts", *Proceedings 4th Symposium on Fault-Tolerant Distributed Computing,* Silver Spring, MD, 1984

[Far72]

D. J. Farber, K.C. Larson, "The System Architecture of the Distributed Computer System - the Communications System", *Symp. on Computer Networks,* Polytech. Institute of Brooklyn, 1972

[Lam84]

L. Lamport, "Using Time Instead of Timeout for Fault-Tolerant Distributed Systems", *ACM Trans. on Prog. Languages and Systems, 6, 2,* April 1984

[Met76]

R. M. Metcalfe, D. R. Boggs, "ETHERNET Distributed Packet Switching for Local Computer Networks", *CACM, Vol 19,* 1976

[Pie72]

J. R. Pierce, "Network for Block System of Data", *Bell System Technical Journal, Vol. 51, No. 6,* July - August 1972

[Sch84]

W. Schröck, "A Fault-Tolerant Loop-Structured Local Area Network on the Base of Fibre-Optical Transmission", *Proceedings 2.GI/NTG/GMD Conference on Fault-Tolerant Computing Systems,* Bonn, September 1984

[Wol79]

J. J. Wolf, et al, "Design of a Distributed Fault-Tolerant Loop Network", *Proceedings 9th IEEE Fault-Tolerant Computing Symposium,* Madison, WI, 1979