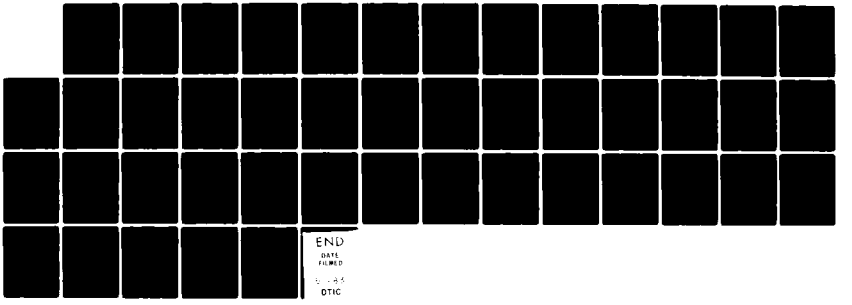
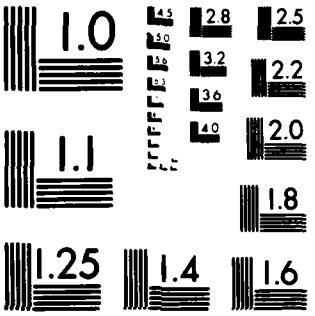
AD-A127 667 NETWORK MANAGEMENT RESEARCH(U) CALIFORNIA UNIV BERKELEY 1/1
ELECTRONICS RESEARCH LAB C V RAMAMOORTHY 14 MAR 83
ARO-16984.2-EL-A DAAG29-79-C-0171

UNCLASSIFIED F/G 9/2 NL





ARO 16984.2-EL-A

AD A127 ...

NETWORK MANAGEMENT RESEARCH

FINAL REPORT

C. V. Ramamoorthy

August 14, 1982 - March 14, 1983

U. S. ARMY RESEARCH OFFICE

DAAG29-79-C-0171

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

DTIC FILE COPY

APPROVED FOR PUBLIC RELEASE;
DISTRIBUTION UNLIMITED.

83 05 03 025

The views, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

SECURITY CLASSIFICATION OF THIS PAGE (When Date Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A127667	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Network Management Research		5. TYPE OF REPORT & PERIOD COVERED Final: 8/14/82 - 3/14/83
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) C. V. Ramamoorthy		8. CONTRACT OR GRANT NUMBER(s) DAAG29-79-C-0171
9. PERFORMING ORGANIZATION NAME AND ADDRESS Electronics Research Laboratory University of California Berkeley, CA 94720		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS U. S. Army Research Office P.O. Box 12211 Research Triangle Park, NC 27709		12. REPORT DATE
		13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS (If different from Controlling Office)		15. SECURITY CLASS. (of this report)
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

NA

18. SUPPLEMENTARY NOTES

The view, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

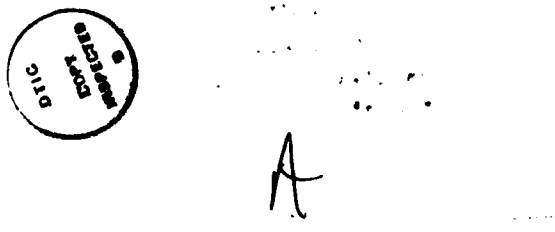
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Date Entered)

ABSTRACT

This report deals with important issues which concern functional components that are present in a distributed processing system for control purposes. These components are *global information management*, *dynamic reconfiguration*, *deadlock handling*, and the communication-oriented components of *protocol*, and *routing*. In the management of global information, the ensuring of consistency and availability are crucial, and unsolved problems exist in integrating techniques for providing these attributes in a viable manner. Dynamic reconfiguration, necessary for providing load-balancing and reliability, poses the problems of design of the state determination mechanism, the derivation of conditions for reconfigurability and the design of reconfiguration strategies. Correct and efficient deadlock detection algorithms need to be developed to resolve deadlocks due to concurrent usage of resources. In the area of communication protocols, efficient protocol analysis techniques and tools for protocol specification and design are needed to ensure the correctness of each class of protocols. In the design of routing algorithms, distributed routing algorithms are usually preferred, but due to the nature of parallel processing, they also give rise to some difficulties. Hierarchical routing algorithms are more suitable for very large networks; however, little work has been done in this area.



A

A

1. Introduction

A distributed processing system (DPS) is a collection of autonomous processing elements (PE's) connected by a communication subnet, which provides real-time data distribution facilities between the PE's. In the following discussion, we will refer to PE's and processors in the communication subnet as *nodes*.

In this report, we consider some important functional components which must be present in order to control a DPS, namely, *global information management*, *dynamic reconfiguration*, *deadlock handling*, and the components associated with communication subnets, *protocol* and *routing*. We are especially interested in the design issues concerning these components for the case of *large* systems, where centralized solutions have obvious disadvantages. Our functional approach to partitioning the DPS is in contrast to the ISO model, in which the system is partitioned into a set of seven logical layers, each layer being built using the facilities provided by layers below and in turn providing facilities to the layers above. For our purposes, the ISO model is too communication-oriented to provide a basis for discussion of the functional components mentioned above. Moreover, many issues we will be discussing will be applicable to all or most layers of the model.

To coordinate all the functions of a distributed Processing system(DPS), certain control information should be shared and communicated among members. The requirements for dealing with such *global* information are especially severe in a distributed environment. The partition of information for storage, the currency and the correctness of the information, the frequency and methods of update are all significant issues to be addressed.

The reconfigurability of a distributed system is important for its ability to enhance the fault tolerance of a system by tailoring the system configuration in response to failures. The steps involved in dynamic reconfiguration are fault

detection, fault location, reconfiguration and recovery. Techniques for carrying out these steps are discussed.

In the distributed environment, system deadlock could cause severe degradation in performance and poor utilization of resources. We will review some of the current approaches in attacking this problem.

Since the correct operation of communication protocols strongly affects the reliability and performance of distributed processing systems, correctness of the design and implementation of protocols has to be ensured in order to provide reliable communication. To validate correctness of protocols, specification methods and formal models of protocols are studied, and an implementation model for protocols is presented. Further, with the increasing interests in office automation and broadcast capability of long haul networks, we will also point out issues on multi-party communication.

Due to the possible existence of several alternative paths between PE's and sizable amount of traffic in the distributed system, reliable and adaptive routing techniques become more important in order to provide high throughput and fast response time. Because of our interest in large systems, we will deal in this report distributed and hierarchical routing algorithms. Distributed routing algorithms have received a lot of attention recently as the centralized schemes are too vulnerable. However, due to the nature of parallel processing, distributed routing algorithms are more difficult to design. Previous work has concentrated almost exclusively on the minimization of the delay of individual information packets, and other objectives are usually ignored. Hierarchical routing algorithms will become more and more important as larger and larger networks are built in the future. The question of how to update information in such hierarchical networks is a very important issue.

2. Global Information Management

A key function of a distributed system is to allow nodes to share information. We call such shared information *global*. Global information may be used for control purposes e.g. node status tables, routing tables, etc. or it may be used by multiple users e.g. database files. In the following discussion we will often omit the adjective *global* for brevity.

In order to satisfy the requirements imposed by the entities using the information, we will require techniques for guaranteeing various desirable attributes, namely *rapid accessibility*, *security*, *integrity*, *availability* and *consistency*.

Availability and consistency are related issues which arise out of the occurrence of failures and the concurrency present in the distributed system respectively and together they constitute a major part of the problem of managing global information. Below we discuss the problems and the corresponding solution techniques concerned with ensuring the availability and consistency of global information.

2.1. Availability

The availability of a system is defined as the probability that the system will be functioning correctly at any time during its scheduled working period. We can extend this definition and say that the availability of a piece of global information is the probability that an authorized entity that wishes to read or update it is able to do so. We also include the proviso that the information read should not be out-of-date or corrupted as a result of prior failures.

The difficulty of ensuring the availability of information depends on the kind and degree of failure that must be prevented from making the information unavailable or unusable. Failures may occur at nodes or links. Nodes may crash, be jammed or destroyed. In these cases, we assume that after the instant of failure,

the failed node does not send out further messages till it recovers or only sends out meaningless streams of bits which are easily detected as emanating from a failed node. But nodes may fail in such a way that they send out incorrect messages but these messages are not easily detected to be incorrect. In such cases the incorrect information obtained from these messages may spread throughout the distributed system. Such failures may occur if some node(s) are taken over by malicious agents or if a node sends out information whose incorrectness is not detected by local checking software or hardware. We refer to failed nodes which are able to mislead correctly-operating nodes in this manner as *malfunctioning* nodes. In addition to node failures, failures in the communication subnet may result in messages not being transmitted or received in time.

Given the set of nodes and the communication subnet through which they communicate with each other, we require copies of information to be stored at multiple nodes to prevent loss of availability if some copies become unavailable due to node or communication failures. Another scheme would be to split the information into a number of parts and store the parts at different nodes with error-correcting information so that the information could be reconstructed even if some part(s) become unavailable. But this scheme does not have general applicability because the information may have to be reconstructed before any processing can be done on it, resulting in heavy communication costs. However, error-correcting codes can and should be used e.g. to ensure the correctness of each copy, or for correcting errors arising in communication.

From the viewpoint of availability, the following protocols used in managing the replicated global information are relevant:

(a) *read and update access protocols*

Let us assume that transactions are being processed serially, i.e. there is no concurrency in the system, for the time being. Further, let us assume that

no malfunctions occur. Then a number of protocols exist for performing the read and update operations. The selection of the protocols to be used in a design will affect the availability of the information for read and update operations. For example, the read protocol may involve reading any single copy and the update protocol may involve updating all copies. Alternatively, we could read x copies and update y copies where $(x+y)$ is greater than the total number of copies, using a timestamping mechanism to determine the most up-to-date value if some of the x values read were not coincident. Yet another alternative would be to direct all reads and updates to a single copy,called the *primary*, which is responsible for distributing updates to all the other copies. These protocols ensure different degrees of availability for read and update operations. as well as other attributes such as response time, communication costs, etc. Thus, in a distributed system design, the choice of the number of copies of the different pieces of information (files,tables, etc.) and the read and update protocols to be used should be made in the light of the requirements on the values of the above attributes. Unfortunately, the related optimization problem, called the file allocation problem, becomes intractable if formulated in a way which realistically models all the parameters involved [ROT 77]. For example, work on the file allocation problem usually assumes pre-selected read and update protocols in determining the degree of replication of the various files(e.g. [MAH 76]).

Suppose we also wish to guard against malfunctions. If $(2x+1)$ copies exist, then by reading all copies and computing the majority value, we can protect against upto x malfunctioning nodes. However, if more than x malfunctioning nodes exist, then two nodes trying to read this information, may end up with different majority values, because a malfunctioning node may give out different values to different read requests for the same information. This gives rise to the following possibility of spread of error. Consider two pieces of information, *d1* and *d2*. Suppose that a majority of the nodes which hold copies of *d1* are

malfunctioning but that the same is not true for *d2*. Then if a transaction which reads *d1* and updates *d2* is processed, the values of *d2* held by correctly operating nodes may diverge. In solving this problem, recently published work on the Byzantine Generals Agreement problem is relevant [LAM 80, DOL 82a, DOL 82b].

Consider a node T which wishes to transmit a value to a set of receiving nodes {R}. Then, the Byzantine Generals Agreement is reached among the nodes in {R} if the following conditions are fulfilled:

1. If the transmitter operates correctly, all receivers arrive at the value *v*.

2. All receivers arrive at the same value, whether or not the transmitter is malfunctioning.

To show the nature of this agreement, we show an example of a completely-connected network of four nodes in Fig. 1. Assume that the transmitting node (marked T in the figure) is required to transmit the value 5 to the receiving nodes (marked R in the figure). We show two possible scenarios, in the left and right parts of the figure respectively, the first involving a malfunction in a receiver and the second in the transmitter itself. We also show how a algorithm given in [LAM 80] is applicable to this network to reach the Byzantine Generals Agreement, assuming (as is true for the two scenarios described above) that there is at most one malfunctioning node in the network. The algorithm requires two phases for our example, under the assumption made above. In the first phase, the transmitter sends its value (5) to all the receivers. Observe that in the second scenario, the transmitter does this incorrectly. In the second phase, each receiver sends the value received in the first phase to all receivers, including itself. Observe that in the first scenario one receiver executes this phase incorrectly. After this phase, each receiver computes the *median* of the values received in the second phase. A quick look at Fig. 1 will verify that all correctly operating receivers arrive at the same value, 5 in the first scenario, 2 in the

second. The conditions of the Byzantine Generals Agreement are fulfilled in both scenarios. If more malfunctioning nodes are to be tolerated, more phases will be required [LAM 80].

Algorithms of polynomial complexity which reach this agreement are given in [DOL 82a, DOL 82b]. The problem of restricting spread of error and its solution in the the light of these ideas, is shown in [RAM 82a]. Here, it is shown how algorithms solving a generalized form of the Byzantine Generals Agreement can be used in conjunction with assertions restricting the values that a variable may take, to prevent error from spreading throughout the system, even if some or all the nodes holding copies of the variable are malfunctioning.

(b) *commit protocols*

Commit protocols (such as the 2-phase commit protocol [GRA 78]) are used to ensure the atomicity of transactions i.e. either all the effects of the transaction are installed (*commit*) into the global information structures concerned or none are (*abort*). Examples of situations where the latter may be a desirable alternative are (a) failures may occur which make completion of the transaction problematic, (b) it may be found that the transaction can not be completed without violating some consistency requirement as a result of concurrently executing transactions which have been operating on the same information.

From the viewpoint of preserving availability, the important question is whether or not the protocol is *nonblocking* [SKE 81]. A commit protocol is said to be non-blocking for a class of failures, if a failure in this class cannot cause a node to wait for another node to recover before it can decide if a transaction is to be committed or aborted. Unless this decision is made, the information being used in this transaction cannot be made available to other transactions. Examples of such non-blocking commit protocols are in [SKE 81, HAM 80].

(c) *node recovery protocol*

A node which has failed or been isolated from changes to the global information because of a partition in the network must be brought up-to-date when the failure has been repaired or the partition ceases to exist. There are broadly two alternative ways of doing this. The first way involves the receiving node obtaining a full copy of the latest version of the information from another node and installing it. The other way is to buffer all the messages that can not be delivered to a node because of failure of the node or its isolation by partition. The first alternative is feasible only if the size of the information is small and failures are rare [GAR 82].

2.2. Consistency

The general nature of consistency requirement of global information is as explained below.

We have in the distributed system a number of concurrently executing transactions (or processes) which operate in various ways on the global information. Each transaction is made up of atomic actions. The effect of the execution of each atomic action on the information is indivisible in the sense that it has exclusive access to it during the execution. For example in a database context, the transaction is made up of read and update actions. Another example is the allocation table for a set of shared resources. The transactions which share the resources perform acquire and release actions on resources thus altering the allocation table to show the changed status of the resources. The deadlock detector executes read actions on the allocation table. An atomic action "sees" the effect of other atomic actions which have executed earlier e.g. in the database example, a read action sees the effect of prior update actions. Consistency requirements are restrictions placed on what an atomic action may be allowed to see. In the database example, one requirement that is usually made is that a

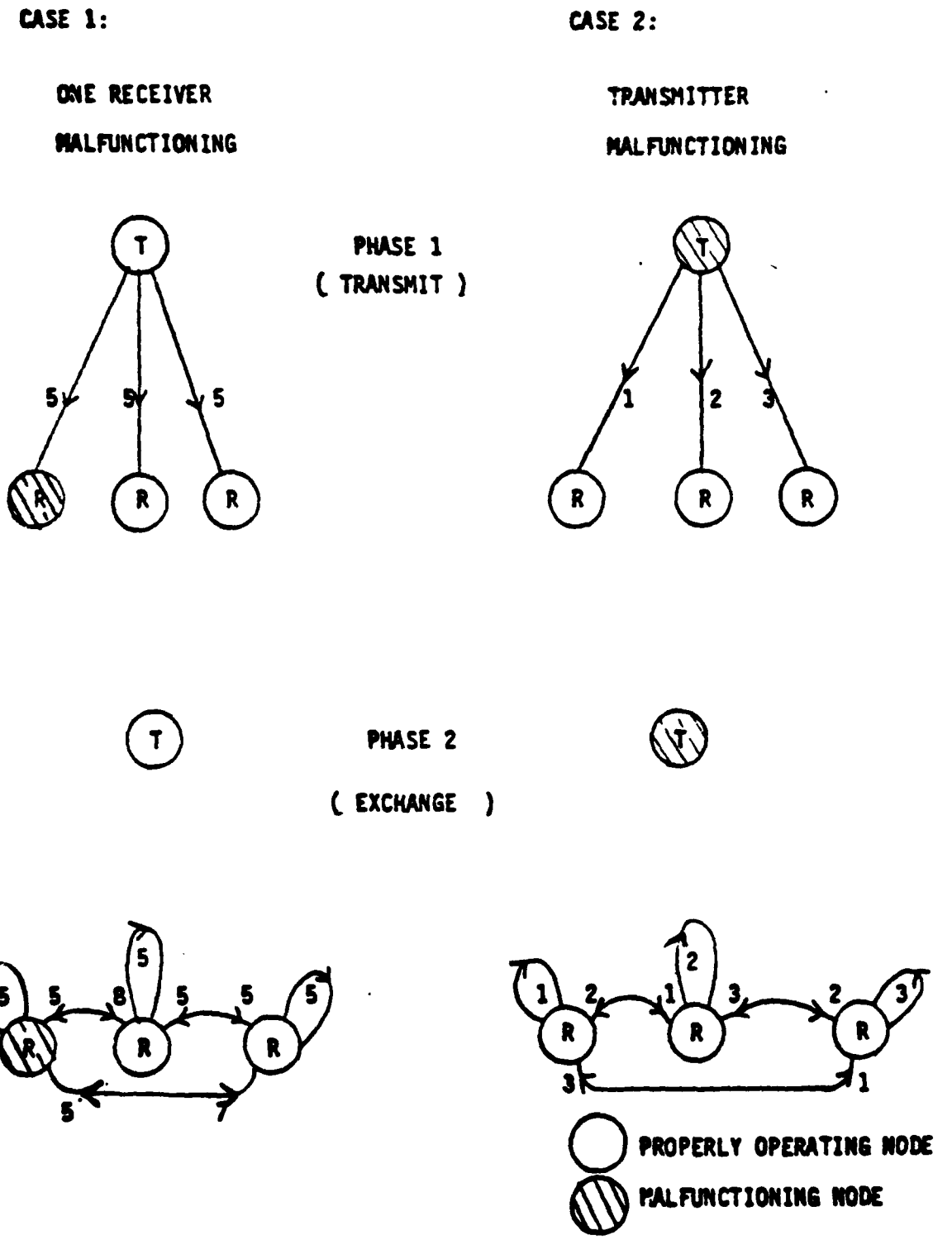


Fig. 1: Reaching Agreement in the Presence of at most one Malfunctioning Node

read action see the effects of all actions of a transaction which executed earlier or see the effects of none. In the example of the resource allocation table, a requirement may be that if the read action of the deadlock detector sees the effect of an acquire action by a transaction, it should also see the effects of any release executed by the transaction prior to the acquire operation. Otherwise, an inconsistent picture of the status of resources and transactions may be formed by the deadlock detector, resulting in the detection of false deadlocks.

The techniques that can be used for ensuring consistency in distributed databases have been elegantly classified in [BER 81]; they fall into the two broad categories of *locking* and *time-stamp ordering*. The answer to the question of which technique is preferable for distributed databases with given requirements must await further research; however, it can be concluded from the available literature that time-stamp ordering appears to provide a more versatile and efficient mechanism for preserving the consistency of global information used in the control of the distributed system. Examples of such cases are in node recovery[HAM 80], maintenance of information on which nodes in the system have failed [WAL 81], dynamic reconfiguration [MA 81] etc.

Some sort of clock mechanism has to be available from which the time-stamps can be derived. For the reasons outlined above and others which space considerations prevent us from mentioning it appears that in a distributed system a clock facility accessible from every part of it can be a very useful feature. Such a facility should have the following characteristics:

(i) it should be distributed for reasons of reliability,survivability and efficiency of access.

(ii) the facility should assign time values which reflect the ordering of events in the computer system. For most applications, it would be sufficient if the values reflected the ordering of events at a single node and the order-

ing of events at different nodes imposed by the flow of messages.

(iii) the value of the clock at a node should preferably be close to the real-world time. This in turn implies that clock values at different nodes should not drift appreciably from one another.

(iv) Maintenance of the clock facility should be inexpensive.

[LAM 78] has proposed a distributed clock mechanism synchronized by messages which satisfies properties (i) and (ii). The mechanism requires the clock at a node to be advanced when a message arrives bearing a timestamp value greater than the local clock value. Hence, all the clocks have a tendency to catch up with the fastest one among them. This in turn may cause them to drift ahead of the real-world time. However, turning them back may vitiate the required ordering of events. [BEL 79] suggest a *slowing down* of clocks, when too large a drift from the real-world time is noticed. This would provide property (iii). Much of the functionality required of the facility could be implemented by hardware or microcode and this could help to satisfy the efficiency requirement mentioned in (iv) above.

2.3. Providing Facilities for Availability and Consistency

Although techniques for handling individual problems in this area are fairly well-developed theoretically, there is a paucity of systems which have implemented any but the simplest options available. The SDD-1 [HAM 80] is one of example of experimentation with a novel distributed operating system (the Relnet) which has attempted to provide time-stamp based lower-level mechanisms on the basis of which the availability and consistency requirements can be fulfilled. Much work remains to be done in exploring and evaluating ways of selecting from the various options available and putting the selected options together to realize viable systems.

3. Communication Protocols

Any distributed data processing system has to be supported by a underlined communication subsystem which may consist of many different components such as terminals, terminal controllers, network front-end processors, communication channels and host computers. Each of these components has its own operating characteristics. A basic function to be performed in the communication subsystem is to provide access paths by which remote end users can communicate with each other. Communication protocols are a set of rules established to regulate the interactions between the attached entities and to accommodate for the differences of components along the path such that the communication can proceed in an orderly fashion. There is a way of looking at the functions of access paths in terms of layered hierarchical structures of network architectures [WEC 79, ZIM 80]. Each layer provides a particular set of services to its next higher layer (or end users).

A number of communication protocols in computer networks of various processing in the past decade. They play very crucial roles in maintaining smooth operations of the entire network. Most of them are implemented in software, except the lowest physical level protocols. However, many design errors or undesired and unexpected behavior have presented in most protocols, due to the informal design techniques. Hence, to verify the correctness of the protocol design and to facilitate its hardware realization for communication, formal specification methods and implementation models for communication protocols are needed in the design and implementation phases. The objectives of these methods include :

(1) to provide a precise and compact specification to eliminate ambiguous interpretation;

(2) to formally define many properties of protocols such as completeness, boundedness, deadlock-freeness and proper termination;

(3) to support formal proofs of those properties;

(4) to provide a better management of complexity; and

(5) to facilitate automatic implementation of the protocol.

The specification method and the implementation model should be general enough so that they are applicable to every layer of the protocol hierarchy. The complexities of developing these models are mostly due to the distributed nature of information and concurrency of operations. With the development of formal specification models and implementation models, a consistent and systematic design process from the specification down to the architecture level of the protocol is demonstrated, which achieves the above goals.

3.1. Techniques For Modeling Communication Protocols

Currently, there are two basic approaches for modeling and validating point-to-point protocols [SUN 79]: (1) state-oriented models together with reachability analysis [MER 79b, RAM 82c, WES 78] and (2) language-oriented models followed by program proving [TEN 78, SCH 81].

The first approach primarily models the control flow in protocols. Protocol properties can be validated in terms of the structure of the state transition graph. However it suffers from the state explosion problem. The size of the state transition graph grows rapidly with protocol complexity. This can be exemplified by the number of states used in [RAZ 80] where they jump from 235 to 3890 when undefined, lost and external signals all included in the primitive model for X.21 protocol. Also, transition-oriented models do not attempt to adequately model data transfer aspects of protocols such as timers and sequence numbers. It will make the state explosion problem worse if data transfer aspects are included in pure transition oriented models.

The second approach, the language-oriented approach is poor in control flow modeling because of the lack of suitable and simple representations for protocol properties. The data transfer aspects instead can be successfully represented because of the capability in handling variables and parameters. It requires the use of program proving techniques which is very complex. The formulation of assertions and/or invariants for proving protocol correctness involves semantics of protocol specifications and needs a great deal of ingenuity to derive.

There are some other augmented state transition models [DAN 78, BOC 80] attempting to integrate the features of both approaches. However, a great deal of work remains to be done. On the synthesis and implementation of protocols, the research literature is scarce [GOU 76, HOF 80]. The reported research is not complete in the sense that it does not deal with protocol multiplexing [COH 79] which is an important implementation feature of layered protocols.

In next two subsections, a specification method and a consistent implementation model for communication protocols, which are applicable to every protocol layer, will be briefly presented to expose various properties and complexities during the development of such models.

3.2. Specification Model Development

Here, Petri net models [PET 77] are chosen as the basic specification modeling tool. With their concurrent, asynchronous and nondeterministic nature, and many common properties such as liveness and boundedness they share with protocols, Petri nets are very suitable to model communication protocols. Since state oriented models and language oriented models are complementary to each other with respect to control flow representation and data transfer modeling capabilities, Petri nets are augmented with state variables

and transition procedures to model full range properties of protocols.

In layered protocol structures, a protocol at some particular level conceptually consists of two communicating entities and a full-duplex virtual link. The *local modeling* approach is an appropriate way to represent the nature and properties of this structure. In local modeling, each entity and each link are modeled separately by a Petri net. An advantage of this approach is that it can be directly implemented in each party without the problem of inconsistent or noncompatible decomposition among the parties. However, to ensure the correctness of a protocol, one still has to go through global analysis. Simply looking at the behavior of each local model cannot have the global view of the interactions between communicating entities and hence cannot guarantee the correctness of the protocol. Since direct synthesization from the state transition graph of each party will give rise to the state explosion problem in complex protocols, a compact global Petri net model for communicable entities is constructed from their local models directly in order to alleviate the state explosion problem. In each individual local model, there is a need to communicate with outside world. Therefore a set of external input (output) places is defined in each local model, which may be used to represent incoming (outgoing) messages to (from) the local model. Entities are communicable if their external places match with each other's.

In order to model the behavior of a link, a set of primitive transitions is defined in the link machine, such as ***normal operation***, ***message lost***, ***message duplicated***, and ***message corrupted***. The purposes are: (1) error recovery strategies used in protocols can be formally incorporated, and (2) the performance of the communication link will be taken into consideration in evaluating the performance of the entire protocol.

To analyze the control flow of the global model, reachability analysis is employed. Only after the general control properties have been proved correct, the specific protocol requirements will be checked against the specification by employing conventional assertion proving techniques. Regarding the state explosion problem, [RAM 82] has developed a Petri net abstraction procedure to control this problem. The complexity of analysis has been reduced significantly by employing this method. Although the method is effective for models which are highly structured, research efforts are still needed to resolve the state explosion problem in general.

3.3. Implementation Model Development

The realization of the protocol at a particular layer involves the implementation of the entity machines only. Link machines model the behavior of lower level protocols, which will then be taken care when we implement lower level protocols. In order to bridge the gap between the specification model and the actual implementation, an intermediate implementation model is necessary. An implementation model for the entity machine is developed from its Petri net model.

The implementation model provides a framework for the architecture of a protocol handler, from which the protocol handler is readily realized. The implementation model proposed in Fig. 2 is compatible with the choice of our augmented Petri net model. It is external event driven because the operations of the protocol service are of the request-response type. Moreover, since the layered hierarchy of the protocol allows each level to multiplex several, possibly different, higher level entity machines [COH 79], our implementation model also supports this capability by including an Entity Machine Multiplexer (EMM) module. EMM schedules the processing of service requests.

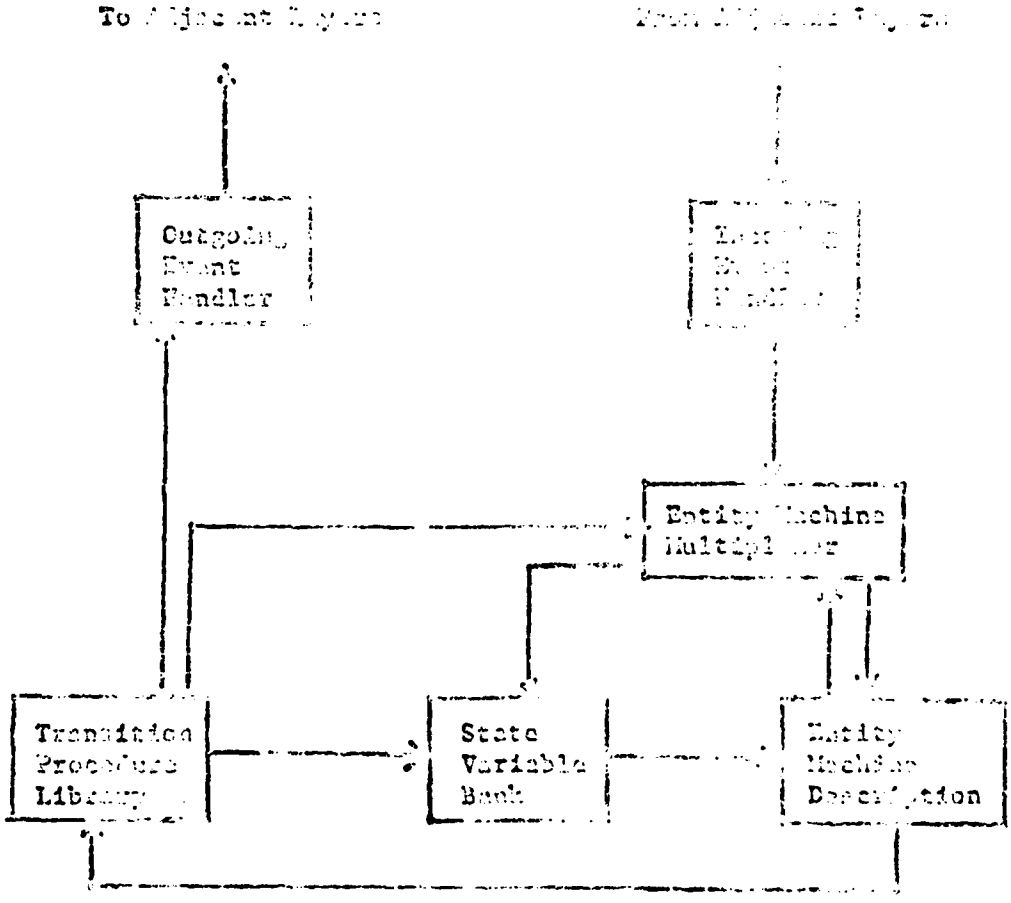


Fig. 7 Implementation model of a protocol handler at a given layer

Entity Machine Description (EMD), State Variable Bank (SVB) and Transition Procedure Library (TPL) are derived from our formal specification model directly. The EMD is the machine readable form of the Petri net model. It could be an incidence matrix representation or a production system implemented in associative memory such that, given an input marking pattern, we may readily know the firable transitions. The SVB consists of separate local copies of state variables for entities which are running concurrently. The TPL is the library of transition procedures. As in [HOF 80], depending upon the choices of hardware realization approaches, these procedures could be implemented in random logic or microprogramming or even handled by microprocessors.

The incoming event handler decodes the messages and identifies the incoming requests and the destination entity for each request. Interrupt mechanisms or polling schemes could be used in receiving data. Once an request queued in EMM is activated, EMD and SVB will be consulted according to the entity identity and some control signals will be generated to activate a proper transition procedure. Depending on the scheduling policy, timing requirements and priority structures, EMM could multiplex the entities on request-by-request basis or transition-by-transition basis. In the first case, an activated entity machine will complete its service for an request before next entity machine is activated. In the second case, an activated entity machine has to wait for rescheduling after the execution of a single transition.

Transition procedures will modify corresponding state variables and produce any necessary outgoing messages for outgoing event handler. The outgoing event handler encodes and combines messages in a well-defined format and forwards them to other adjacent layers.

3.4. Multi-Party Communication

With the growing interest in distributed processing systems, new applications such as office automation, teleconferencing, electronics mails have been continuously emerging. They frequently require the exchange of multi-destination messages. For instance, to update replicate files in a distributed data base, the data base manager may want to transmit multi-destination messages to those nodes where the replicate files locate. Multi-party communication is the transmission of multi-destination messages among some specified nodes. An objective to employ multi-party communication is to further exploit the inherent concurrency in distributed systems. Most of research to date is restricted to point-to-point communication, very few researchers pay attentions to the multi-party communication problem.

Multi-destination routing, that is, how to direct a message from its source to a group of intended receivers, is the first important issue. It is essential to the design of multi-communication systems. Broadcast routing is a special case of multi-destination routing. However, routing is only one aspect of the design. To ensure the reliable transport of multi-destination messages, robust and correct multi-party communication protocols are needed. Techniques we have presented before for modeling and designing point-to-point communication protocols should be extended to cover the multi-party communication. A lot of research effort is needed in this nearly unexplored area.

4. ROUTING

The goal of routing algorithms is to provide some best collection of paths between message sources and destinations, given various network conditions such as the delay of each link, the network topology and so on. The definition of

"best" collection really depends on the the user or system's objectives. For example, to achieve minimum delay of individual packets has been a common objective for many existing routing algorithms such as the original and new Arpanet routing algorithms [MCQ 74, MCQ 80]; to achieve minimum delay of all system-wide packets is another objective [FRA 73]; extremely reliable transmission and others are some of the typical objectives of the existing algorithms. The objectives are closely related to the user's requirements. The user may require that the routing algorithm support either the datagram or virtual circuit kind of communication, or both. He may also require the routing algorithm to deliver multi-priority messages varying from the "Federal Express" type of messages to the "bulk rate" type. The user may want a routing algorithm that can be used to design the network topology while another routing algorithm may be used at real time. If the algorithm is to be used at real time, the user may specify how reliable and how fast the algorithm should be. If the network is to be operated in the dynamic environment, reliability is probably the most important concern. Survivability needs usually will require the system to be operated without any central controller and to be able to reconfigure in a decentralized fashion whenever the system status changes. In order to achieve these objectives, the routing designers would have to consider many factors such as delay and bandwidth of each of the links, flow control, network topology, processor capability, memory size and so on. The routing algorithm should be able to adapt itself to the changing environment such as a sudden burst of input from the user, or congestion among some of the nodes in the network. The global topology as well as the local configuration should be considered before computing the routing table. The node processor capability and memory size are limited, thus the routing algorithms designed should not use up too much computational resources or occupy large amount of memory. During the past ten years, a great deal of research has been done in this area resulting in numerous

routing algorithms. However, many of the algorithms do not consider one or more of the factors involved mentioned above. For example, some routing algorithms designed for packet radio networks do not consider the multiaccess protocol [GAF 81]; the flow deviation routing algorithm [FRA 73] and Courtois' algorithm [COU 81] do not treat the update problem for they are designed to be used at the design stage and not for real-time application; Gallager's distributed algorithm has been pointed out to be of theoretical importance but difficult to be used in the real environment [CER 81]. Due to the complexity of routing algorithms, it may not be possible to devise a single algorithm that can satisfy all objectives mentioned above and work under all different environments.

In the future, a network is likely to be equipped with several different kinds of routing algorithms, each with different objectives and suitable for some specific environments only. As the network topology changes, or users' requirements change, the network automatically changes its routing policy so as to achieve maximum utilization of the resources, highest efficiency or minimum delay. However, in order to achieve this goal, we will have to understand various tradeoffs among numerous routing algorithms available right now. Many researchers only discuss the specific properties of their algorithms and do not give a comprehensive comparison of their algorithms with the other ones. Research in comparison study and classification of the routing algorithms is very much in need. Since we are interested in large networks, we choose to discuss distributed and hierarchical routing algorithms. Ever since distributed routing algorithms were first popularized by the design and implementation of the original Arpanet algorithm, a lot of attention has been paid to that type of algorithms. Hierarchical routing algorithms are discussed because they are becoming important for the future as the size of networks grow larger and larger, and comparatively little has been said about them in the literature.

4.1. Distributed Routing Algorithms

There are many advantages to distributed routing algorithms as compared the more conventional centralized routing algorithms, the most notable one being *reliability*. Distributed routing algorithms do not require any central controller to compute the routing table, the failure of which halts the operation of the network. Usually, both the initial computation of the routes and the subsequent update process are carried out autonomously without any supervision by a central monitor. The whole process is iteratively done by computation of the routing table at each node and exchange of information among the neighbors. This type of algorithm was first introduced by the original Arpanet algorithm, and subsequently many distributed routing algorithms appears have appeared [CHU 78, JAF 81, GAL 77, GAF 81, SEG 81, SEG 82, MCQ 80]. The actual number of publications is far greater than the list indicates.

Due to the nature of parallel processing and the exchange of routing information by many different nodes at the same time, distributed algorithms are more difficult to understand, to verify, and to analyze than their centralized counterparts. This is witnessed by the development of the original Arpanet routing algorithm. The original Arpanet algorithm evolved from Baran's hot potato algorithm [BAR 64], and its centralized version was developed by Bellman, Ford and Fulkerson some time back [LAW 76, SCH 80]. However, the proof of its correctness was not available until 1977 [TAJ 77]. One of the major problems of this routing algorithm is that a loop may form during the update process. Many of the recent publications still discuss new techniques to handle the problem of the loop for that algorithm [JAF 81, CHU 78].

There are many questions remaining unanswered. For example, the question of how much information should be kept at each node has not been completely understood. The amount of information stored at each node will cer-

tainly affect the update cost as well as the computational overhead. If we store complete topological information at each node, the problem of update becomes one of the update problems in a fully replicated data base. The new Arpanet algorithm takes this approach and the whole network topology information is replicated throughout the network. The update algorithm is the flooding algorithm which ensures that every node that is reachable will be reached[MCQ 80]. If we store no information at each node, the only routing algorithms we could have are the flooding and random routing algorithms. There is no need to update. In between these two extremes, there are many possible ways to implement a distributed routing algorithm. The original Arpanet algorithm is an example of one that stores condensed information instead of whole network connectivity, and the update protocol is more complicated than the new Arpanet algorithm. It has been observed that it is better to store more information at each node , because by doing so one is likely to increase the reliability, decrease the communicational overhead but obtain greater computational overhead [TSA 82]. However, the question of how much information should be stored at each node and what kind of information should be kept remain basically open.

Distributed routing algorithms would be no use if they could not update the routing table autonomously without excessive communicational and computational overhead. The design of such distributed routing algorithms is still a challenging task. As we have discussed above, the update protocol is dependent on the how and what information is kept at each node. However, given the kind and amount of information stored at each node, there are many kinds of update protocols that could be used. For example, there are several kinds of update protocols that could work for a network in which each node keeps the same information as the original Arpanet. How should we choose from them?

Much research in distributed routing has concentrated on supporting the datagram, however, there are times when we need to support the virtual circuit

kind of communication. Can distributed routing algorithms satisfactorily support virtual circuits or a combination of datagrams and virtual circuits? We think so, but more work needs to be done before the belief can be substantiated. Some early work has been done in this area [MOS 77].

Furthermore, many distributed routing algorithms have limited their objectives to minimizing the delay of individual packets. Much work remains to be done in designing distributed routing algorithms that could satisfy other objectives such as minimum average system-wide messages delay, or minimum communication control packets, or maximum flow and so on.

4.2. Hierarchical Routing Algorithms

As the network size grows larger and larger, both centralized and distributed conventional routing algorithms becomes less suitable. The difficulties of using conventional routing algorithms in a very large network come from the excessiveness of the communicational and computational overhead. Kleinrock and Kamoun have shown that given the limitation on memory size at each node and the need for periodic updates, hierarchical routing algorithms would have to be used in order to avoid the long delay and small throughput that will be incurred by using conventional distributed routing algorithms [KAM 79]. However, much less work has been done in this area than in any other area of routing. In order to do hierarchical routing, the network is partitioned into several clusters with links connecting these clusters. Communication between nodes of different clusters is referred to as inter-cluster communication. For the within-cluster communication, conventional routing algorithms are employed. However, inter-cluster communication requires some handlings.

Most of the work that has been done in hierarchical routing concentrates on extension of the original Arpanet routing algorithm to a hierarchical scheme, e.g. [KAM 79, KLE 77, MCQ 74, HER 82]. These schemes thus incur both of the

advantages and disadvantages of the original Arpanet algorithm. They are relatively easy to implement, use decentralized control, and have fast response time when the good news (delays along some links decrease) arrives. However, they perform poorly when some of the links fail or the delays along some of the links increase. The loop problem may arise as in the case of the original Arpanet routing algorithm. In contrast, some other researchers have extended centralized routing algorithms to hierarchical scheme, e.g. [CHU 81]. In Chu's scheme, all nodes would have to send their status to the local supervisors; the supervisors would then broadcast the routing table to local nodes after computing the routing information. Whenever there is any change in the network, the nodes that detect the change will report to the supervisor and the supervisor would compute the routing table again. If there is sufficient change in the routing table, the supervisor would broadcast this information to all the nodes. The major disadvantage of this scheme is that it uses central controllers: The loss of the central controller will have disastrous effects on the network. In the following discussion, we will refer to those hierarchical algorithms that are extended from the centralized conventional routing algorithms as centralized hierarchical algorithms, and refer to those algorithms that are extended from the distributed schemes as distributed hierarchical algorithms.

In fact, there are many more ways to implement hierarchical routing algorithms. For example in the centralized case, we could use hill-climbing algorithm instead of Chu's scheme. In this scheme, if a node wishes to communicate with a distant node, it would have to send the packet to the center where it will have more information to make the decision on how to route the packet. If the center does not know how to route the packet, it will forward the packet to the center at a higher level; otherwise, it will forward the packet to the destination. This process is similar to the traveler climbing up a hill. The process by which the packet is sent from the topmost center to the the destination could take

exactly the reverse form: the center forwards the packet to the center that is one level below, which will either forward the packet directly to the destination or forward the packet to the center one level down, and the process is repeated until the packet arrives at the destination. Another possible way of forwarding is that the center computes the whole route to the destination and sticks the address to the packet before forwarding, and the intermediate nodes just follow the instructions on the packet. There are, of course, many other ways to implement the centralized hierarchical scheme. In the case of the distributed hierarchical scheme, we could extend the new Arpanet routing algorithm instead of the original Arpanet routing algorithm. Such algorithms could potentially avoid many of the disadvantages of the original Arpanet algorithm such as the loop, slow response time for bad news and so on. The reader is referred to [RAM 82b, TSA 82] for more details of such an algorithm. Some hybrid schemes also could be designed, e.g. one in which a distributed routing algorithm is used for local communication, while another centralized algorithm is used at a higher level, or vice versa. As we have just demonstrated there are many ways to implement hierarchical routing, and it is essential to characterize various routing algorithms and carry out research in comprehensive comparison of these algorithms.

A very important problem associated with hierarchical routing is how and how often we should update the routing table. Previous investigations have always chosen to update immediately if there are any changes in the network using either a centralized or a distributed scheme. However, it seems that it is not necessary to communicate every change as soon as possible. Not every change in the network topology will have global effects or any effect outside the local cluster. For example in Fig. 3, the failure of link AB may not have any effect outside the cluster, as there are sufficient redundant paths connecting east, west, south, and north points. A notion of "delayed update" has been devised

that could potentially save a lot of communicational overhead [RAM 82b, TSA 82]. The idea of delayed update is that we postpone any update until we have to perform it.

In order to do hierarchical routing, we should first divide the nodes in the network into some clusters. We still do not fully understand what constitutes an optimal clustering structure. Kleinrock and Kamoun have done excellent work in this area, and their objective is minimization of the routing tables stored at each node [KLE 77]. However, there are some other objectives such as reliability, communicational overhead and so on. The problem of finding clustering structures with objective of optimizing such attributes is unsolved. For example, should the clusters be overlapped? How do we dynamically recluster the network as network topology changes? How do we know which nodes are in which clusters? Most of previous work has concentrated on the static network, where once a node is assigned to a cluster, it will remain there forever. If some of the nodes move from one cluster to another cluster, how should the network respond? These problems are open.

5. Dynamic Reconfiguration

The design of dynamically reconfigurable architectures is basically driven by enhancing the reliability and availability of a system through changes in its configuration so that the effect of failures and faults may be bypassed. This ability of a system known as reconfigurability can be defined as the ability to change its physical and/or functional organization in response to failures or faults in the system. A sequence of events will occur in a dynamically reconfigurable system in response to any faults. The sequences of responses consist of the following four steps : (1) detection of fault; (2) location of the fault; (3) reconfiguration or repair; and (4) computation recovery. Techniques for carrying out these steps will be discussed subsequently.

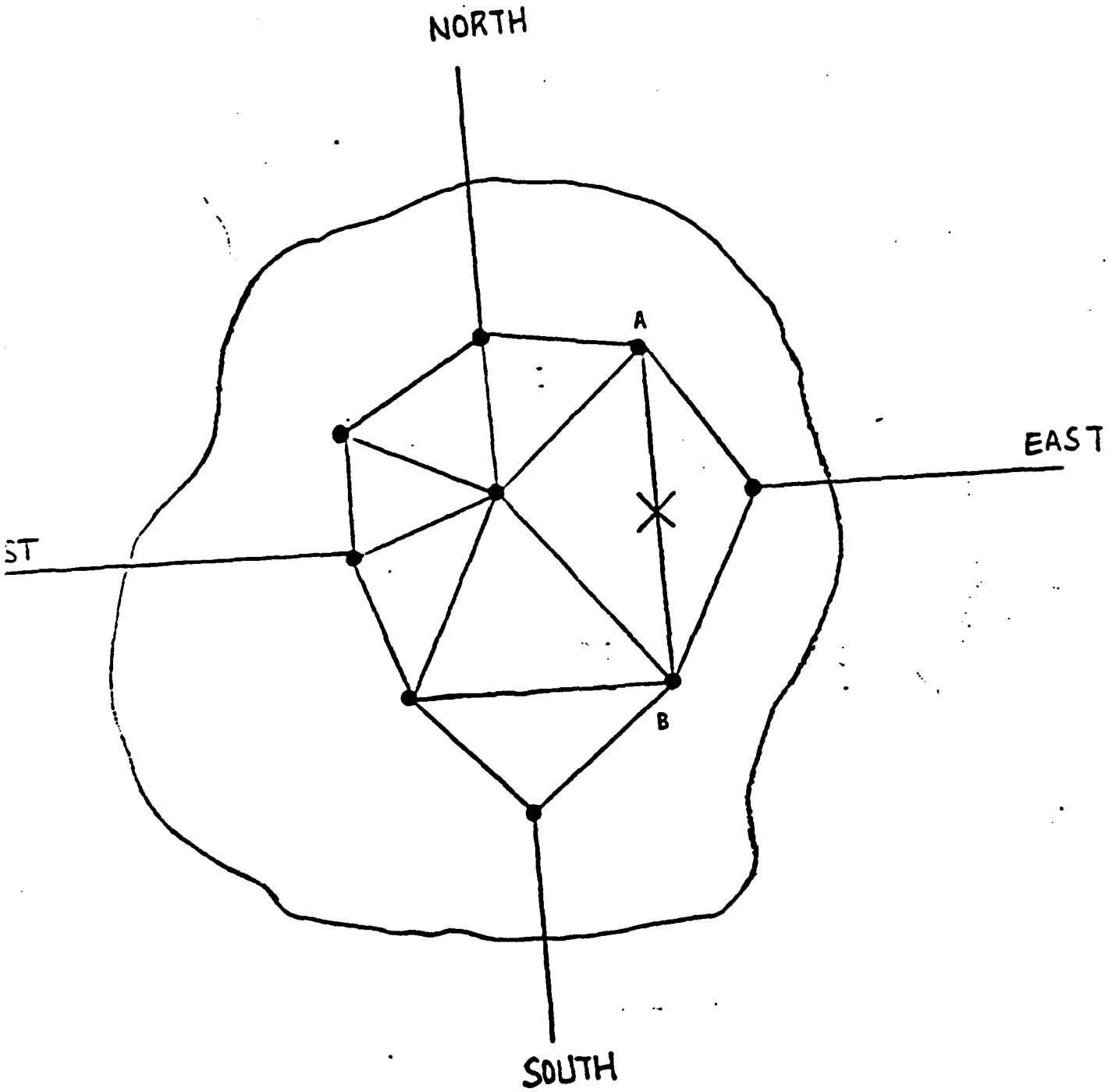


Fig. 3

5.1. Fault Detection

The first step in designing dynamically reconfigurable systems is to design mechanisms which are able to detect faults or errors caused by faulty elements. Since steps (2), (3) and (4) can be started only if the faults are able to be detected, the correctly designed fault detection facilities are extremely important for the proper operations of reconfigurable systems.

Fault detection can be implemented by means of special hardware and /or software during the system operation. The hardware methods of detection include error detection codes such as parity, duplication and comparison, majority voting, self checking, logic circuits, status check and flags, and the built-in testing schemes. Software methods of fault detection usually employ redundant modules, concurrent execution and comparison of results on executable assertions. No matter which methods are chosen, fault detection mechanisms should permit distributed or hierarchical fault detection so that the failures at different levels of the distributed system hierarchy can be handled in a decentralized manner.

In systems controlled in a distributed fashion, the distribution of the fault detection mechanisms is essential. It not only meets the design philosophy of such systems but also makes the systems more robust. Each nonfaulty processing element must be able to independently and correctly detect any failures and malfunctions of other processors, based on the analysis of information which are exchanged between neighboring nodes through communication links. The good management of global information certainly is very helpful in accelerating the detection of faults. Because of the high communication cost involved in interprocessor communication, most of the above described hardware fault detection techniques may not be applicable to the detection of faulty processors in the distributed environment. For example, the majority voting technique may incur

too much communication overhead if processors are geographically distributed. [KUH 80] has developed a simple but effective way of detecting and isolating remote faulty processing elements so that the fault detection is achieved in a distributed fashion.

5.2. Fault Location

Once the faults are detected, the next step is the location of faults. Again distributed or hierarchical fault location techniques will be more appropriate to locate faults in a distributed computing system. Most of the previous work in fault location deals with faults at the logic level. The stuck-at-1 or stuck-at-0 may be the most frequently used model to represent faults in logic circuits. This circuit level representation is too detailed to show faults at the system level and too low-level to describe the faulty behavior at the system level. At the system level, we are more interested in locating faults of some specific units such as processors and links with little regard to the specific locations of faults within the faulty unit.

A faulty unit is a unit in which one or more faults is present. Tests which are performed to identify the faulty unit may be classified as hardware-implemented or software-implemented. Typical hardware-implemented tests employ redundant facilities and information in the form of error detecting codes or replication. The testing is then carried out by feeding carefully selected sets of inputs during normal operation and observing system outputs with code checkers or comparators. These techniques, however, are not adequate to handle all fault-location problems; because, firstly, faults may not always be detected by checking algorithms used and, secondly, the selected input sets may not be complete enough to detect and disclose fault locations. To resolve these difficulties, sequences of tests should be used instead. These test sequences, if they are relatively short or easily generated, can be produced by

dedicated hardware in the unit. Such approach is particularly attractive to be used in VLSI implementation. For more complex sequences and for testing interfaces between units, software tests can be applied by one unit to another. One basic assumption about these test sequences is that at least one processing element functions correctly. Starting from the nonfaulty processor, each processor can then be tested and determined faulty or not.

Error propagation is one of the important issues that affect the techniques for fault location significantly. Obviously, if errors are allowed to propagate throughout the system in an unrestricted manner, location of errors will be difficult to perform if not impossible. Thus, the reconfiguration and recovery steps may become impossible. There are two strategies in dealing with the error propagation problem. The first strategy is to prevent errors from propagating beyond the boundary of processors. A commonly used technique is to provide some hardware mechanisms which are able to perform concurrent fault detection for any information into or beyond boundary of a processing element. Such concurrent fault detection permits immediate identification of erroneous data, thus enables tight error containment. Another strategy is to require each individual processing element record its own history of communication, such as messages exchanged, so that, when necessary, the extent of error propagation can be determined by tracing the history of interactions. This approach requires that checkpoint information be maintained and requires special mechanisms to protect the processes implementing fault tolerance. The second strategy may involve a lot of communication overhead if the messages are inter-processor messages.

5.3. Reconfiguration Strategies

After the faults are located, either the faulty processing elements will be repaired and the computation recovered or the whole system will be reconfigured in such a way that the faulty units are excluded from the system without human intervention. Methods for computation recovery have been discussed extensively in [KIM 79, KOH 81]. Here, we will only focus on the reconfiguration issue.

Given a system faulty condition, there may be many different methods to reconfigure the system to the normal operation. A systematic way to design the optimal reconfiguration strategies is described as follows. First, appropriate models and analysis techniques should be developed to analyze faulty behaviors of the systems. Based on these models, the conditions for feasibility of reconfiguration are derived. Such a feasibility study indicates whether a system would be able to recover and recuperate from the given failures. Under various assumptions about types and numbers of simultaneous faults, faults during reconfiguration or not, priori knowledge of fault probabilities or not and so on, [HEL 78, MA 81, JEN 82] have described various models and corresponding figures of merit to study the optimal reconfiguration. Also, some problems in designing the optimal reconfiguration strategies have been proved to be NP-completeness [MA 81]. However, more general models for describing the faulty behavior of a system are still needed to provide a more realistic view in studying dynamic reconfiguration.

Besides developing formal methods of selecting the optimal reconfiguration strategy, there are some other practical considerations that should be taken into account in designing dynamically reconfigurable systems and realizing reconfiguration strategies. For examples, the processors in the system should be functionally compatible so that the function of a failed processor can be

shifted to other compatible processors. The interconnection network should be able to remain connected even after a node in the connected path fails. This may require special attention on topology design and need some switching elements to perform connections. The reconfiguration strategies designed to cope with different types of failures should be verified formally to assure their correctness. Experiences obtained in protocol validation can be applied to this aspect. Because of the high communication cost in distributed systems, the reconfiguration strategy should also be designed to minimize communication overhead so as to be efficient and cost effective.

6. Deadlock Handling

Three classes of strategies may be distinguished for handling deadlocks. Deadlock *prevention* strategies work by imposing some constraint on the pattern of resource usage by requesting processes, e.g. resource ordering, acquiring all needed resources simultaneously, etc. Deadlock *avoidance* algorithms operate by granting requests only when at least one path exists for the processes to complete execution, e.g. Habermann's algorithm. In deadlock detection strategies, no constraint need be imposed on the granting of an available resource; periodic checks are made of the state of resources and processes to ensure that no process is deadlocked. The state can be maintained in the form of a *general resource* graph [HOL 72]. It can be shown that if all processes having requests are blocked, the necessary and sufficient condition for deadlock is the existence of a cycle in the graph.

A comparison of these techniques in the context of their behavior in distributed environments shows that deadlock avoidance is unlikely to be cost effective because of the volume of communication required to cost-effective because of the volume of communication required to transmit the usage

patterns of processes in advance. Further, if the run-time computations of safe paths are to be distributed, additional communication may be required on each resource request. Deadlock prevention strategies tend to reduce concurrency by forcing processes to request resources earlier than required. They are most effective when a "natural" ordering exists in the use of resources. For example, the designers of Medusa ensured that the layers in Medusa were allocated disjoint sets of resources and thus prevented deadlocks which spanned layers.

Therefore, if a high degree of concurrency is desired, detection strategies appear to be the best strategy in a distributed environment. They can be centralized, hierarchical or distributed in the allocation of detection capability over a network. As a result of non-zero communication delays, resource allocation information at a node may not be up-to-date. The detection algorithms, computing on the basis of such information, may be unable to detect an existing deadlock, or give an indication of deadlock when none exists. In fact, many of the algorithms proposed in the literature have been shown to behave incorrectly under race conditions.

Centralized and hierarchical algorithms which behave robustly in the face of the above-mentioned race conditions have been developed in [HO 79]. The algorithms can be executed in one or two phases, trading off response-time for space.

Truly distributed algorithms are very important because such algorithms require only the nodes involved in the deadlock to participate in detecting it. Hence a minimum of reconfiguration is required under node crashes and network partitions. Three distributed algorithms described in the literature are in [MEN 79, CHA 82, OBE 82]. [MEN 79] stores "condensed" information to improve response time, but does not adequately address the problems of updating such condensed information. On the other hand, the algorithms in [CHA 81] and [OBE

82] do not condense information and hence the response time may not be small enough. In addition, false deadlocks are avoided in [OBE 82] by a validation phase after the preliminary detection, which will cause the response time to increase. Development of an efficient and robust distributed algorithm with a low response time must await future research.

7. Summary

This report has focussed on some unsolved problems related to certain functions involved in the control of a distributed processing system.

In the management of global information, availability must be maintained in the presence of failures and consistency must be maintained in the presence of concurrency. With reference to the former attribute, many algorithms and techniques are appearing in the literature for handling various kinds and degrees of failures and preserving availability in spite of them. With reference to the latter attribute, it is becoming clear that time-stamp ordering based on a distributed clock mechanism is a very flexible and versatile method for ensuring consistency. However, there is a lack of research in exploring putting together these techniques in a unified, coherent manner to realize viable systems.

In the area of dynamic reconfiguration, although general procedures to respond fault conditions in a system are well understood, efficient and robust algorithms are still strongly demanded for detecting faults distributedly, for managing the error propagation problem and for selecting optimal reconfiguration strategies. The desirable distributed algorithms should take into account both computation efficiency and communication overhead.

In the field of deadlock handling there is a lack of distributed algorithms, which use condensed information to detect deadlocks rapidly and solve the problems involved in managing this condensed information.

Most of research in the area of long haul computer communication is limited to issues on point-to-point communication to date. We have shown general techniques for modeling and analyzing communication protocols and examples of the specification and implementation model development. However, to model and analyze full range properties of communication protocols, research effort is further needed to extend hybrid models, to manage the state explosion problem and to explore issues in multi-party communication.

Distributed routing algorithms are usually preferred over the centralized schemes. However, due to the nature of parallel processing, they are difficult to understand, to verify, and to analyze Existing algorithms suffer from a lack of validation for these reasons and hence research is needed on the characterization and comparison study of these algorithms. Much previous work is the area has concentrated in minimizing the delays of individual packets. Future research in distributed routing algorithms should include other objectives such as minimizing the average system-wide message delay, minimum communication control packets and so on.

Since employment of conventional routing algorithms would incur excessive communication and computational overhead, hierarchical routing algorithms are necessary for networks of very large size. Previous research in this area has almost exclusively concentrated on extension of the original Arpanet algorithm. Research is needed in exploring some other objectives to implement hierarchical routing. Other issues such as optimal clustering structures that achieve various objectives, dynamic reclustering strategies and so on are also needed to be investigated.

8. References

[BAR 64] Baran, P. "On Distributed Communications," series of 11 reports. The Rand Corporation, Santa Monica, Calif. August 1964.

[BEL 79] Belford, G.G. and Grapa, E. "Setting Clocks 'back' in a Distributed Computing System," 1st Intl. Conf. Distributed Computing Systems, Huntsville, Ala., Oct. 1979.

[BER 81] Bernstein,P.A. and Goodman,N. "Concurrency Control in Distributed Database Systems", ACM Computing Surveys, June 1981.

[BOC 80] Bockmann, G.V. "A General Transitions model for Protocols and Communication Services," IEEE Trans, Comm., Vol. COM-28, No. 4, April 1980.

[CER 81] Cerf, V., "Packet Communication Technology," in *Protocols & Techniques for Data Communication Networks*, edited by F. F. Kuo, Prentice-Hall, 1981.

[CHA 82] Chandy, K.M., Misra, J. and Haas, L.M., "A Distributed Deadlock Detection Algorithm and its Correctness Proof," Tech. Rep. TR-LCS-8204. The University of Texas at Austin, Feb. 1982.

[CHU 78] Chu, K.-C., "A Distributed Protocol for Updating Network Topology Information," IBM Research Report RC 7235, 1978.

[CHU 81] Chu, W., and Shen, M., "A Hierarchical Routing and Flow Control Policy (HRFC) for Packet Switched Networks " IEEE Tran. on Computer, 1981.

[COH 79] Cohen, D. and Postel, J., "On Protocol Multiplexing," Proc. 4 6th Data Comm. Symp., Nov. 1979.

[COU 81] Courtois, P.-J., and Semal, P., "An Algorithm for the Optimization of Nonbifurcated Flows in Computer Communication Networks," Performance Evalluation 1(1981) 139-152.

[DAN 78] Danthine, A. and Bremer, J., "Modelling and Verification of End-to-End Transport Protocols," Computer Networks, vol. 2, Oct. 1978.

[DOL 82a] Dolev, D., Fischer,M.J., Fowler,R., Lynch,N.A. and Strong,H.R. "An Efficient Byzantine Agreement without Authentication", to be published.

[DOL 82b] Dolev, D. and Strong, H.R. "Authenticated Algorithms for Byzantine Algorithms for Byzantine Agreement",IBM Research Report RJ3416 (40682), April 1982.

[FRA 73] Fratta, L., Gerla M., and Kleinrock, L., "The Flow Deviation Method: An Approach to Store-and Forward Computer-Communication Network Design," Networks, 3, pp. 97-133, 1973.

[GAF 81] Gafni, E., and Bertsekas, D., "Distributed Algorithms for Generating Loop-Free Routes in Networks with Frequently Changing Topology," IEEE Trans. on Comm., Jan. 1981.

[GAL 77] Gallager, R.G., "A Minimum Delay Routing Algorithm using Distributed Computation," IEEE Trans. on Communications, COM-25(1), January 1977.

[GAR 82] Garcia-Molina, H. "Reliability Issues in Fully Replicated Databases", Computer,Sept. 1982.

[GOU 76] Gouda, M.G. and Manning, E.G., "Protocol Machines: A Concise Model and its Automatic Implementation," Proc. of the 3rd ICCC, Toronto, Aug. 1976.

[GRA 78] Gray, J.N., "Notes on Database Operating Systems," in *Operating Systems - an advanced course*, R. Bayer et al, Eds., Springer Verlag, New York 1978.

[HAM 80] Hammer, M. and Shipman,D. "Reliability Mechanisms for SDD-1: A System for Distributed Databases",ACM Trans. Database Systems, Dec. 1980.

[HEL 78] Helvik, B.E., "An Approach to Optimal Reconfiguration in Dynamic Fault-Tolerant Systems", 8th Int'l Conf. on Fault-Tolerant Computing, June 1978.

[HER 82] Heritsch, R., "A Distributed Routing Protocol for a Packet Radio Network," M.S. Thesis, Naval Postgraduate School, March 1982.

[HOF 80] Hoffmann, M.G., "Hardware Implementation of Communication Protocols: A Formal Approach," Proc. of 7th Symp. on Computer Architecture, May 1980.

[HOL 72] Holt, R.C., "Some Deadlock Properties of Computer Systems," ACM Computing Surveys, 4,3 (Dec. 1972).

[KAM 79] Kamoun, F., and Kleinrock, L., "Stochastic Performance Evaluation of Hierarchical Routing for Large Networks," Computer Networks 3(1979) 337-353.

[KIM 79] Kim, K.H., "Error Detection, Reconfiguration and Recovery in Distributed Processing Systems", Proc. 1st Int'l Conf. on Distributed Computing Systems, Oct. 1979.

[KLE 77] Kleinrock, L., and Kamoun, F., "Hierarchical Routing for Large Networks--Performance Evaluation and Optimization," Computer Networks, 1 (1977) 155-174.

[KOH 81] Kohler, W.H., "A survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems," ACM Computing Surveys, 13, 2 (June 1981).

[KUH 80] Kuhl, J.G., Reddy, S.M., "Distributed Fault-Tolerance For Large Multiprocessor Systems", Proc. of 7th Symp. on Computer Architecture, May 1980.

[JAF 81] Jaffe, J. M. and Moss, F.H., "A Responsive Distributed Routing Algorithm for Computer Network," Proc. The 2nd International Conference on Distributed Computing Systems, Paris, France, April 1981.

[JEN 82] Jen, C., "Overhead and Reconfigurability Considerations in the design of Distributed Computing Systems", Ph.D. Dissertation, University of California, Berkeley, 1982.

[LAM 78] Lamport, L., "Time, Clocks and Ordering of Events in a Distributed System," Comm. ACM, 21, 7 (July 1978).

[LAM 80] Lamport, L.,Shostak, R. and Pease, M. "The Byzantine Generals Problem", Research Report, Computer Science Laboratory, SRI International, March 1981.

[LAW 76] Lawler, E., *Combinatorial Optimization: Networks and Matroids*, Holt-Rinehart-Winston, 1976.

[MA 81] Ma, Y., "Techniques for the Design and Management of Dynamic Computer Networks", Ph.D. Dissertation, Univ. of California, Berkeley, 1981.

[MAH 76] Mahmoud,S. and Riordan, J.S. "Optimal Allocation of Resources in Distributed Information Networks",ACM Trans. Database Systems,March

1976.

[MCQ 74] McQuillan, J., "Adaptive Routing Algorithms for Distributed Computer Networks," Bolt, Beranek and Newman Inc., Cambridge, MA, Report No. 2831, May 1974

[MCQ 80] McQuillan, J., and et al., "The New Routing Algorithm for the Arpanet," IEEE Trans. on Comm., May 1980.

[MEN 79] Menasce, D.A. and Muntz, R.R., "Docking and Deadlock Detection in Distrubuted Databases," IEEE Trans. Software Eng. SE-5, 3 (May 1979).

[MER 79] Merlin, P.M., and Segall, A., "A Failsafe Distributed Routing Protocol," IEEE Trans. on Comm., Sept. 1979.

[MER 79b] Merlin, P.M., "Specification and Validation of Protocols", IEEE Trans. Commm., Vol COM-27, No 11, Nov. 1979.

[MOS 79] Moss, F. H. and Merlin, P.M., "A Routing Scheme for Session Oriented Stored and Forward Computer Networks," Proc. of IEEE National Telecommunications Conference, 1979.

[OBE 82] Obermarck, R., "Distributed Deadlock Detection Algorithm", ACM Trans. Database Systems, June 1982.

[PET 77] Peterson, J.L., "Petri Nets", Computing Surveys, Vol 9, No 3, Sept. 1977.

[RAM 82a] Ramamoorthy, C.V. and Ganesh,S.L. "Global Information Management", UCLA Packet Radio Analytical Workshop, Aug. 1982.

[RAM 82b] Ramamoorthy, C.V. and Tsai, W.-T., "Update Protocols for Hierarchical Routing Algorithms," UCLA Packet Radio Analytical Workshop, Aug. 1982.

[RAM 82c] Ramamoorthy, C.V. and Dong, S.T., "On Modeling and Validation of Communication Protocols", Proc. of Int'l Computer Symp., Taiwan, R.O.C., Dec. 1982.

[RAZ 80] Razouk, R., Estrin, G., "Modeling and Verification of Communication Protocols in SARA : The X.21 Interface", IEEE Trans. Computer, Vol C-29, No 12, Dec. 1980.

[ROT 77] Rothnie,J.B. and Goodman,N.A. "A Survey of Research and Development in Distributed Data Management",Proc. Conf. Very Large Database Systems, Tokyo, Japan,Oct. 1977.

[SCH 80] Schwawrtz, M., and Stern, T. E., "Routing Techniques Used in Computer Communication Networks," IEEE Trans. on Comm., April 1980.

[SCH 81] Schwartz, R.L., Melliar-Smith, P.M., "Temporal Logic Specification of Distributed Systems", Proc. 2nd Int'l Conf. on Distributed Computing Systems, April 1981.

[SEG 81] Segall, A., "Advances in Verfiable Failsafe Routing Procedures," IEEE Trans. on Comm., April 1980.

[SEG 82] Segall, A., "Decentralized Maximum-Flow Protocols," Networks, Vol. 12(1982) 213-230.

[SKE 81] Skeen, D.,"Non-blocking Commit Protocols",SIGMOD International Conf. on Management of Data, Ann Arbor, Michigan, 1981.

[SUN 79] Sunshine, C., "Formal Techniques for Protocol Specification and Verification," IEEE Computer, Vol. 12, No. 9, Sept. 1979.

[TAJ 77] Tajibnapis, W. D., "A Correctness Proof of a Topology Information Maintenance Protocol for a Distributed Computer Network," CACM, July 1977.

[TEN 78] Teng, A.Y., Liu, M.T., "A Formal Approach to the Design and Implementation of Network Protocols", Proc. COMPSAC, Nov. 1978.

[TSA 82] Tsai, W.-T., "Routing Techniques for Dynamic Computer Networks," M.S. Report, Computer Science Division, University of California, Berkeley, Aug. 1982.

[WAL 81] Walter, B., "A Robust and Efficient Protocol for Checking the Availability of Remote Sites",Proc. 6th Berkeley Workshop on Distributed Data Management and Computer Networks, Asilomar, Feb. 1982.

[WEC 79] Wecker, S., "Computer Network Architecture," IEEE Computer, Vol. 12, No. 9, Sept. 1979.

[WES 78] West, C.H., "A General Technique for Communication Control Validation", IBM J. Res. Develop., Vol 22, July 1978.

[ZIM 80] Zimmermann, H., "OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection," IEEE Trans. Comm, Vol. COM-28, No. 4, April 1980.


DATE:
ILME