

Group Communication in Real-Time Computing Systems: Issues and Directions

K. H. (Kane) Kim

University of California
Irvine, CA 92697, U.S.A.
Kane@Ece.Uci.Edu

Abstract: Group communication in real-time computing systems has been a subject of research for almost two decades but it is not yet a mature technological field. The purpose of this paper is to review the main goals of developing group communication protocols and establish a practical framework of protocols toward meeting the goals. As a part of this, an attempt is made to clarify the relationship between group communication protocols and fault tolerance. Remaining major research issues and promising directions for solution search are also discussed.

Index Terms: group communication, multicast, fault tolerance, bounded delay, delivery, acknowledgment, retry, multicast delay bound.

1. Introduction

Group communication in real-time computing systems has been a subject of research for almost two decades but it is not yet a mature technological field [B1], [B2], [C1], [C2], [C3], [G1], [K1], [K8], [M1]. The main challenge in establishing group communication protocols is to deal with possible fault occurrences. There have been some proposals for using group communication protocols, in particular, reliable multicast mechanisms, as basic building-blocks for fault-tolerant distributed systems. The validity of this thesis cannot be established until practical reliable multicast mechanisms get established. Until then, we feel that it is sensible to cast group communication protocols as applications of established real-time fault tolerance techniques effective in handling failures of low-level components such as processors, paths in communication/interconnection networks, processor-network interfaces, and operating systems components.

This paper starts in Section 2 with a characterization of the application goals of group communication protocols. Then in Section 3, the relationship between group communication protocols and fault tolerance is clarified and a practical framework for fault-tolerant real-time multicast facilities is established. Effective programming interfaces for real-time multicast facilities are discussed in Section 4. Remaining major research issues are summarized in Section 5.

2. Basic goals of multicasts

There are largely two types of conceivable needs for

group communications in real-time distributed computing systems (DCS's).

(M1) *Server replicas* and tightly coupled heterogeneous servers:

Components of a DCS often maintain the client-server relationship among themselves. For the sake of attaining high system reliability and performance, servers are often replicated. These server replicas must then maintain strict consistency among their states. Each message from a client must be received consistently by these server replicas.

Also, clients and diverse servers are often tightly coupled in the sense that they interact closely and every party should read in the same order the messages from multiple sources even if not every party reads the identical set of messages.

(M2) *News multicast* to casual readers:

This communication pattern occurs when the sender's behavior after multicasting the news does not depend on the receivers' behavior and it is not necessary for the receivers to read the news items in the same order.

Therefore, the basic purpose of developing group communication protocols to be used in real-time DCS's is to attain high performance in operating tightly coupled servers and/or achieving casual news multicasts. However, two additional purposes have been discussed in literature.

(G1) Abstract distributed programming:

Recently as a programming abstraction for distributed software components, to be more specific, for their interactions, the *logical multicast channel* (LMC) has been proposed. LMC is meant to be a complement to the *remote method call* facility. LMC is an abstract facility for message sharing among groups of distributed software components. For example, programmable *data-field channel* (DFC) [K6], [K6] is a case of LMC which facilitates *time-stamp ordered message multicasts* as well as *distributed shared memory variables*. LMC's may often lead to more abstract and compact distributed programming than the remote method call does. Moreover, if a high-performance low-level multicast facility is available, use of LMC's will also lead to higher performance of distributed application software.

(G2) Basic building-blocks for fault-tolerant distributed systems:

There have been some proposals for using group communication protocols, in particular, reliable multicast mechanisms, as basic building-blocks for fault-tolerant distributed systems. Although there is nothing seriously wrong in this conception, the effectiveness of this must be evaluated after establishing reliable multicast mechanisms. Ironically, these proposals did not include any example of a practical mechanism for reliable real-time multicast. Therefore, it is more urgent to establish practically effective mechanisms for reliable real-time multicast.

☆ : Fault Source Component

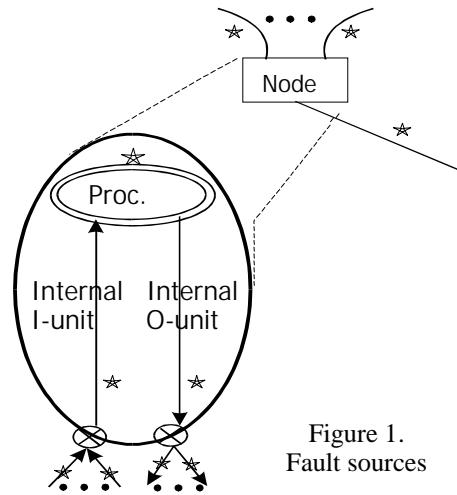


Figure 1. Fault sources

point message communications in spite of component failures. This is an old issue. Therefore, it seems worth casting group communication protocols as applications of established real-time fault tolerance techniques effective in handling failures of low-level components.

3.2. Real-time fault-tolerant point-to-point communication of a message

A formal definition of the problem of *real-time fault-tolerant point-to-point communication of a message* is now in order. 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.

3. Group communications vs. Fault tolerance

3.1. The main challenge in group communications

Multicasts in absence of the possibility of component failures are simple programming problems. In systems based on point-to-point networks, a multicast is merely a finite sequence of point-to-point single message communications. In systems based on physical broadcast facilities, a multicast may become a single broadcast with a group ID in the message header field. In the rest of this paper, we will focus on systems equipped with point-to-point networks since multicast problems in such systems are more complicated than in other systems.

The main challenge in establishing group communication protocols is to deal with possible fault occurrences. Therefore, it is important to first establish and understand the effective techniques for detecting and recovering faults in real-time DCS's. To think that one can try to find effective group communication protocols first and use them to handle component failures is analogous to the thinking that one can find a good house and then use the house to establish a good foundation and a good roof.

Assume that techniques have been found for handling failures of low-level components which occur during single point-to-point message communications. Here major cases of low-level components are processors, paths in communication/interconnection networks, processor-network interfaces, and operating system components. Then possible component failures during a multicast do not introduce any new problems. Therefore, the essence of the challenge is to perform single point-to-

In the model depicted in Figure 1, 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. *Faults in the I-unit* represent the 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.

3.2.1. Detection of transient faults

Let us first consider the impacts of temporary failures of fault source components.

(1t) Detection by the *network infrastructure* which refers to the facility involved in carrying a message from the sender processor to the receiver processor, i.e., the O-unit of the sender node, the point-to-point interconnection network, and the I-unit of the receiver node:

This is a case where the network infrastructure has a *self-checking* capability, i.e., the ability of the infrastructure to detect its own fault that manifests itself in the form of message loss and message corruption. The sender processor may learn this fault of the network infrastructure, in which case it may or may not make retry at sending the message and this is an application-dependent decision. Therefore, a real-time message communication is completed either with the successful delivery to the receiver or with delivery failure known to the sender.

(2t) Detection by the sender processor:

The sender processor typically detects the failure of the message communication by noticing the absence of the acknowledgment message from the receiver. However, the problem here can become quite complicated because the possibility exists that the receiver processor received the main message and generated the acknowledgment message (ack-message) properly but network infrastructure failed to carry the ack-message. Therefore, upon noticing the absence of the ack-message, the sender cannot be certain that the receiver did not receive the main message. Moreover, the receiver which received the main message may proceed to take some actions while the sender is suspecting that the receiver might not have received the message. The sender may resend the message to remove this uncertainty and the receiver can discard the redundant message after noticing that it is retransmission of the message it received earlier. However, the receiver must send an ack-message again and it may again get lost inside the network infrastructure although the probability of this second loss may be quite low.

If the low-probability event of the second loss of the ack-message occurs, then the sender may resend the message again to remove the uncertainty. Each time the sender detects the absence of the ack-message, the sender faces the question of whether the main message is now obsolete with respect to the application semantics, i.e., whether it is useless to send that message now. If it has become obsolete, then the sender will take a different course of actions. So, a very difficult question arises here: what happens if the sender takes a different course of actions since the message it has tried to send has become obsolete while the receiver actually received the message and is now following a course of actions based on the received message? There cannot be any clean and general answer to this question. Another difficult question is: what happens if the message that the sender has tried to send has become obsolete and the receiver indeed has not received the message because of the fault

in the network infrastructure or within the receiver itself? This is a dangerous situation to avoid.

Therefore, the arrival at the sender processor of an ack-message (corresponding to the initial transmission or some retransmission of the main message) within an acceptable time bound should be a part of the conditions for successful completion of a real-time message communication. Until such time at which an ack-message arrives at the sender, the receiver must not perform any irrevocable actions based on the received message. So, successful message communication can now be defined as follows.

Definition:

(D1) Communication of a message is *successful* if and only if the message reaches the receiver processor with its content in tact within the *guaranteed delivery time bound* (DTB) and a corresponding ack-message returns to the sender processor within the *acknowledgment time bound* (ATB). DTB is the bound determined at the design time on the length of the time interval from the instant at which the message leaves from the sender processor to the instant at which the message reaches the receiver processor. ATB is the bound determined at the design time on the length of the time interval from the instant at which the main message leaves from the sender processor to the instant at which an ack-message from the receiver corresponding to the initial transmission or a retransmission of the main message arrives at the sender processor. ■

Now if an ack-message does not return within ATB, the sender can act as if the message communication failed regardless of whether the receiver processor received the message or not. The attempted real-time message communication has been completed with delivery failure known to *the sender which will attempt to inform the receiver* (so that the latter may revoke some actions taken). Needless to say, communication of critical messages must be designed into applications with the good understanding of DTB and ATB.

(3t) Detection by the receiver processor:

The receiver processor can detect the failure of the message communication only when it knows in advance the *transmission schedule* of the sender processor. For example, if the sender processor previously told the receiver processor that the former would send an important message at 10am, then the latter can detect the failure if the message does not arrive by DTB seconds after 10am. For another example, if the system is equipped with a TDMA bus network and every processor is designed to send some message even with null content during its slot in each TDMA cycle, then every other processor can detect the failure of message communication initiated by the former processor [K1], [K8], [K8], [K9].

In this case, the receiver processor should attempt to inform the sender of the non-arrival of the message. The

sender can learn of the failure not only by this notice from the receiver but also by a time-out on the return of the ack-message.

Again, whether the sender makes a retry at sending the message depends on the application. In any case, the attempted real-time message communication can be completed with delivery failure known to both the sender and the receiver.

3.2.2. Detection of permanent faults

So far, we have considered the impacts of transient faults of fault source components occurring during the message communication. Let us now consider the impacts of permanent faults.

The capabilities for detection of the permanent faults of various fault source components are distributed within the system including the network infrastructure. Some of those capabilities may be born by the sender processor and the receiver processor. Such distributed capabilities for detection of permanent faults are collectively called the *system-wide permanent fault detection (SwPFaD) facility*. A good example of an approach to implementing the SwPFaD facility is discussed in [K4].

It should be noted that the detection by the SwPFaD facility of permanent faults is separate from the sender processor's time-out on the ack-message. Even the ordering of the two events varies depending upon the circumstances.

(1p) Permanent faults of some components in the *network infrastructure*:

These faults are detected by the SwPFaD facility in general. If the sender processor experiences a time-out on the ack-message before the SwPFaD facility concludes on detection of these permanent faults, the sender processor acts as in the case of handling transient faults. Once these permanent faults are concluded, the only important question here is whether the network infrastructure has enough capacity (e.g., alternate route) left for carrying a message from the sender processor to the receiver processor. If it has, the sender processor may exercise a resend option. If it does not, then the sender processor concludes the message communication attempt with the failure result. Furthermore, the sender processor may follow a drastically different application scenario from that point on or give up the current application. Basically, no new challenging issues are introduced.

(2p) Permanent faults of the I-unit, the O-unit, and the processor of the receiver node

These faults are detected by the SwPFaD facility in general. If the sender processor experiences a time-out on the ack-message before the SwPFaD facility concludes on detection of these permanent faults, the sender processor acts as in the case of handling transient faults. Once the permanent fault in any of the three components of the receiver node is concluded, then practically the receiver node is dead from the sender processor's point of view.

Therefore, the sender processor must conclude the message communication attempt with the failure result.

(3p) Permanent faults of the I-unit, the O-unit, and the processor of the sender node

These faults are detected by the SwPFaD facility in general. If the sender processor experiences a time-out on the ack-message before the SwPFaD facility concludes on detection of permanent faults of the I-unit or the O-unit of the sender node, the sender processor acts as in the case of handling transient faults. Once the permanent fault of any of three components of the receiver node is concluded, then practically the receiver node is unusable for any meaningful application. Therefore, not only the message communication attempt from the sender processor is automatically concluded with the failure result but also a *system-wide fault tolerance action sequence* to compensate for the loss of the sender node must begin. A part of the fault tolerance action sequence will be for the receiver node to remove the remaining effects of the failed message communication attempt of the lost sender processor.

Definition:

(D2) An attempt for communication of a message is said to be *completed* at one of the following instants:

- (1) When the sender processor receives an ack-message within ATB;
- (2) When the sender processor learns, before receiving an ack-message, of the permanent fault of any of the three components of the receiver node;
- (3) When the sender processor learns, before receiving an ack-message, of the permanent fault of the I-unit or the O-unit of the host (sender) node;
- (4) When the sender processor learns that the network infrastructure no longer has the capacity for carrying a message from itself to the receiver processor and then the sender experiences a time-out on the ack-message;
- (5) When the SwPFaD facility concludes, before the sender processor receives an ack-message, on the permanent fault of any of the three components of the sender node. ■

3.3. Real-time fault-tolerant multicast

Let us now extend the definition of the real-time fault-tolerant point-to-point message communication into one for *real-time fault-tolerant multicast*. A multicast involves a group of n receiver processors. As mentioned before, the fault-free case is easy.

Definition:

(D3) Multicast of a message is successful if and only if the message reaches each receiver processor with its content in tact within the guaranteed delivery time bound (DTB) after the message leaves from the sender and a corresponding ack-message returns to the sender processor within the acknowledgment time bound (ATB) after the message leaves from the sender. ■

If the message is delivered to some members of the

receiver processor group but not to other members, then the multicast should be cancelled in the case where receivers are tightly coupled servers. If the receivers are casual news readers, then such a cancellation may not be required but we will not consider this easy situation in the rest of this paper. Therefore, it may be safer to make every receiver to process the received message only after it is certain that all other receivers received the same message. In RT DCS's (in which the global time base is available), this can be accomplished the most conveniently by asking every receiver to process the message at a certain time called the *official release time*, e.g., 10am, by which the multicast can definitely be completed. Then it is possible that an ack-message from a certain receiver arrives at the sender but the sender later notifies the receiver about the cancellation of the multicast. Therefore, one problem with this conservative approach is the large delay in completing a multicast since the delay should include the time needed for the sender or some other authority to notify cancellation of the multicast to the receivers which received the message already.

Even if some receivers receive multiple multicast messages at around the same time, there is no inconsistency among the receivers due to the use of official release times. This is an important advantage of the approach of attaching the official release time specification to each multicast message over other approaches of ordering multiple multicast messages.

If communication of the message to any of the n receivers fails, then the multicast fails. Therefore, for communication of the message to each receiver, up to m retries must be utilized.

Definition:

(D4) An attempt for multicast of a message is said to be *completed* at one of the following instants:

- (1) When the communication of the message to the last receiver is successfully completed, i.e., the sender processor receives an ack-message from the last receiver within ATB;
- (2) When the communication of the message to any of the receivers is completed with the failure result as defined in D2. ■

Once a multicast attempt becomes a failure, then the application can perform one of the following three sequences of actions:

- (a) The entire application is aborted.
- (b) A segment of the application that encompasses the failed multicast action is aborted and a recovery action sequence such as a state rollback followed by an execution of a less ambitious application-segment, is invoked.
- (c) A smaller-scale multicast is attempted and if successful, a corresponding less ambitious application scenario is followed.

An interesting and important special case of (c) is

where the number of receivers in the new smaller-scale multicast is the same as or less than the number of successful receivers in the larger multicast just failed. In this case, the notice from the sender node about the cancellation of the failed multicast is sent along with the notice about the decision to move forward with a less ambitious predefined application scenario. Therefore, each receiver getting this expanded notice will not abort the message received earlier and instead move forward to follow a newly selected application scenario. For example, the original multicast may have involved 10 receivers which are server replicas. When communication of the message to five server replicas became successful and communication to five others became failures, the sender or the application coordinator may decide to proceed with five server replicas and give up on the other five. The five surviving server replicas and the system management facilities must be aware of this decision so that from this on they may concern with the consistency among themselves only.

Therefore, an invocation of a fault-tolerant real-time multicast action can be specified as:

FR multicast (receiver-group, official-release-time 10am, on-failure action-sequence-b()).

From the definitions, D1, D2, D3, and D4, one can calculate a tight *multicast delay bound*, i.e., bound on the amount of time taken for completion of a multicast (with the success or failure result). The designer of the sender processor must of course choose the official release time for a multicast message with good understanding of the multicast delay bound.

4. Programming interfaces for real-time multicast facilities

In order to support easy use of real-time multicast facilities by the distributed computing application designers, easy-to-use programming interfaces need to be developed. This is an area where a very limited amount of research has been conducted. The interface adopted in the TMO (time-triggered message-triggered object) structuring scheme [K4], [K5], [K6], [K6] is briefly presented here as a case study.

Figure 2 depicts the programmable *data field channels* (DFC's) to which distributed TMO's are connected.

Real-time DFC is a promising feature of the TMO scheme that facilitates highly abstract (relieving the programmer of the burden of dealing with underlying OS services and network protocols) and yet highly efficient cooperation among remote TMO's. The essence of the real-time DFC scheme is to facilitate dynamic creation of *logical multicast channels* and dynamic connection of objects (or processes in process-structured systems) to the logical channels in such a way that the idiosyncrasies of the physical communication networks are transparent to

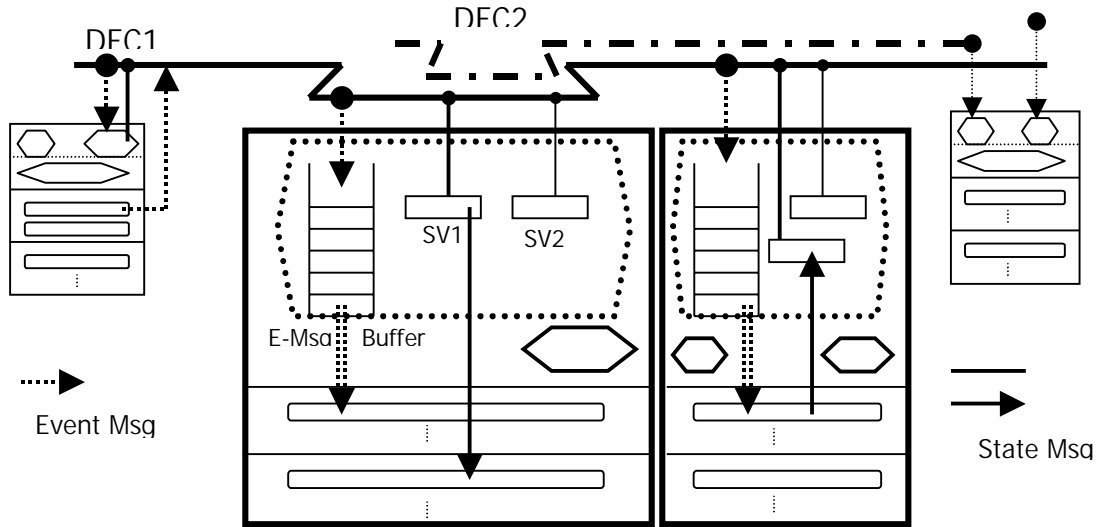


Figure 2. Programmable DFC's connecting TMO's

the object designer. In other words, when objects are designed to communicate via the logical multicast channels only, objects can be dynamically relocated without impacting other cooperating objects.

A logical multicast channel can be implemented over point-to-point networks as well as over broadcast-enabled bus networks.

The programmable DFC scheme supports not only conventional *event messages* but also *state messages* which are based on the distributed replicated memory semantics.

A state message carries information to be stored in a fixed memory location in each receiver corresponding to the ID of the state message [K2], [K6], [K9]. Therefore, the ID of a state message represents a group of replicated memory units, each capable of holding the information carried in the state message and belonging to a different receiver. The producer of a state message timestamps the message at the message production time. Each receiver will read the content of its state message memory at its convenient time. This means that the producer may update the contents of the state message memory units at a higher frequency than that at which a certain receiver reads the content of its state message memory. A state message is thus typically used to share the periodically observed state information about a dynamic state-varying item, e.g., temperature of an oven.

Therefore, as depicted in Figure 2, a DFC is associated with an event message buffer in the data area of each TMO connected to the DFC. Two basic operations can be performed on an event message.

(1) announce(DFC1, msg7, official-release-time 11am, on-failure action-sequence-c()): Any TMO connected to DFC1 can perform this operation and the result is the

delivery of the message msg7 to all other TMO's connected to DFC1.

(2) receive(DFC1): A TMO can perform this operation to pick an "officially released" event message from its buffer. Event messages are picked one at a time in the order of their official release times.

For state messages, the following two basic operations are relevant.

(1) g-update(DFC1, SV1, value3, official-release-time 11:30am, on-failure action-sequence-d()): Any TMO connected to DFC1 can perform this *global update* operation and the result is the updating of all replicas of variable SV1 located in other TMO's connected to DFC1 with the value value3.

(2) read(DFC1, SV1): A TMO can perform this operation to read the most recent officially released value contained in its object variable SV1.

Therefore, DFC's serve as an alternative mechanism for interaction among distributed objects and complements the remote method call mechanism. In some sense, TMO's are more loosely connected via DFC's and maintain higher degrees of autonomy than when connected via the remote method call mechanism.

5. Summary of research issues

The research issues exposed in preceding sections can be summarized as follows.

(1) The SwPFaD (system-wide permanent fault detection) facility plays an important role in determining the efficiency of fault-tolerant real-time multicast. Although a small number of useful techniques have been developed, this area requires much further study.

(2) The performance aspect of the fault-tolerant real-time multicast protocol presented in Section 3.2 and 3.3, needs to be studied much further. There is also a question as to whether this protocol is optimal in any reasonable sense.

(3) Efficient multicasts over the network infrastructure consisting of a mixture of broadcast subnets and point-to-point subnets are another subject which has not been studied sufficiently. A good efficiency measure is the multicast delay bound.

(4) Programming interfaces for the real-time multicast facilities have not been developed sufficiently. In Section 4, the programming interface in the TMO scheme was briefly discussed but it is still undergoing experimental validation. The programming interface, especially, the programming of an action sequence to be followed on completion of a multicast with the failure result, is an important topic for future research.

Acknowledgment: The research work reported here was supported in part by the US Defense Advanced Research Project Agency under Contract N66001-97-C-8516 monitored by SPAWAR and by a grant from the NSF NGS (Next-Generation Software) '99 Program.

References

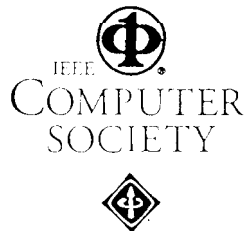
- [B1] Barcellos, Marinho P. and Ezhilchelvan, Paul D., "An End-to-End Reliable Multicast Protocol Using Polling for Scalability", Tech. Rept. 609, Dept. of Computing Science, Univ. of Newcastle upon Tyne, 1997.
- [B2] Birman, K., Schiper, A., and Stephenson, P., "Lightweight Causal and Atomic Group Multicast", ACM Trans. on Computer Systems, Vol. 9, No. 3, August 1991, pp. 272-314.
- [C1] Chang, J. M. and Maxemchuk, N., "Reliable Broadcast Protocols", ACM Transactions on Computer Systems, Vol. 2, No. 3, pp. 251-273, Aug. 1984.
- [C2] Cristian, F., Aghili, H., Strong, R., Dolev, D., "Atomic Broadcast: From simple message diffusion to Byzantine Agreement", Proc. FTCS 15, Ann Arbor, Michigan, pp.200-206, 1985.
- [C3] Cheriton, D. R., and Skeen, D., "Understanding the Limitations of Causally and Totally Ordered Communication", 14th ACM Symposium on Operating Systems Principles, pp. 44-57.
- [G1] Grossglauser, M. "Optimal Deterministic Timeouts for Reliable Scalable Multicast", IEEE INFOCOM '96, San Francisco, CA, March 1996.
- [K1] Kim, K. H., and Shokri, E. H. "Minimal-Delay Decentralized Maintenance of Processor-Group Membership in TDMA-Bus LAN Systems", Proc. IEEE Computer Society's 13th International Conf. on Distributed Computing Systems, May, 1993, Pittsburgh, pp. 410-419.
- [K2] Kim, K.H., Mori, K., and Nakanishi, H., "Realization of Autonomous Decentralized Computing with the RTO.k Object Structuring Scheme and the HU-DF Inter-Process-Group Communication Scheme", Proc. 1995 IEEE CS's Int'l Symp. on Autonomous Decentralized Systems (ISADS 95), Phoenix, AZ, April. 1995, pp.305-312.
- [K3] Kim, K. H., "Object Structures for Real-Time Systems and Simulators", IEEE Computer, Vol. 30, No. 8, August 1997, pp. 62-70.
- [K4] Kim, K.H., and Subbaraman, C., "Dynamic Configuration Management in Reliable Distributed Real-Time Information Systems", IEEE Trans. on Knowledge and Data Eng., Vol.11, No.1, Jan./Feb. 1999, pp. 239-254.
- [K5] Kim, K.H., "Real-Time Object-Oriented Distributed Software Engineering and the TMO Scheme", Int'l Jour. of Software Engineering & Knowledge Engineering, Vol.9, No.2, April 1999, pp.251-276.
- [K6] Kim, K.H., Ishida, M., and Liu, J., "An Efficient Middleware Architecture Supporting Time-Triggered Message-Triggered Objects and an NT-based Implementation", Proc. ISORC '99 (IEEE CS Int'l Symp. on Object-oriented Real-time distributed Computing), May '99, pp.54-63.
- [K7] Kopetz, H. , Gruensteidl, G., and Reisinger, J., "Fault-Tolerant Membership Service in a Synchronous Distributed Real-Time System", Proc. IFIP WG 10.4's 1st Int'l. Working Conf. on Dependable Computing for Critical Applications, Santa Barbara, August 1989, pp. 167-173.
- [K8] Kopetz, H., and Gruensteidl, G., "TTP - A Time-Triggered Protocol for Fault-Tolerant Real-Time Systems", Proc. 23rd IEEE Int'l Symp. on Fault-Tolerant Computing (FTCS-23). Toulouse, France, pp. 524-532. A revised version appeared in IEEE Computer, vol. 24 (1), pp. 22-66.
- [K9] Kopetz, H., 'Real-Time Systems', Kluwer Academic Pub., 1997.
- [M1] Miller, C. Kenneth, "Multicast Networking", 51-10-01 Auerbach Publications, 1996.
- [M2] Mori, K., "Autonomous Decentralized Systems: Concept, Data Field Architecture, and Future Trends", Proc. IEEE CS Int'l Symp. on Autonomous Decentralized Systems (ISADS 93), Mar. 1993, Kawasaki, Japan, pp. 28-34.

Proceedings

7th IEEE Workshop on
**Future Trends of
Distributed Computing Systems**

December 20–22, 1999
Cape Town, South Africa

Sponsored by
IEEE Computer Society



Los Alamitos, California

Washington • Brussels • Tokyo