

Dynamic Configuration Management in Reliable Distributed Real-Time Information Systems

K.H. (Kane) Kim, *Fellow, IEEE*, and Chittur Subbaraman

Abstract—Large-scale information systems emerging in challenging application fields must meet the high standards of reliability, maintainability, and service interruption bound requirements. Their operations are entirely, or partially, of the distributed real-time data object manipulation type. A new architecture for such systems is presented in this paper. The original aspects of the architecture are mainly in two parts: 1) the time-triggered message-triggered object (TMO) structuring of the middleware and the application software of distributed real-time information systems; and 2) the dynamic configuration management subsystem (DCMS), based on the supervisor-based network surveillance (SNS) scheme. The positive impacts of this TMO structuring on maintainability and service interruption bounds are first discussed, with distributed replicated information service systems and other systems as examples. Then, the main discussion dwells on the DCMS architecture—in particular, formal presentation of its key component: the SNS scheme. As a component of DCMS, the network surveillance (NS) subsystem enables fast learning by each interested fault-free node in the system of the faults or repair completion events occurring in other parts of the system. Currently, concrete real-time NS schemes effective in distributed systems based on point-to-point network architectures are scarce. The SNS scheme presented in this paper is a semicentralized real-time NS scheme effective in a variety of point-to-point networks. This scheme is highly scalable. An efficient implementation model for the SNS scheme is presented that can be easily adapted to various commercial operating system kernels. This paper also presents a formal analysis of the SNS scheme, on the basis of the implementation model, to obtain its strongly competitive tight bounds on the fault detection latency. Finally, some DCMS implementation issues are discussed that remain to be addressed in future research.

Index Terms—Object, distributed computing, information service systems, real time, TMO, time-triggered, message-triggered, configuration management, network surveillance, point-to-point networks, fault detection latency, latency bound, supervisor.



1 INTRODUCTION

LARGE-SCALE information systems emerging in challenging application fields such as defense, transportation management, and factory networking, must meet the high standards of reliability, maintainability, and service interruption bound requirements. Their operations are entirely or partially of the distributed real-time (RT) data object manipulation type. *RT data* are the data that are useful for relatively short periods for the given users although the length of the useful period may differ for different users. Some RT data are kept in *memory-resident RT databases* while other RT data are kept in *disk-resident RT databases* [20]. The other kind of data are stored in *archival databases* which provide information services with no guaranteed tight response time bounds.

As the complexity of these systems continues to grow, the attractiveness of structuring them in object-oriented (OO) forms is also increasing. This is because of the modularity, generality, and natural abstraction benefits that

OO approaches bring in. So far, such OO structuring attempts have not reached their full potential due to some limitations of the conventional object structuring scheme. In particular, the lack of features for explicit treatment of timing characteristics in computing operation design is the main limitation of the current widely practiced object structuring scheme [4].

Another important requirement that almost all complex information systems must meet is the RT fault tolerance capability. The interruption in the information system services due to component failures must be strictly limited. For example, if a processing node equipped with CPU, memory, and I/O interfaces, becomes unusable, then it must be detected quickly and the software objects hosted on that node must be reactivated on other healthy node(s). This *dynamic configuration management* must be done in a timely fashion without violating the *service interruption bound* requirement. Ideally no effective interruptions in services must occur. Similarly, if an inter-node link fails, an appropriate dynamic configuration action that may involve not only a communication network reconfiguration but also a software reconfiguration, must take place. Sometimes, new components are inserted into the distributed system in operation or previously failed components are repaired and reinserted into the system. These components must be recognized and brought into the operating system

- K.H. Kim is with the Department of Electrical and Computer Engineering, University of California at Irvine, Irvine, CA 92697. E-mail: kane@ece.uci.edu.
- C. Subbaraman is with the Microsoft Corporation, 1 Microsoft Way, Redmond, WA 98052. E-mail: chitturs@microsoft.com.

Manuscript received 20 Mar. 1998; revised 15 Aug. 1998.

For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 108410.

configuration by the *dynamic configuration management subsystem* (DCMS), again in manners meeting the service interruption bound requirement.

The purpose of this paper is two-fold. The first is to present an architecture for distributed RT information systems (DiRTISS) that addresses the two major issues mentioned above, i.e., pervasive application of OO structuring and timeliness-guaranteed dynamic configuration management. To fully realize the potential of OO structuring in DiRTISS, we propose to use the recently developed extension of the widely practiced basic object structuring scheme, called the *time-triggered message-triggered object* (TMO) structuring scheme [11], [14]. A set of fundamental types of RT data server objects and their replicas that can be efficiently structured by use of the TMO scheme are presented. This set is a sufficiently powerful set for handling all conceivable practical types of complex information service systems. Active objects that are users of RT data but do not provide passive data services are also specific instances of TMOs. Therefore, use of the TMO scheme achieves to a great extent the uniformity in structuring complex information systems.

A key component of DCMS is the *network surveillance* (NS) subsystem which enables fast learning by each interested fault-free node in the system of the faults or repair completion events occurring in other parts of the system. Currently concrete RT NS schemes effective in distributed systems based on point-to-point network architectures are scarce [10]. Presenting a new RT NS scheme called the *supervisor-based NS* (SNS) scheme is the other purpose of this paper. The SNS scheme is a semicentralized RT NS scheme effective in a variety of point-to-point networks. This scheme is highly scalable and can be implemented entirely in software using commercial off-the-shelf (COTS) components without requiring any special-purpose hardware

support. An efficient implementation model for the SNS scheme which can be easily adapted to various commercial operating system kernels is presented. A prototype implementation has been used together with a TMO-structured nontrivial defense application prototype to demonstrate the capability of the scheme. This paper also presents a formal analysis of the SNS scheme on the basis of the implementation model to obtain its strongly competitive tight bounds on the fault detection latency.

Section 2 presents a brief overview of the TMO structuring scheme. In Section 3, the new DiRTISS architecture that is based on the TMO structuring scheme and contains a new timeliness-guaranteed DCMS is presented. The new NS scheme, the SNS scheme, is then described in Section 4. Section 5 presents an implementation model of the SNS scheme and an analysis of the fault detection latency bounds. The paper concludes with Section 6.

2 BACKGROUNDS ON THE TMO STRUCTURING SCHEME

The *time-triggered message-triggered object* (TMO), formerly called the RTO.k object, has been developed in recent years to support rigorous and cost-effective development of RT distributed systems. Based on the initial abstract framework formulated in late 1980s, a concrete syntactic structure associated with precise execution semantics was developed in recent years [11], [14]. The basic structure of a TMO is depicted in Fig. 1. It is an extension of the conventional basic object structure and most important and unique extensions are summarized below:

- 1) Time-triggered spontaneous methods (SpMs) clearly separated from the message-triggered service methods (SvMs):

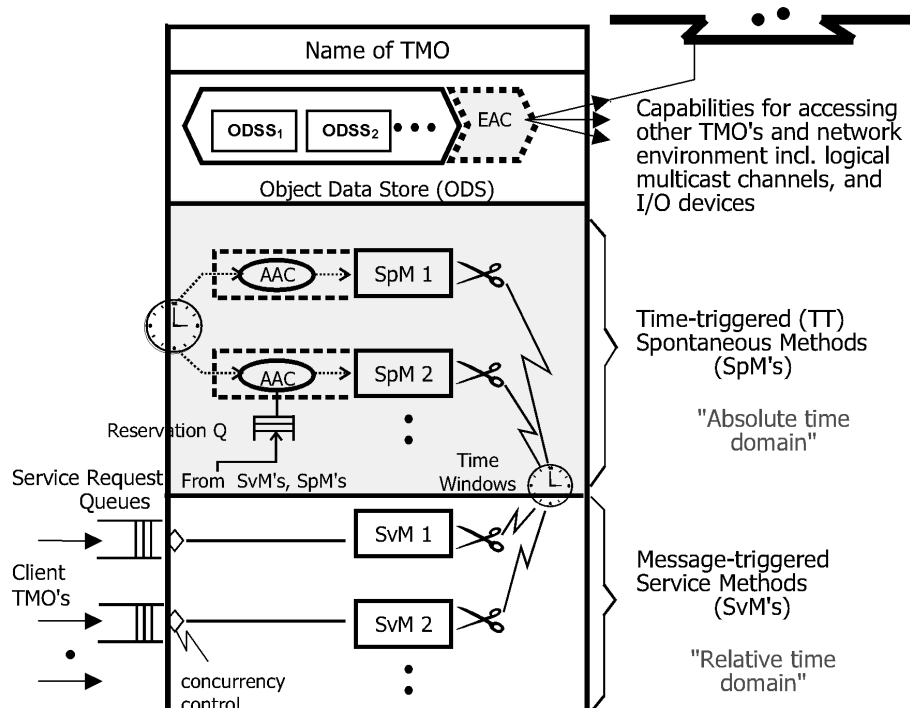


Fig. 1. Structure of the time-triggered message-triggered object (TMO) (adapted from [14]).

For some methods of a TMO, an RT clock serves as the mechanism for triggering the method executions as the clock reaches some values specified at design time and such methods are called *time-triggered (TT-) methods*, also called the *spontaneous methods (SpMs)*, and clearly separated from the conventional *service methods (SvMs)* triggered by messages from clients. The two types of methods in a TMO are different not only in the way their executions are triggered but also in that

“actions to be taken at real times *which can be determined at the design time* can appear only in SpMs.”

Therefore, actions of the type “at constant-clock-value do S” or the type “sleep-until constant-clock-value” can appear only in SpMs. Triggering times for SpMs must be fully specified as constants during the design time. Those RT constants appear in the first clause of an SpM specification called the *autonomous activation condition (AAC)* section. An example of an AAC is

“for t = from 10 a.m. to 10:50 a.m. every 30 min
start-during (t, t + 5 min) finish-by t + 10 min”

which has the same effect as

{“start-during (10 a.m., 10:05 a.m.) finish-by 10:10 a.m.”,
“start-during (10:30 a.m., 10:35 a.m.) finish-by 10:40 a.m.”}

A provision is also made for making the AAC section of an SpM contain only candidate triggering times, not actual triggering times, so that a subset of the candidate triggering times indicated in the AAC section may be dynamically chosen for actual triggering. Such a dynamic selection occurs when an SvM (or another SpM) within the same TMO requests future executions of a specific SpM.

2) Basic concurrency constraint (BCC):

Incorporation of SpMs means introducing the potential for the following two new types of concurrent executions of object methods in addition to the potential for concurrent executions of SvMs that exist in conventional objects:

(Type I) *Concurrency among SpM executions*: This concurrency is specified in an implicit but natural manner, e.g., two SpMs to be triggered at 10 a.m.

(Type II) *Concurrency between SpM executions and SvM executions*.

In order to dramatically reduce the designer’s efforts in guaranteeing timely service capabilities of TMOs, the execution rule which prevents conflicts between SpMs and SvMs is incorporated. Basically, activation of an SvM triggered by a message from an external client is allowed only when potentially conflicting SpM executions are not in place. To be exact, when a message-triggered SvM is not free of conflict with an SpM in accessing the same portion of the *object data store (ODS)*, execution of the former method (SvM)

must not be allowed in a time zone earmarked for a TT-execution of the latter method (SpM). This restriction is called the *basic concurrency constraint (BCC)*. Therefore, executions of SpMs are not disturbed by SvM executions and triggering times of SpMs are fixed at the design time. If a statement of the type “at 10am do S” appears in an SpM, its timely execution can be easily assured.

3) Ordered isolation (OI) rule (option):

Unlike the above two features, this is an optional feature of the TMO structuring scheme. It also helps in easing the design-time guaranteeing of timely services. The term *initiation timestamp* or *I-timestamp* is defined as follows. In the case of an SvM execution, the *I-timestamp* is defined as the record of the time instant at which the execution engine initiated the SvM execution after receiving the client request for the SvM execution and ensuring that the SvM execution can be initiated without violating the BCC and other execution rules. In the case of an SpM execution, the *I-timestamp* is defined as the record of the time instant at which the SpM execution was initiated according to the AAC specification of the SpM. Also, a segment of the object data store (ODS), called an *ODS segment (ODSS)*, is a basic unit of data storage which can be reserved for exclusive access by a method of a TMO. The *ordered isolation rule* has two parts:

(OI-1) A method execution with an older I-timestamp must not be waiting for the release of an ODSS held by a method execution with a younger I-timestamp.

(OI-2) In addition, a method execution may never be rolled back due to an ODSS conflict.

The clear separation between SpMs and SvMs and the BCC make the TMO model clearly distinguished from other proposed RT object models [1], [7], [21]. In addition, the TMO specifies a *time window* for each output action and completion event of a method, which is a feature not found in the conventional basic object model but adopted in different forms in all discussions on an RT extension of the basic object model.

An underlying design philosophy of the TMO scheme is that an RTCS will always take the form of a network of TMOs. The designer of each TMO provides a guarantee of timely service capabilities of the object by indicating the *time window for every output* produced by each SvM (and each SpM which may be executed on requests from SvMs) in the specification of the SvM (and some relevant SpMs) advertised to the designers of potential client objects. Before determining the time window specification, the server object designer must convince himself/herself that with the *object execution engine* (hardware plus operating system) available, the server object can be implemented to always execute the SvM such that the output action is performed within the time window. Again, the BCC contributes to major reduction of these burdens imposed on the designer.

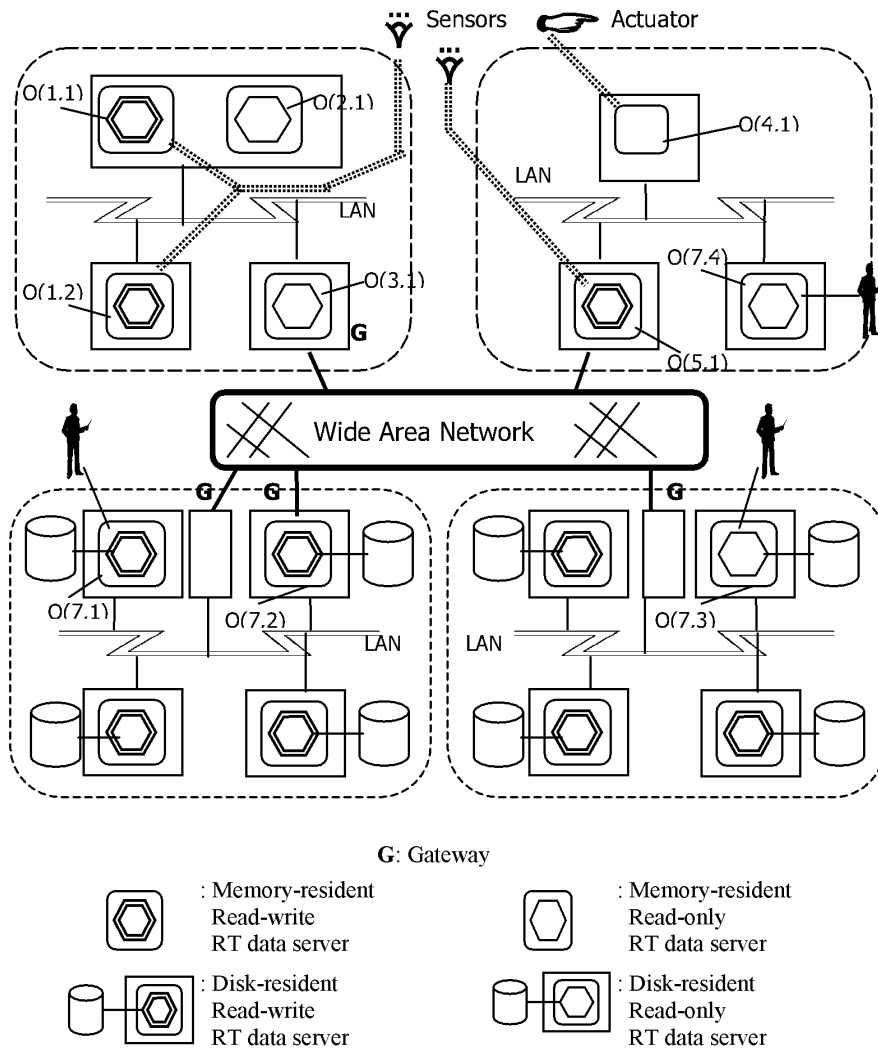


Fig. 2. The proposed architecture for distributed real-time information systems.

The TMO model is effective not only in the multiple-level abstraction of RT (computer) control systems under design but also in the accurate representation and simulation of the application environments. In fact, it enables uniform structuring of control computer systems and application environment simulators [14] and this presents considerable potential benefits to the system engineers.

3 A TMO BASED ARCHITECTURE FOR DISTRIBUTED REAL-TIME INFORMATION SYSTEMS

A high-level view of the proposed architecture for distributed real-time information systems (DiRTISs) is depicted in Fig. 2.

3.1 Fundamental Types of RT Data Server Objects and TMO Network Structuring

The architecture makes provision for incorporating the following fundamental types of RT data server objects.

(Type 1a) Disk-resident read-write RT data server objects

An example of a disk-resident read-write RT data server object is $O(7, 1)$ in Fig. 2. This object

encapsulates the data in the attached disk and serves as an interface to the users, using its ODS (in memory) effectively as a conduit. Therefore, method completion times of objects implementing this RT database are addressed in terms of 10 milliseconds or larger time units. Worst-case disk I/O times must be well understood by the system engineers. The object serves data read and/or write requests from many types of clients. Some clients are interactive human users and others are objects distributed over the same local area network (LAN) or the wide area network (WAN).

(Type 1b) Disk-resident read-write RT data server replicas

One replica of $O(7, 1)$ is $O(7, 2)$ and it is in the same LAN. The purpose of having this kind of a replica is to enhance the reliability and availability. One of the most soundly established type of replication is the primary-shadow replication [10], [13]. The *primary-shadow TMO replication* (PSTR) scheme has been developed and demonstrated via a prototype implementation [13]. Under such an implementation, $O(7, 2)$ does not interact with clients while $O(7, 1)$ is

healthy. $O(7, 2)$ receives copies of all external inputs from the LAN and tries to keep its computation state up-to-date so that if $O(7, 1)$ crashes, it may take over the primary server role quickly. Worst-case latencies incurred by both computing components and communication components must be well understood in order to realize reliable RT data server replicas.

(Type 1c) Disk-resident read-only data server replicas

Another replica of $O(7, 1)$ is $O(7, 3)$ and it is in another LAN. It interacts with clients while $O(7, 1)$ interacts with different clients. It functions as just a *read-only data server agent* which serves nearby read-only clients, primarily interactive human clients. $O(7, 1)$ and $O(7, 3)$ cooperate to let $O(7, 3)$ carry acceptably up-to-date RT data.

(Type 2a) Memory-resident read-only data server replicas

$O(7, 4)$ is also a replica of $O(7, 1)$ in another LAN which is a read-only data server agent. A unique characteristic of this agent is that it is a memory-resident data server object. It keeps a partial copy of the data in $O(7, 1)$. If a client requests data items not kept in the ODS of the $O(7, 4)$, then $O(7, 4)$ must obtain the data item from $O(7, 1)$ through interaction over the WAN.

(Type 2b) Memory-resident read-write RT data server objects

$O(1, 1)$ in Fig. 2 is an example of a memory-resident read-write RT data server object. Method completion times of this type of objects are addressed in terms of milliseconds or smaller time units. $O(1, 1)$ serves clients which are other objects in the system. It is also connected to a sensor. So, it provides a sensor database. An intelligent sensor can also be viewed as a client of $O(1, 1)$.

(Type 2c) Memory-resident read-write RT data server replicas

A replica of $O(1, 1)$ is $O(1, 2)$ in the same LAN. The relationship between $O(1, 1)$ and $O(1, 2)$ is a primary-shadow relationship facilitating RT recovery.

Another replica of $O(1, 1)$ is $O(5, 1)$ in a different LAN. $O(5, 1)$ does not mechanically replicate the functionalities of $O(1, 1)$ but it is explicitly designed to serve as a functional backup for $O(1, 1)$, using its own sensor. In other words, unlike $O(1, 2)$ which is mechanically created by the tool without burdening the application designer, $O(5, 1)$ must be designed together with $O(1, 1)$ in a coordinated fashion. Explicit application-specific design efforts are needed to ensure the state consistency between $O(1, 1)$ and $O(5, 1)$.

(Type 2d) Memory-resident read-only RT data server objects

$O(2, 1)$ is a memory-resident read-only RT data server object. Its clients are other objects such as $O(3, 1)$. $O(2, 1)$ can obtain its data from other objects as a part of serving a request from a read-only client. Or if it is

a TMO containing SpMs, it can obtain data from other objects through prescheduled executions of its SpMs.

(Type 3) Active objects providing no data services to clients

$O(4, 1)$ is an active object such as a TMO with SpMs. It does not provide passive data services to any client but it commands an actuator. It is also a client (often read-only) of the data services from some RT data server objects such as $O(1, 1)$.

In the proposed architecture, all these types of objects are structured as TMOs. The advantages of this structuring include:

- 1) Efficient and reliable design of servers with accurate timings: This is a consequence of the careful design of action timings and the thorough understanding of the timings of the execution engines that are forced and aided by the TMO scheme.
- 2) State synchronization of replicated RT data server objects: This becomes much easier with the TMO replication schemes such as the PSTR scheme [13].
- 3) Easy and accurate determination of the worst-case load imposed on the executing node by a group of TMOs.

In addition, uniform structuring of all types of RT data server objects and their replicas discussed in this section, which is enabled by the TMO scheme, is a major benefit of the proposed DiRTIS architecture.

3.2 Dynamic Configuration Management Subsystem (DCMS)

Fig. 3 depicts the DCMS part of the proposed DiRTIS architecture. As shown, DCMS is a distributed software subsystem and it is structured largely in three layers:

- 1) *network surveillance (NS) subsystem*,
- 2) *network reconfiguration (NR) subsystem*, and
- 3) *object distribution (OD) subsystem*.

As mentioned before, the NS subsystem enables fast learning by each interested fault-free node in the system of the faults or repair completion events occurring in other parts of the system. NS schemes thus facilitate each healthy node in the system to have *practically complete* and *timely* knowledge of the health conditions of other nodes in the system as well as the health conditions of various components of the interconnection network. Once the need for a change in the network configuration (e.g., a node or link crash, a node rejoin) is detected by the NS subsystem, then the NR subsystem establishes a network of healthy nodes and healthy links. The NR subsystem consists of the NR supervisor and one or more NR workers distributed among different nodes. Whenever the NR subsystem changes the network configuration and whenever new application TMOs need to be created, the OD subsystem establishes a new distribution of TMOs among the healthy nodes.

Note that this DCMS organization is amenable to hierarchical extension. In a large-scale distributed system, it is

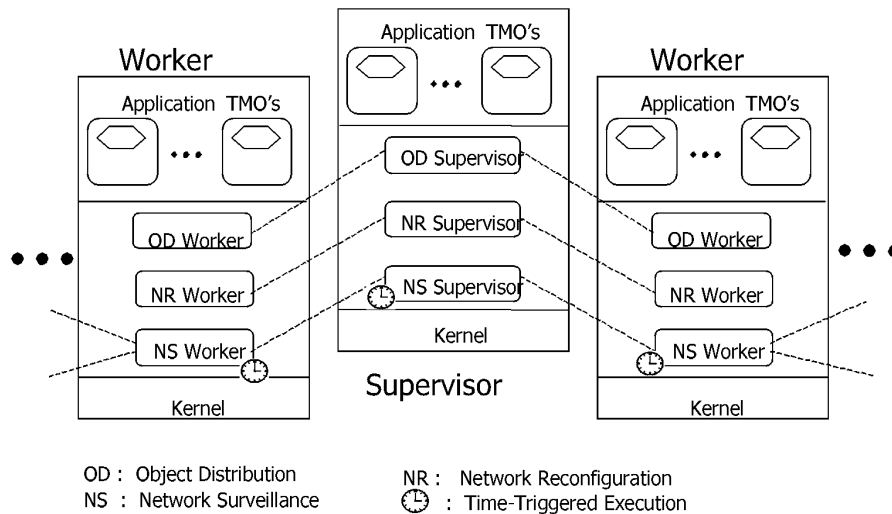


Fig. 3. The dynamic configuration management subsystem (DCMS) in the proposed architecture.

natural to establish a WAN-wide DCMS coordinating multiple LAN-wide DCMSs.

From the viewpoint of realizing a DCMS with verified properties, the most challenging component is the NS subsystem. In the past, quite a bit of research has been conducted in the area of NS, also known as *group membership maintenance* [3], [5], [6], [8], [9], [10], [15], [17], [18], [19], under a variety of assumptions. However, only a few schemes were devised for use in RT computer systems. In safety-critical RT distributed computing applications, the *worst-case fault detection latency* is a measure of critical importance. Most of the practical schemes that have been built or put to experiments so far for use in ultrareliable RT systems [15], [10] require bus-based LAN-type communication architectures in which the broadcast is not much more expensive in time cost than the point-to-point message communication is. Thus, relatively little research efforts have gone into developing concrete NS schemes for point-to-point network architectures in RT application environments. Point-to-point networks are typical interconnection structures not only in multicampus or wide area networks but also in highly parallel multicomputer networks.

In the remainder of this paper, a new RT NS scheme effective in a variety of point-to-point networks is presented.

4 THE SUPERVISOR-BASED NETWORK SURVEILLANCE (SNS) SCHEME

4.1 Network Types and Fault Model

As illustrated in Fig. 2, a system may consist of multiple LANs. Each LAN contains one or more gateway nodes. Within a LAN, the local nodes may be connected with one another either via a broadcast bus or via a point-to-point network. In this paper, we assume that all nodes communicate among themselves in point-to-point fashion. A further optimization of this scheme for systems where some nodes communicate via cheap broadcasts is not discussed in this paper.

Two nodes are said to be *neighbors* of each other if they are connected by a direct point-to-point link. Every message sent from a source node to a non-neighbor destination node is stored for a while in each intermediate node while the node makes a routing decision. Such a routing scheme is commonly known as a *store-and-forward* routing scheme.

4.1.1 Fault Sources

Since the number of components in a typical point-to-point network architecture is large, a clear and yet accurate representation of all possible fault sources is a challenging issue. The most practical approach here is to group various potential fault sources into a manageable set of categories.

In the model depicted in Fig. 4, possible sources of faults in a node are represented by a *processor*, an *incoming communication handling unit* (I-unit), and an *outgoing communication handling unit* (O-unit). *Faults in the processor* represent faults in the executing software that could cause the node to crash, faults in the processor hardware, faults in memory modules, etc. [2]. *Faults in the I-unit* represent the

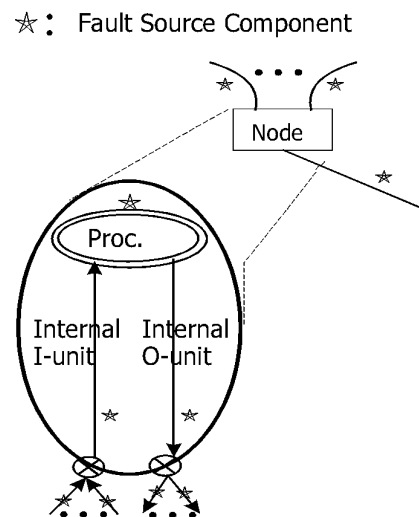


Fig. 4. Fault sources.

faults in various components of a node (both hardware and software) that are involved in receiving a message from the network. *Faults in the O-unit* represent faults in various node components (both hardware and software) that are involved in sending a message to the network. The messages sent or received by a node here include both the messages that are destined for the node itself as well as the messages that are stored in the node temporarily before being routed off to other destinations. Any observed fault of a node could be an instance of a combination of faults in the fault sources mentioned above. The *remaining fault source* is the point-to-point interconnection network. Faults in the interconnection network represent faults in one or more links of the interconnection network.

In the remaining discussion, we call each of the four fault sources described in the fault source model above as a *fault source component*. We assume that since the routing scheme is of the store-and-forward type, a permanent failure of any one of the fault source components in a node disables not only the node's processing capabilities but also the node's routing capabilities.

4.1.2 Fault Types Covered and Fault Frequencies Assumed

The SNS scheme has been designed to detect various types of faults that occur in various fault sources mentioned in Section 4.1.1 subject to the fault frequency assumptions stated below.

- (A1) The fault-source components in each node do not generate messages containing erroneous values nor untimely messages.
- (A2) Each of the nodes performing store-and-forward functions (as well as the source node) transmits each stored message twice continuously. It is assumed that this makes the probability of transient faults in the components of the two neighbor nodes and transient faults in the link between the two neighbor nodes causing message losses to be

negligible. This *dual redundant transmission* approach was also exploited in [15].

- (A3) Let $S_N = \{\text{Processor, I-link, O-link}\}$, i.e., the set of components in node N , and $LL(P, Q) = \{\text{Link connecting two neighbor nodes } P \text{ and } Q\}$. The time interval from the time of the occurrence of a permanent fault F in S_N or in $LL(P, Q)$ to the time every healthy node in the system learns of the fault occurrence is the *detection period* of the SNS scheme for the fault F . It is assumed that during the detection period, no other fault occurs in node M where $M \neq N$, or in $LL(I, J)$, where $(I, J) \neq (P, Q)$. The *worst-case detection latency* of the SNS scheme for all possible hardware faults in the system is denoted by WDL . Therefore, it is assumed here that no second permanent hardware fault occurs in the system within WDL time units after the occurrence of the first permanent hardware fault F .

In addition, the SNS scheme imposes the following on the system designer.

- (A4) The topology of the point-to-point network used, the nature of the application, and the task relocation arrangement must be such that the node or link failures occurring within the bounds of A1, A2, and A3 cannot lead to permanent partitioning of the application system into inoperable disconnected subsystems during the lifetime of the application mission.
- (A5) The clocks in the nodes are kept synchronized sufficiently closely for practical purposes, i.e., for the given applications. Global Positioning System-based approaches and other cheaper high-precision approaches have become available in recent years.

4.2 Major Operating Rules of the SNS Scheme

Fig. 5 shows the basic operations of the SNS scheme. As shown in the figure, there are two types of nodes that execute the SNS scheme, the *worker* nodes and a *supervisor* node. The worker nodes are mainly responsible for judging

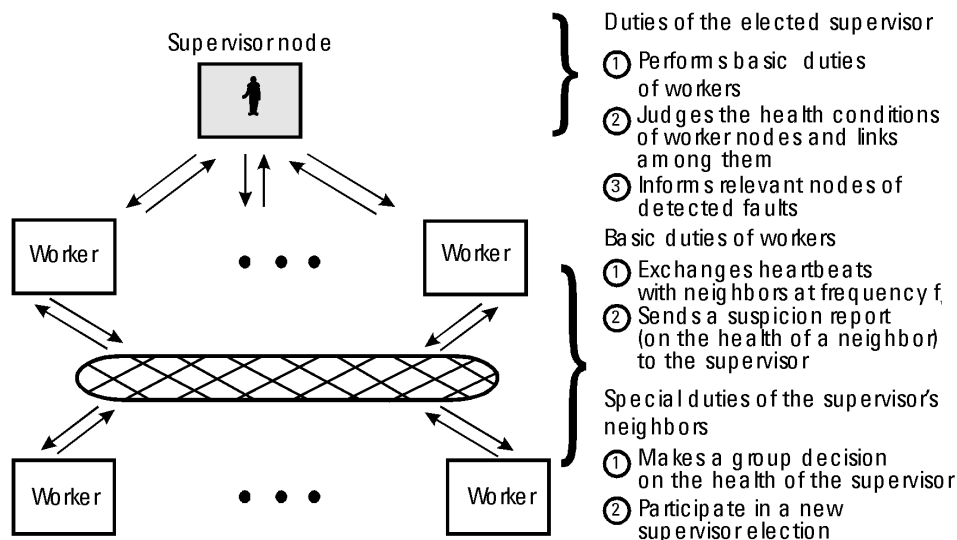


Fig. 5. Major operating rules of the supervisor-based network surveillance (SNS) scheme.

their own health status, the health status of their neighbor nodes, and the health status of the links attached to themselves. The supervisor node performs all the duties that a worker normally does. In addition, it is responsible for collecting *fault suspicion reports* from worker nodes, judging on the basis of the collected information whether a fault has indeed occurred, and then sending the fault occurrence notice to all the healthy worker nodes in the system, if necessary.

The SNS scheme is executed by a set of nodes S_{NS} that consists of n worker nodes including a supervisor node. Under the normal mode of operation, each node in S_{NS} will have at least two healthy neighboring nodes. If the number of neighbors of a node p in S_{NS} drops down to one, then p is treated as a component of its neighbor node q . Thereafter, q and its neighbors are responsible for detecting the faults in p . A subset of S_{NS} executes some distributed RT application tasks. In Fig. 2, S_{NS} would be all the nodes or a subset of them.

Under the SNS scheme, *two copies* of every message are sent from a source node to a destination node, the first copy along one path P_1 and the second copy along a disjoint alternate path P_2 . Since it is assumed that

- 1) transient faults are masked by the dual redundant transmission approach (assumption A2),
- 2) permanent faults cannot occur in more than one node or link during the period of a message transmission which is much shorter than WDL (assumption A3), and
- 3) node or link failures do not lead to permanent partitioning of the application system during the lifetime of the application mission (assumption A4),

it is always possible for the source node to find P_1 and P_2 such that at least one copy of the message sent will definitely reach the destination node as long as the destination node remains healthy to receive it.

4.2.1 Basic Duties of Worker Nodes

All the healthy worker nodes in the system perform the following basic duties:

- 1) Every healthy worker node in the system periodically exchanges *heartbeat* signal messages with each of its healthy neighbors at the interval of π .
- 2) Absence of a heartbeat signal from a neighbor node within a specific deadline will raise the suspicion of the worker node. The suspecting node then attempts to find out the location of the fault at the level of the fault source components modeled in Section 4.1. Here a node cannot distinguish between a permanent fault in the O-unit in its neighbor node from a permanent crash of the processor in that node.
- 3) Once the worker node locates the fault source component to the extent it can, it sends a *fault suspicion report* to the supervisor.

4.2.2 Duties of the Supervisor

The supervisor node performs the following additional duties:

- 1) Once the supervisor receives a fault suspicion report from a worker node, it judges whether the suspicion is indeed true. For this, the supervisor may use the fact that it has received a certain type of fault suspicion reports from $k > 1$ worker nodes.
- 2) After confirming the fault, the supervisor proceeds to inform the relevant nodes of the detected faults.

4.2.3 Special Duties of the Supervisor's Neighbor Nodes

The supervisor's neighbor nodes also perform the following additional duties:

- 1) Make a group decision about the health of the supervisor,
- 2) In case the current supervisor is judged to be faulty, participate in a new supervisor election in which each of the current supervisor's healthy neighbors takes part, and
- 3) Once one among the old supervisor's neighbors is elected as a new supervisor, the newly elected supervisor informs all the healthy worker nodes about the fault in the old supervisor as well as the assumption of its new supervisor role.

Clearly, this SNS scheme which takes advantage of the concurrency in message communication yields much better fault detection latency bounds than any approach based on logical ring organization of the nodes.

4.3 Worker Node Protocol

The protocol for a worker node (which is not a neighbor of the supervisor) is now presented in detail.

Notation

- (N1) Given a node X in the system, $r(X)$ is any one of its neighbor nodes. $LL(X, r(X))$ represents the direct two-way link between X and $r(X)$.
- (N2) π : The interval at which a node sends heartbeat signals and analyzes the heartbeat signals received from its neighbors.
- (N3) $h_{X,Y,m}^i$: The event of node X sending the m th copy of the i th heartbeat to node Y . Here m is in the range $[1, 4]$ because every message is sent over two different paths and sent twice over each link.
- (N4) a_X^i : The event of node X starting the analysis of the i th round of heartbeat signals received from each of its neighbors.
- (N5) HA : The worst-case execution time of the analysis of heartbeat signals received in one round.
- (N6) $t(e)$: Time at which event e occurs. So,

$$t(h_{X,Y,4}^{i-1}) = t(h_{X,Y,4}^i) - \pi.$$

- (N7) MD : The *worst-case message transmission delay* from one node to another including the dual redundant transmission time and the time for sending out two copies.

- (N8) SR : The *synchronization range*, i.e., the maximum deviation among the heartbeat generation times of different nodes in each round, which is also assumed to be equal to the maximum deviation among the analysis start times of different nodes in each round. Always $SR < MD$ and often $SR \ll MD$.

The interval π , the heartbeat generation times, and the analysis times must be chosen such that for any two nodes X and Y in the system, $t(h_{X,Y,4}^i) + MD < t(a_Y^i)$, and $t(h_{X,Y,4}^i) + MD < t(a_X^i)$. π is of course greater than MD . On the other hand, $t(a_X^i)$ for any X should be as early as possible in order to facilitate fast detection of faults.

It is also desirable to have $t(a_X^i) + HA < t(h_{X,Y,1}^{i+1})$ and $t(a_Y^i) + HA < t(h_{X,Y,1}^{i+1})$ because they lead to simple analysis of the heartbeat signals received, although it is not mandatory. This means that it is desirable to have $\pi > MD + HA + SR$. As will be shown below, $\pi < WDL$, and π is a major factor impacting WDL .

Fig. 6 depicts the protocol executed by a worker node as a part of its basic duties. The protocol can be broadly divided into three phases. As mentioned above, Phase 1 of the i th round precedes Phase 2 of the i th round by MD time-units but Phase 1 of the $(i + 1)$ th round may be in overlap with Phase 2 of the i th round. Phase 3 is also executed periodically and in each round, it can be executed either before or after Phase 2.

(Phase 1) Periodic sending of heartbeat signals to each neighboring node: Every healthy node X periodically sends at the interval of π a heartbeat signal to each of its healthy neighbors, $r(X)$, four times continuously; the first two times along the direct link K connecting X and $r(X)$ (only if K is healthy), and the next two times along an alternate path that is found by X and does not involve K .

(Phase 2) Periodic analysis of the heartbeat signals received from neighbors: Every healthy node X periodically analyzes at the interval of π the heartbeat signals received from the neighbors during the most recent period of 2π duration. This analysis may reveal faults as follows:

(Case 2a) Processor crash and/or permanent fault in the O-unit in $r(X)$: Suppose all neighbor nodes of node X have been known to X by $t(a_X^i)$ as healthy nodes. If node X has received by time $t(a_X^i)$ no copy of the heartbeat signal generated from a certain neighbor node $r(X)$ during the i th round, i.e., none of the four copies, $h_{r(X),X,j}^i$, where $1 \leq j \leq 4$, but has received at least one copy of a heartbeat signal from every other healthy neighbor node, then X can suspect node $r(X)$ of having contracted a processor crash and/or a permanent O-unit fault, in short, a PO fault. However, at $t(a_X^i)$, X cannot be certain about the PO fault in $r(X)$ because the same symptom can be observed if the signal of $h_{r(X),X,1}^i$ was to be the last to arrive at X among the first copies of the heartbeat signals generated during the i th round and X contracted a permanent fault in its I-unit, in short a PI fault, immediately before the arrival of the signal of $h_{r(X),X,1}^i$. It, thus, marks

in its own record the state of $r(X)$ as “possibly faulty” and sends a fault suspicion report to the supervisor node with an indication that it is unsure of the fault location. Fig. 7 shows two such cases which node X cannot distinguish at time $t(a_X^i)$. By $t(a_X^{i+1})$, it becomes clear to node X whether the fault source was in $r(X)$ or in the I-unit of X . If node X receives a copy of the heartbeat signal from any node between $t(a_X^i)$ and $t(a_X^{i+1})$, then node X can conclude that the absence of the signal of $h_{r(X),X,1}^i$ was due to a PO fault in $r(X)$. Node X can thus attempt to send a *follow-on report* to the supervisor node. On the other hand, suppose that a neighbor of X , $r'(X)$, has been known to X by $t(a_X^i)$ as being “possibly faulty.” If X has not received by $t(a_X^i)$ any copy of the heartbeat signal generated from another neighbor $r(X)$ during the i th round, then X must conclude that it itself contracted a PI-fault during the $(i - 1)$ th round.

(Case 2b) Permanent fault in an I-unit in X : If node X has missed by $t(a_X^i)$ all copies of the heartbeat signals generated from more than one node during the i th round, then it can immediately conclude its own PI fault. Upon reaching this conclusion, the node tries to inform the supervisor about the fault with an indication that it is sure of the fault location and initiate a *spontaneous shutdown*. Even in the case where the faulty node X fails to inform the supervisor about its PI fault, its healthy neighbor nodes can judge after X initiates a spontaneous shutdown that X has run into a PO fault as explained in Case 2a. Another case of detecting PI-faults was mentioned above in Case 2a.

(Case 2c) Permanent fault in a link $LL(X, r(X))$: Suppose node X received by time $t(a_X^{i-1})$ one copy of the heartbeat signal generated by its neighbor $r(X)$ during the $(i - 1)$ th round via the link $LL(X, r(X))$. If X has not received by $t(a_X^i)$ any copy of any heartbeat signal of the i th round via $LL(X, r(X))$ but it has received a copy of the heartbeat signal of the i th round from $r(X)$ via some other link, then it suspects $LL(X, r(X))$ to have developed a permanent fault and sends a fault suspicion report to the supervisor node with an indication that it is certain about the fault location.

(Phase 3) Periodic checking for notices from supervisor: Every healthy worker node periodically checks whether it has received any notice from the current supervisor regarding the occurrence of faults, a notice from a new supervisor regarding failure of the old supervisor, etc. Upon receiving such information, the worker node takes appropriate action such as shutting itself down, updating its appropriate records, etc.

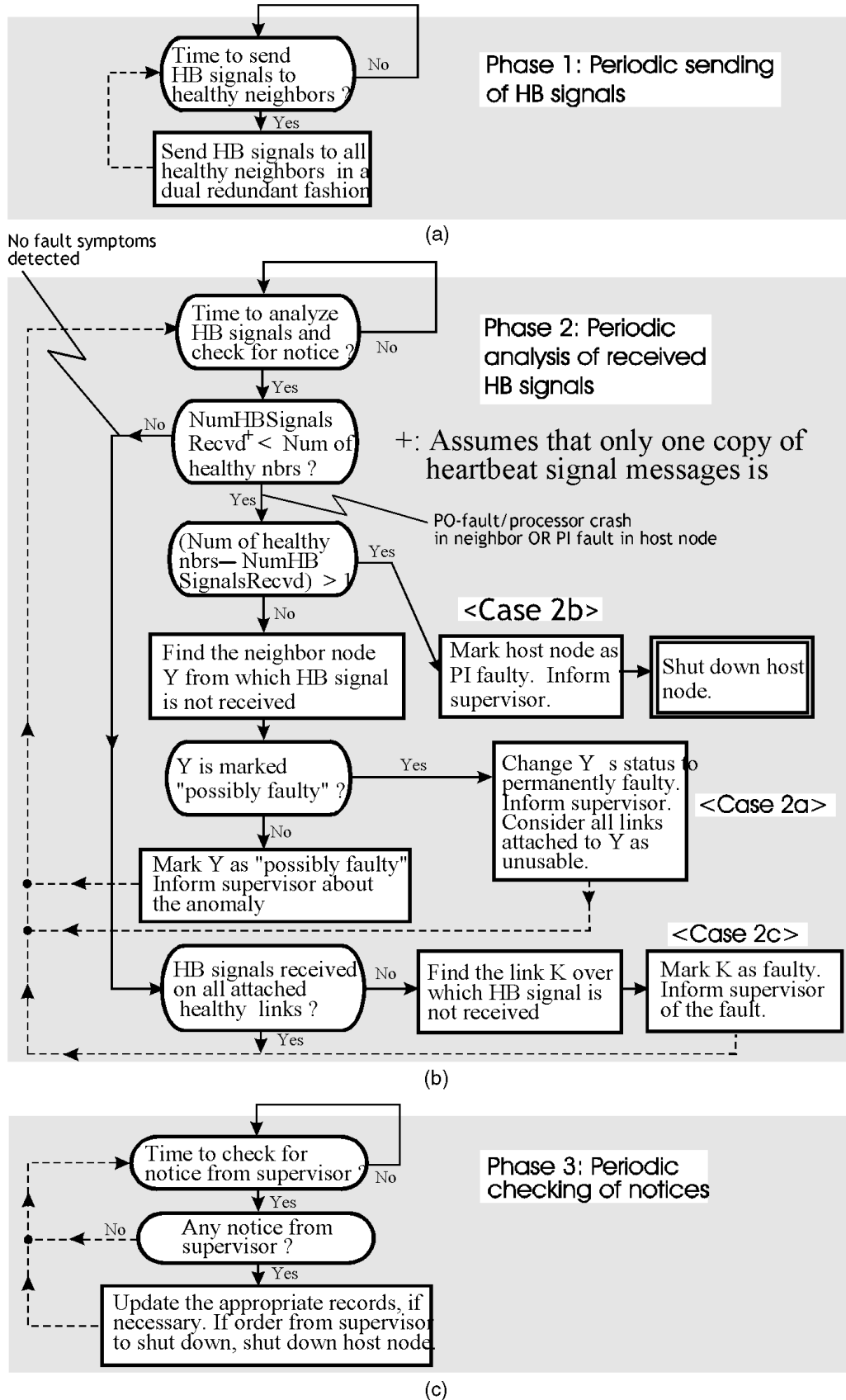


Fig. 6. The worker node protocol.

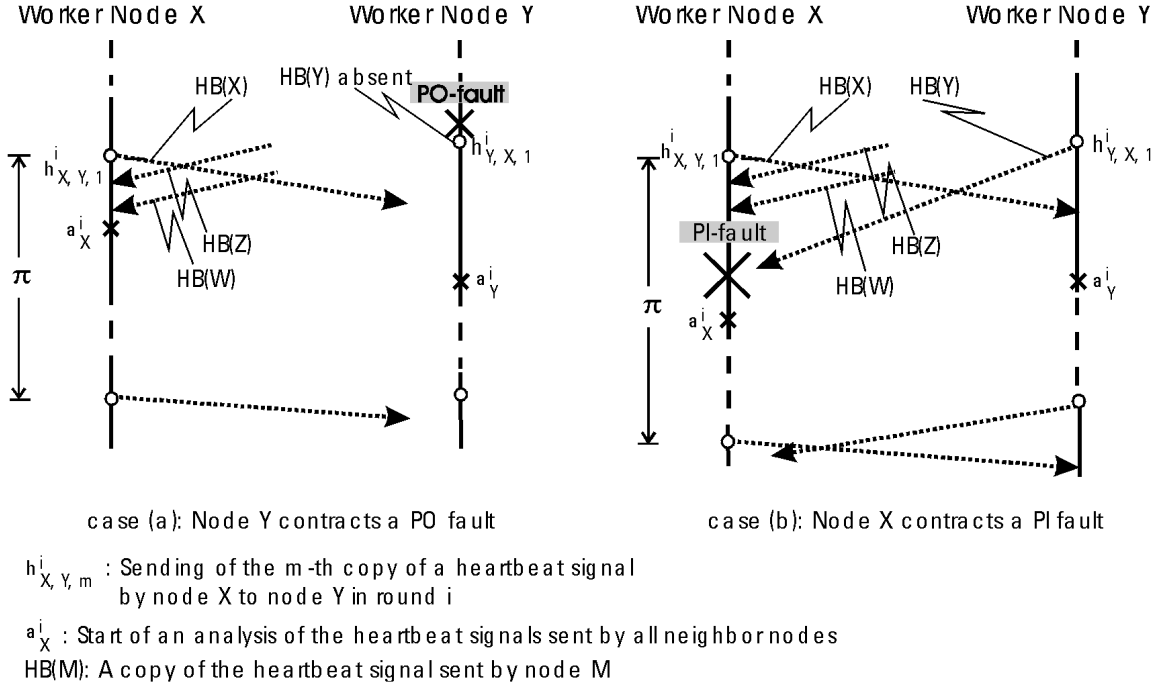


Fig. 7. Node X cannot distinguish between case (a) and case (b) at $t(a_X^i)$.

4.4 Supervisor Node Protocol

Notation

(N9) $fsra_{sup}^i$: The event of the supervisor starting the analysis of the fault suspicion reports sent by worker nodes in the i th round.

The analysis times of the supervisor node must be chosen such that for any worker node X in the system, $t(a_X^i) + HA + MD < t(fsra_{sup}^i)$. At the same time, $t(fsra_{sup}^i)$ should be as early as possible.

The supervisor node takes the following major actions in addition to the basic actions of a worker node:

- 1) Processing fault suspicion reports from worker nodes: The supervisor will in general receive the reports regarding a particular fault from more than one worker node within the interval of π except in the special cases mentioned in Section 4.3 (Cases 2a, 2b, and 2c) where only one worker sends a fault suspicion report because
 - a) it is the only node which did not receive a heartbeat signal from a certain neighbor,
 - b) it did not receive heartbeat signals from multiple neighbor nodes due to its PI fault, or
 - c) it detected the fault in a link but its neighbor connected to the link did not.

This is because each worker node will have at least two healthy neighbor nodes and no more than one node or link can become permanently faulty within the fault detection latency bound (assumption A3). The supervisor relies on these multiple reports to confirm the occurrence of a certain fault. Upon this confirmation, the supervisor multicasts a *notice of the*

fault to all the healthy nodes in the system. In the case where the supervisor receives a single fault suspicion report from a worker node which is unsure of the fault location, it waits until the next round to confirm the fault location. In the case where a worker node has indicated that it is certain about its own PI fault or a link fault, the supervisor does not need any confirming report from any other node.

- 2) Offering advice to worker nodes: Upon confirming a link fault, the supervisor issues advice to all healthy worker nodes against relying on the faulty link for sending any messages.
- 3) Relocation of worker nodes: The supervisor node notifies the NR subsystem (in Fig. 3) of the need for repairing faulty components. Whenever the number of neighbors of a healthy worker node drops down to one, the NR subsystem attempts to relocate the worker node. Due to space limit, this paper does not discuss this relocation issue further.

4.5 Election of a New Supervisor Upon a Supervisor Failure

When a supervisor node fails, the failure is detected by all the healthy neighbors of the supervisor as mentioned in Section 4.2. If a neighbor P of the supervisor node becomes suspicious of the supervisor node being faulty due to the absence of an expected arrival of a heartbeat signal from the supervisor, it sends a fault suspicion report to all the other neighbors of the supervisor. If P receives one similar fault suspicion report from another neighbor of the supervisor during the same or following round, then it can be certain that the supervisor is faulty. Otherwise, it must conclude that it itself is faulty. When the neighbor nodes are certain

about the supervisor node being faulty, they proceed to elect a new supervisor.

Let NE_S denote the set of healthy neighbor nodes of the supervisor node at the time at which the supervisor fails. Each node in the system is assigned a unique ID. Each healthy node in the system maintains a list of the IDs and the connections of all the other healthy nodes in the system with the help of the supervisor node, and this list is updated as fault detections and new node admissions take place. Once the supervisor fails, all nodes in NE_S take part in electing a new supervisor. The knowledge that each node in NE_S has on the health status of other nodes in the system is always up-to-date with the possible exception of the outdated status information about a node or link that has recently become faulty. During the election period, the knowledge is fully accurate. Each node in NE_S checks whether its ID is the minimum among all the IDs in NE_S . Only one winner will emerge from such a check. This winner will assume the new supervisor role and inform all the healthy nodes in the system of its newly assumed role. After that, the new supervisor node will proceed to arrange for the repair of the failed node. Note that once a fault in a supervisor node is detected by a node in NE_S , the subsequent process of electing a new supervisor and informing all healthy nodes of the result involves completely autonomous decisions by each node in NE_S and does not involve any kind of message exchange among the nodes in NE_S , and so it is safe to say that the election contributes nothing to the increase of WDL .

5 AN IMPLEMENTATION MODEL OF THE SNS SCHEME AND AN ANALYSIS OF THE FAULT DETECTION LATENCY BOUNDS

An efficient execution support for the SNS scheme has been designed as a new extension of the DREAM kernel, a timeliness-guaranteed kernel model developed in the authors' laboratory [12]. The design can be viewed as an implementation model of the SNS scheme and its strength is in its modular structure in which the functionality is distributed among various periodically executing *kernel-threads*, making the design easy to analyze and expand. In this section, we present the modular structure of the implementation model. The implementation approaches presented in this section can easily be adapted to various commercial operating system environments as well. On the basis of this implementation model, the fault detection latency bounds are analyzed later in this section.

5.1 Modular Structure of the Implementation Model

Fig. 8 shows the modular implementation model of the scheme. The function of a worker or supervisor node is implemented into the *network surveillance thread* (NST) and two other co-resident kernel-threads, called the *incoming communication thread* (ICT) and the *outgoing communication thread* (OCT).

All these three kernel-threads are structured as time-triggered (TT) threads. They are periodically executing threads whose execution is driven by the RT clock. This structuring eases the determination of the worst-case

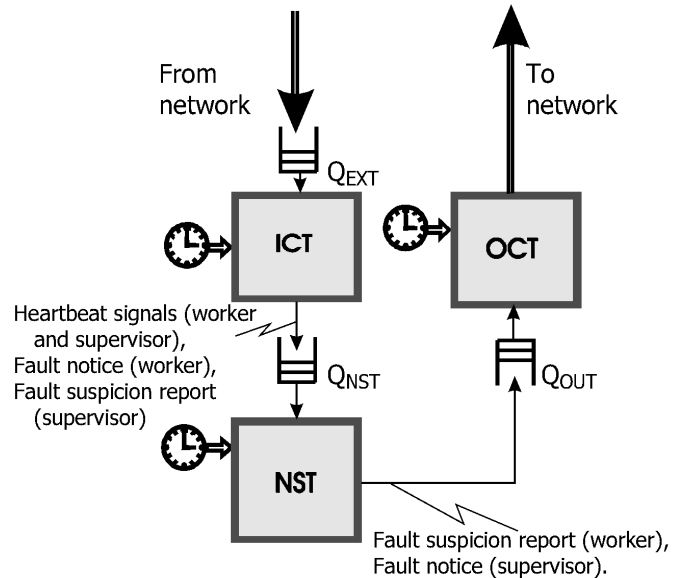


Fig. 8. Modular implementation model of the SNS scheme.

service times of these threads and hence eases the analysis of the fault detection latency bound of the SNS scheme.

The functions of the three kernel-threads are as follows:

- 1) **NST**: This is responsible for almost everything in the protocols described in Section 4 except the generation of the heartbeat signals. For example, some or all of the functions such as periodic analysis of the received heartbeat signals, reporting fault suspicions, election of a supervisor, and supervisory function, are handled by this kernel-thread, depending on the role of the host node.
- 2) **ICT**: This distributes the messages received from neighbors to appropriate destination queues. In particular, it periodically forwards the received heartbeat signals and also the fault notices received from the supervisor node into the queue Q_{NST} checked by NST.
- 3) **OCT**: This generates and sends heartbeat signals (messages) to the node's healthy neighbors. This also sends fault suspicion reports (worker) or fault notices (supervisor) deposited into Q_{OUT} by NST, out to the point-to-point network.

The modular structuring of the SNS scheme allows easy incorporation of complementary mechanisms for network surveillance. Also, such a structuring allows relatively easy analysis of the fault detection latency bounds.

We now formally analyze the performance of the SNS scheme to obtain some tight bounds on the fault detection times. These kinds of analyses provide useful guides to implementers of RT fault-tolerant DCSs in safety-critical applications.

5.2 Definitions

- 1) **Worst-case fault detection latency L_{F-LOC} for local detection of a node fault F** : Maximum amount of time from the occurrence of a fault $F \in \{\text{PI fault, PO fault, processor crash}\}$ in a node to the sufficient generation of fault suspicion reports.

- 2) **Worst-case fault detection latency L_{F-SUP} for identification of a worker node fault F by the supervisor:** Maximum amount of time from the occurrence of a fault $F \in \{\text{PI fault, PO fault, processor crash}\}$ in a worker node to the identification of the faulty node by the supervisor node.
 - 3) **Worst-case fault detection latency L_F for learning of a node fault F by all nodes:** Maximum amount of time from the occurrence of a node fault $F \in \{\text{PI fault, PO fault, processor crash}\}$ to the learning of the fault by all the healthy nodes in the system.
 - 4) **Worst-case fault detection latency L_{LF-LOC} for local detection of a link fault by an attached node:** Maximum amount of time from the occurrence of a fault in a link to the generation of the fault suspicion report by at least one healthy node connected to the link.
 - 5) **Worst-case fault detection latency L_{LF-SUP} for identification of a link fault by the supervisor:** Maximum amount of time from the occurrence of a link fault to the identification of the faulty link by the supervisor node.
 - 6) **Worst-case fault detection latency L_{LF} for learning of a link fault by all nodes:** Maximum amount of time from the occurrence of a link fault to the learning of the fault occurrence by all the healthy nodes in the system.
 - 7) **Worst-case fault detection latency WDL of the SNS scheme:** Maximum amount of time from the occurrence of a fault $F \in \{\text{node fault, link fault}\}$ to the learning of the fault occurrence by all the healthy nodes in the system.
 - 8) **Synchronization range SR :** Maximum deviation among the heartbeat generation times of the OCTs in different nodes in each round, which is also equal to the maximum deviation among the analysis start times of the NSTs in different nodes in each round.
 - 9) **Maximum thread turnaround time:** Maximum interval of time that elapses from the time an item arrives at any of the input queues of the thread to the time the corresponding output is produced.
 - a) **Maximum ICT turnaround time MIT :** Maximum amount of time that elapses from the arrival of a message in the input queue of ICT in a node (i.e., Q_{EXT} in Fig. 8) to the time at which ICT completes the forwarding of the item.
 - b) **Maximum OCT turnaround time MOT :** Maximum amount of time that elapses from the time of arrival of an item at the input queue of OCT (i.e., Q_{OUT} in Fig. 8) to the time at which OCT completes the processing (sending) of the item.
 - c) **Maximum NST turnaround time MNT :** Maximum amount of time that elapses from the time of arrival of an item at the input queue of NST (i.e., Q_{NST} in Fig. 8) to the time at which NST completes the processing of the item. This time includes the time taken by NST waiting for the arrival of the next analysis period and thus reflects an SR as well.
 - 10) **Maximum message transmission time $TRANS$:** Maximum time interval that elapses from the time a message leaves the Q_{OUT} of a node to the time the message reaches the Q_{EXT} of any other node in the system. This time includes the time taken for repeated transmissions of the same message over two different paths as explained in Section 4.2.
 - 11) **Maximum message multicast time $MCAST$:** Maximum time it takes for the OCT in a node to complete its part of a multicast of a message in Q_{OUT} to all healthy nodes in the system. After the OCT completes its part, it takes additional $TRANS$ time units for the last copy of the multicast message to reach the last node in the multicast group.
 - 12) **Heartbeat generation interval π :** Time interval between two successive rounds of heartbeat signal generations. This is also the time interval between two successive rounds of heartbeat analysis.
- Under assumption A5, the clocks in various nodes are synchronized within a reasonably small time value δ . δ is much smaller in comparison with the other time variables defined in this section. So, δ is ignored in the latency bound analysis.

5.3 Timing Chart of the SNS Scheme and an Analysis of L_F

Fig. 9 shows the timing chart for two worker nodes. Various events that occur in two healthy neighbor nodes X and Y are shown. First, the OCT in node X generates the last copy of the heartbeat signal in round i and sends it out to node Y (event $h_{X,Y,4}^i$). After a maximum of $TRANS$ time units, the heartbeat signal reaches the Q_{EXT} of node Y (event $r_{X,Y,4}^i$). After a maximum of MIT time units, the ICT moves one copy of the received heartbeat signal to Q_{NST} , the input queue of NST (event n_Y^i). Then, MNT time units later the NST finishes the analysis of the heartbeat signals received from all healthy neighbor nodes (event c_Y^i).

The following analysis generates the worst-case detection latency L_F for a PO fault F in a certain worker node in the system.

1) Detection of the Fault Locally

The worst-case latency for the detection of the PO fault F in node X by more than one neighbor would occur if F occurs immediately after event $h_{X,Z,1}^i$, where Z is the second last neighbor node to which X sends heartbeat signals in each round. Therefore, during the i th round, only one neighbor will generate a fault suspicion report and other neighbors will do so in the next round. Let node Y be the neighbor of X that is different from Z and generates the first fault suspicion report for F in the $(i + 1)$ th round. Other healthy neighbors of X will also generate the same report at about the same time (i.e., within the interval of SR). The latest copy of the heartbeat signal that node Y expects from X but does not receive is the copy

MCAST time-units. In the worst-case, the multicast message reaches the last destination node in the multicast group within *TRANS* time units after the OCT completes its multicast action. The ICT in each destination node then moves this message to Q_{NST} within further *MIT* time units. The NST finishes the processing of the message within *MNT* time units. Thus, we have

$$\begin{aligned} \therefore L_{PO} &= L_{PO-SUP} + MCAST \\ &\quad + TRANS + MIT + MNT \\ &= \pi + 2 \times (TRANS + MIT + MNT) + MOT \\ &\quad + MCAST + TRANS + MIT + MNT \\ &= \pi + 3 \times (TRANS + MIT + MNT) \\ &\quad + MOT + MCAST \end{aligned} \quad (4)$$

PROPOSITION P1. *If a PO fault or a processor crash occurs in a certain worker node in the system operating under the SNS scheme, then the fault detection latency is within the bound of $\pi + 3 \times (TRANS + MIT + MNT) + MOT + MCAST$, where the parameters are as defined in Section 5.2.*

NUMERICAL EXAMPLE. In the current prototype implementation of the DREAM kernel, we have chosen the periods of the kernel-threads such that $MIT = MOT = 8$ msec, $\pi = 24$ msec, and $MNT = 12$ msec. If we choose $TRANS = 20$ msec, and $MCAST = 12$ msec, then $L_{PO} = 164$ msec.

5.4 Summary of Fault Detection Latency Analysis

In essentially the same manner as done in the previous section, the fault detection latency bounds for other faults can be analyzed as well. In this section we summarize the results of such analysis.

PROPOSITION P2. *If a PI fault occurs in a certain worker node or a link fault occurs in a certain link in the system operating under the SNS scheme, then the fault detection latency is within the bound of $\pi + 2 \times TRANS + 3 \times (MIT + MNT) + MOT + MCAST$.*

PROPOSITION P3. *If the supervisor node in the system operating under the SNS scheme becomes faulty, then the fault detection and supervisor election latency is within the bound of $\pi + 3 \times (MIT + MNT + TRANS) + MCAST + MCAST$, where $MCAST'$ is a special case of $MCAST$ in which recipient nodes are the neighbors of the supervisor.*

6 CONCLUSION

The architecture of distributed RT information systems proposed in this paper has several important advantages. The architecture suggests efficient ways of constructing RT data servers providing dependable timely services. Uniform structuring of all types of RT data server objects and their replicas discussed in this paper is enabled by the TMO scheme and it leads to improved understandability and maintainability of complex information systems. The modular DCMS is another important part of the proposed architecture. The DCMS handles network surveillance, network reconfiguration, and software reconfiguration in an easily understandable expandable manner. A challenge in realizing

such a DCMS is to develop a RT NS scheme that is widely applicable and yet yields a strong worst-case fault detection latency. The SNS scheme is one such scheme found.

The SNS scheme is a broadly applicable RT NS scheme for general-topology point-to-point networks. An implementation model of the scheme that has a modular and easily expandable structure has been formulated. A prototype implementation of the scheme has been incorporated into a DREAM kernel prototype running on a PC network. Through experimental injection of faults, the capability of the implementation was validated to some extent. In our view, the main strengths of the SNS scheme and the implementation model formulated are in that they enable relatively easy determination of tight bounds on the fault detection latency through an analysis, which is of great importance in many hard-real-time applications. A meaningful direction for future research would be to integrate the SNS scheme with fault-tolerant RT object or process execution schemes such as the Primary-Shadow TMO Replication scheme and others [10], [13], [22], thereby further enhancing the fault coverage and survivability of application systems that can be attained by available design techniques. Adaptation of the SNS scheme to highly parallel multicomputers [16] is also a meaningful topic.

ACKNOWLEDGMENTS

The research work reported here was supported, in part, by the United States Defense Advanced Research Project Agency under Contract No. N66001-97-C-8516, monitored by SPAWAR; in part, by the University of California's MICRO Program under Grants No. 96-169 and No. 97-082; in part by LG Electronics; and, in part, by the U.S. Air Force Rome Laboratory and SoHaR Inc.

REFERENCES

- [1] A. Attoui, "An Object Oriented Model for Parallel and Reactive Systems," *Proc. IEEE 12th Computer Soc. Real-Time Systems Symp.*, pp. 84-93, 1991.
- [2] F. Bastani et al., "Toward Dependable Safety-Critical Software," *Proc. Second IEEE Computer Soc. Workshop Object-Oriented Real-Time Dependable Systems*, pp. 86-92, Laguna Beach, Calif., Feb. 1996.
- [3] F. Cristian, "Agreeing on Who is Present and Who is Absent in a Synchronous Distributed System," *Proc. 18th IEEE Computer Soc. Int'l Symp. Fault-Tolerant Computing*, pp. 206-211, June 1988.
- [4] M.A. Ellis and B. Stroustrup, *The Annotated C++ Reference Manual*, Addison-Wesley, Reading, Mass., 1995.
- [5] P. Ezhilchelvan et al., "A Robust Processor-Group Membership in Synchronous Distributed Systems," *Proc. IEEE Computer Soc. Real-Time Systems Symp.*, pp. 173-179, Dec. 1990.
- [6] M. Hecht et al., "A Distributed Fault Tolerant Architecture for Nuclear Reactor and Other Critical Process Control Applications," *Proc. 21st IEEE Computer Soc. Int'l Symp. Fault-Tolerant Computing*, pp. 462-469, Montreal, June 1991.
- [7] Y. Ishikawa, H. Tokuda, and C.W. Mercer, "An Object-Oriented Real-Time Programming Language," *Computer*, pp. 66-73, Oct. 1992.
- [8] W. Jia, J. Kaiser, and E. Nett, "Reliable Multicast Protocol for Fault Tolerant Group Communication," *IEEE Micro*, vol. 16, no. 2, pp. 59-67, Apr. 1996.
- [9] J.L. Kim and G.G. Belford, "A Robust Distributed Election Protocol," *Proc. Seventh IEEE Computer Soc. Symp. Reliable Distributed Systems*, pp. 54-60, Columbus, Ohio, Oct. 1988.
- [10] K.H. Kim, "Action-Level Fault Tolerance," *Advances in Real-Time Systems*, ch. 17, S.H. Son, ed., pp. 415-434, Prentice Hall, 1994.

- [11] K.H. Kim et al., "Distinguishing Features and Potential Roles of the RTO.k Object Model," *Proc. First IEEE Computer Soc. Workshop Object-Oriented Real-Time Dependable Systems*, pp. 36-45, Dana Point, Calif., Oct. 1994.
- [12] K.H. Kim et al., "A Timeliness-Guaranteed Kernel Model—DREAM Kernel and Implementation Techniques," *Proc. Int'l Workshop Real-Time Computing Systems and Applications*, pp. 80-87, Tokyo, Oct. 1995.
- [13] K.H. Kim and C. Subbaraman, "PSRR: A Scheme for Time-Bounded Fault Tolerance in Distributed Object-Based Systems," *Proc. IEEE High-Assurance Systems Eng. Workshop*, pp. 120-128, Niagara on the Lake, Ont., Canada, Oct. 1996.
- [14] K.H. Kim, "Object Structures for Real-Time Systems and Simulators," *Computer*, vol. 30, no. 8, pp. 62-70, Aug. 1997.
- [15] H. Kopetz and G. Grunsteidl, "TTP-A: Time Triggered Protocol for Fault Tolerant Real-Time Systems," *Proc. 23rd IEEE Computer Soc. Int'l Symp. Fault-Tolerant Computing*, pp. 524-533, Toulouse, France, June 1993.
- [16] A.C. Liang, S. Bhattacharya, and W.T. Tsai, "Fault-Tolerant Multicasting on Hypercubes," *J. Parallel and Distributed Systems*, pp. 418-428, vol. 23, no. 3, Dec. 1994.
- [17] H. Lonn and R. Snedsbol, "Efficient Synchronization, Atomic Broadcast, and Membership Agreement in TDMA Protocol," *Proc. Int'l Conf. Parallel and Distributed Systems*, pp. 405-412, France, Sept. 1996.
- [18] L.E. Moser et al., "The Totem System," *Proc. 25th IEEE Computer Soc. Int'l Symp. Fault-Tolerant Computing*, pp. 61-66, Pasadena, Calif., June 1995.
- [19] L. Rodriguez, P. Verissimo, and J. Rufino, "A Low-Level Processor Group Membership Protocol for LANs," *Proc. 13th IEEE Computer Soc. Int'l Conf. Distributed Computing Systems*, pp. 541-550, May 1993.
- [20] S.H. Son, *Advances in Real-Time Systems*, Prentice Hall, N.J., 1994.
- [21] K. Takashio and M. Tokoro, "DROL: An Object-Oriented Programming Language for Distributed Real-Time Systems," *Proc. OOPSLA*, pp. 276-294, ACM, 1992.
- [22] I.-L. Yen, "An Object-Oriented Fault Tolerance Framework Based on Specialization Techniques," *Proc. Third IEEE Computer Soc. Workshop Object-Oriented Real-Time Dependable Systems*, pp. 291-297, Newport Beach, Calif., Feb. 1997.



K.H. (Kane) Kim received his MA degree in the Computer Science Department at the University of Texas at Austin in 1972; and his PhD degree in the Computer Science Division at the University of California, Berkeley, in 1974. He is now a professor of computer engineering and computer science at the University of California at Irvine. Prior to joining the UC Irvine faculty in August 1986, he was a professor at the University of South Florida; an associate professor at the State University of New York at Binghamton; and

an assistant professor at the University of Southern California. He has been expanding a research laboratory called the Distributed Real-Time Ever-Available Microcomputing (DREAM) Lab during the past 15 years, and he has experimentally validated several new basic approaches for fault tolerance broadly applicable in distributed/parallel real-time systems and new object-oriented real-time software/system engineering approaches. He is the originator of the Distributed Recovery Block (DRB) technique and several other basic practical and dependable computing approaches, and the principal originator of the TMO (time-triggered message-triggered object, also called RTO.k) object structuring scheme. He received the 1998 IEEE Computer Society Technical Achievement Award for his contributions to the scientific foundation for both real-time fault-tolerant computing and real-time object-oriented distributed computing. He served as chair of the Computer Society's Technical Committee on Distributed Processing from 1984 to 1986, and he has hosted several IEEE conferences. He served as a member of the founding Editorial Board for *IEEE Transactions on Parallel and Distributed Systems* from 1989 to 1994. He was one of the two co-founders of the Korean Computer Scientists and Engineers Association in America in 1982, and he was the KOCSEA's president in 1991. He is a fellow of the IEEE, and a member of the IEEE Computer Society and IFIP WG 10.4 on Dependable Computing.



Chittur Subbaraman received his bachelor's degree in instrumentation engineering from the University of Calicut, India, in 1991; his master's degree in electrical engineering from Louisiana State University, Baton Rouge, in 1993; and his PhD degree in computer engineering from the University of California at Irvine in 1997. He is currently a software engineer in the Windows NT clusters group of the Microsoft Corporation. His research interests include distributed clustered systems, real-time operating systems, and real-time object-based distributed/parallel computer systems.

ing systems, and real-time object-based distributed/parallel computer systems.