North-Holland
Microprocessing and Microprogramming 22 (1988) 263–271

An Algorithm for Solving Distributed Termination Problem

R.K. Arora and M.N. Gupta

Computer Services Centre Indian Institute of Technology Delhi Hauz Khas: New Delhi-110016 India

A spanning tree based termination detection algorithm for solving distributed termination problem is given along with its correctness proof. In this algorithm, the root process, apart from assuming the additional responsibility of termination detection, is treated like any other process of the distributed program. The root process initiates detection-message wave *only* after getting sufficient evidence in support of termination detection condition from each of its children which in turn obtain the supporting information from their own respective children, if any. The detection-message wave either gets purged on the way or concludes termination on reaching back the root process.

Keywords: Distributed program, Termination detection, Message communication, Spanning tree.

1. Introduction

The development of low cost processor technology has contributed a great deal towards the development of distributed processing and has resulted in the emergence of many new problems. One such problem is concerned with the detection of termination of a distributed program [4]. The problem requires taking snap-shots of processes of the distributed program at different instants of time and then testing whether the program has terminated.

A number of algorithms employing different topological structures like an arbitrary network [2, 9], spanning tree [4, 5, 12] and unidirectional ring [1, 3, 6, 11, 13, 14, 15] have already appeared in the literature. In the present paper, we give an algorithm which employs spanning tree structure for detecting the distributed termination. In earlier reported solutions, employing this structure, apart from the drawback of freezing the basic computation [4], the root of the tree, after initiating the algorithm, remains busy in sending repeat detection-message waves to each of its children till it finally detects the distributed termination condition. We, however, depart from these and allow the root process to start the detection-message wave *only* after obtaining sufficient supporting evidence from each other process of the distributed program. This wave either gets purged on the way or concludes the termination on reaching back the process. Once the wave is purged, another wave is started by the root process only after obtaining fresh supporting evidence from each of its children. Other than this, the root process is treated like any other process of the program P.

After defining the distributed termination problem in Section 2, we give the algorithm and its correctness proof in Section 3 and then conclude with comments on the performance aspects of the algorithm.

2. Distributed Termination Problem

Consider a distributed program P consisting of n communicating sequential processes p_i, $1 \leq i \leq n$, at the nodes of a connected undirected graph. The processes complete main computation after a finite time, by exchange of messages called *basic* messages and local processing. The channels connecting the various nodes of the graph are such that they cater for unbounded buffers and error free messages. The messages are delivered in the order sent and the delay experienced by them is arbitrary but finite. An acknowledgement is expected (sent) for each basic message communication sent (received) by a process.

Each proces p_i, $1 \leq i \leq n$, has a local predicate C_i. A process can terminate only after it has satisfied its local predicate. However, a process can terminate only when all other processes of the program P have satisfied their respective local predicates simultaneously. This condition is called *Distributed Termination Condition* and hereafter referred to as DTC. The Distributed Termination Problem involves the

detection of an event when the DTC is satisfied.

A process is said to be *passive* when it has satisfied its local predicate; otherwise it is called *active*. An active process can make an already passive process active again by engaging the latter into basic communication. A passive process does not initiate any such communication but is always ready to receive messages from other active processes of the program P. Further, an active process can have basic communication only with a subset of the set of processes constituting the program P. The set of processes of program P, with whom an active process, say P_i, engages itself into basic communication is referred to as the set of neighbours of p_i.

3. Algorithm for Distributed Termination Problem

In the algorithm presented here we have employed the concept of neighbouring processes from our previous paper [15]. We use a fixed spanning tree of the given graph with the process, say p_1, at its root. Each process is augmented by a part, called control part, for the purpose of detecting Distributed Termination Condition [4]. These control parts communicate with one another only through exchange of messages called control messages. This control message communication is in addition to the basic message communication and is superimposed for the purpose of detecting the Distributed Termination Condition.

The algorithm for solving the Distributed Termination Problem has been developed in two different phases, viz. a detection phase, followed by a termination phase. Detailed logic for establishing the truth of the DTC has been developed in the detection phase whereas the processes of the distributed program P have been terminated in the termination phase.

3.1 Detection Phase

The detection phase basically employs three different types of control messages as detailed below:

(i) *I-am-through* – Issued by a non-root process upon satisfaction of its local condition.

(ii) *Detection-message* – Emitted by the root process for establishing the truth of suspected DTC.

(iii) *I-am-up-again* – Issued by a non-root process which has already sent the I-am-through message to its parent but now has become active again.

The processes are assumed to have unique identifications and control messages always carry the identification of their initiators.

Each process, in the algorithm, is required to maintain the following information:

(i) its own state information (active or passive)

(ii) the information about the state of its neighbours.

This information is maintained by modifying the code as follows:

'Whenever an active process sends a basic message to another process, both processes (sender as well as receiver) record the state of each other as active. Whenever a process becomes passive, it sends an I-am-passive message to each of its neighbours. Upon receipt of an I-am-passive message by a process from a neighbour, the process records the status of that neighbour as passive and remains passive, if already passive'. No acknowledgement is expected for an I-am-passive message.

Additionally, an internal/root process stores the information about the state of its respective children and keeps the record of valid detection-messages (to be explained later) received from them.

The state of a child process is recorded by its parent as passive when the parent received an I-am-through control message from it. The passive state is recorded as active when an I-am-up-again message is received from the child.

The root/internal process also maintains the value of the sequence number. For the root process, the sequence number (with initial value zero) is incremented each time a detection-message wave spreads out to its children, whereas an internal process notes down the value of the sequence number associated with the detection-message upon receiving it from its parent.

The underlying strategy, to be followed by the detection phase, is as follows:

A leaf process sends an I-am-through message to its parent only after its local condition is satisfied, i.e. when it finds that it is itself passive and all its neighbours have become passive.

An internal process sends an I-am-through message to its parent when it finds that it is itself passive, all its neighbours are passive and it has already received an I-am-through control message from each of its children.

Eventually, when the root process discovers that it has received an I-am-through message from each of its children, it is itself passive and all its neighbours have become passive, it increments the sequence number and emits a detection-message wave to each of its children.

Apart from this, whenever a non-root process becomes active, after having already sent an I-am-through message to its parent, it immediately sends an I-am-up-again control message to its parent which, in turn, records its status as active. The parent process then sends I-am-up-again message to its parent and the latter records the status of the former as active. This process continues till the message reaches the root process.

Upon receipt of a detection-message from its parent, an internal process notes down the sequence number associated with the detection-message and checks whether the process itself, all its neighbours and its children are still passive. Depending upon this, a detection-message is emitted to each of its children or gets purged.

In the event of a leaf process receiving a detection-message, a check is made to ascertain whether the process and all its neighbours are passive. If so, the detection-message is returned to its parent; otherwise it is purged.

When the detection-message is received by a process from one of its children, the following checks are conducted:

- The sequence number associated with the detection-message is tallied with that of the process.
- It is verified whether the process, its neighbours and all its children are still passive.

If both checks are met, the detection-message is considered to be *valid* and the necessary note, to this effect, is made in the process's control section. Otherwise the detection-message is purged. The process sends a detection-message to its parent only after receiving a valid detection-message from each of its children.

Finally, when the root process received valid detection-messages from each of its children, it concludes DTC and enters the termination phase.

The algorithm is now fully described by answering the following questions:

- What local condition is to be satisfied for initiating an I-am-through message by a process?
- What local condition is required to be satisfied by a root process for initiating a detection-message wave?
- When is an I-am-up-again message issued and how does it modify the control section information of its parent process?
- How is a response to an incoming detection-message generated by a process?

Algorithm

Step 1. **Whenever a process, say p_i ($1 \leq i \leq n$) becomes passive**
begin
state (p_i): = passive;
for each $p \in$ neighbours of p_i **do**
send I-am-passive *to* p
end

Step 2. **Upon receiving an I-am-passive message by a process p_i from some other process p ($p \in$ neighbours of p_i)**
begin
state$_i$ (p) : = passive;
If state$_i$ (p_j) = passive for all $p_j \in$ neighbours of p_i
and state (p_i) = passive
then
If p_i is a leaf-process
then begin
send I-am-through *to* parent (p_i);
/* parent (p_i) denotes parent of p_i */
$s(p_i)$: = '1'
/* $s(p_i)$ denotes a bit flag. Its value is initially 0. It becomes 1 when a non-root process sends an I-am-

```
                through message to its parent */
            end
        else
            If p_i is the root-process
            then begin
                If child_i (p_j) = passive for all p_j ∈ CH_i
                    /* CH_i denotes the set of children processes of p_i */
                then begin
                    seq (p_i) := seq (p_i) + 1;
                      /* seq (p_i) initially 0 */
                    seqdm = seq (p_i);
                    /* seqdm is the sequence number carried by the detection messages */
                  for each p ∈ CH_i do
                      begin
                      RCBDM_i (p) := false
                        /* RCBDM_i (p) would be used for keeping the record of receipt of valid detection-message for p */
                      send a detection-message to p
                      end
                    end
                else begin
                    If child_i (p_j) = passive for all p_j ∈ CH_i
                    then begin
                        send I-am-through to parent (p_i);
                        s(p_i) := '1'
                    end
                  end
        end
```

Step 3. **Upon receiving an I-am-through message by a process, say p_i, from some process p, $p \in CH_i$**

```
begin
    child_i (p) := passive;
    If state_i (p_j) = passive for all p_j ∈ neighbours of p_i
      and state (p_i) = passive
        and child_i (p_j) = passive for all p_j ∈ CH_i
    then
        If p_i is an internal-process
        then begin
            send I-am-through to parent (p_i);
            s(p_i) := '1'
            end
        else begin
            seq (p_i) := seq (p_i) + 1;
            seqdm := seq (p_i);
            for each p ∈ CH_i do
              begin
              RCBDM_i (p) := false
              send detection-message to p
              end
            end
end
```

Step 4. **Whenever a process, say p_i, change state from passive to active**

```
begin
    state (p_i) := active;
    If s(p_i) = '1'
    then begin
        If p_i is a leaf-process or p_i is an internal process
        then begin
            send I-am-up-again to parent (p_i);
            s(p_i) := 0
            end
        end
end
```

Step 5. **When a process p_i receives I-am-up-again message from p, $p \in CH_i$**

```
begin child_i (p) := active;
    If s(p_i) = '1'
    then begin
        send I-am-up-again to parent (p_i);
        s(p_i) := 0
        end
end
```

Step 6. **Upon receipt of the detection-message by a process p_i from a process p where p_i is either parent of p or $\in$ CH, the set of children of p**

```
begin
If state (p_i) = passive
  and state_i (p_j) = passive for all p_j ∈ neighbours of p_i
then
```

```
If p_i is the root-process
then begin
    If seq (p_i) = seqdm
    and child_i (p_j) = passive for all p_j
                        ∈ CH_i
     then begin
    RCBDM_i (p) = true ;
    If RCBDM_i (p_j) = true for all p_j
                        ∈ CH_i
    then enter the termination-phase
    else purge the detection-message
    end
else
    purge the detection-message
    end
else
    If p_i is a leaf-process
    then return the detection-message
    else
        If p_i ∈ CH
        then If child_i (p_j) = passive for
                all p_j ∈ CH_i
        then begin
            seq (p_i) := seqdm ;
            for each p_j ∈ CH_i do
              begin
              RCBDM_i (P_j) := false ;
              send a detection-message
                        to p_j
              end
            end
        else
            purge the detection-message
     else
    If seq (p_i) = seqdm
    then
        If child_i (p_j) = passive for all p_j ∈
                        CH_i
        then
            RCBDM_i (p) = true
            If RCBDM_i (p_j) = true for
                all p_j ∈ CH_i
            then
            send a detection-message to
                    parent (p_i)
        else
            purge the detection-message
      else
            purge the detection-message
        else
            purge the detection-message
    end
```

3.2 Termination Phase

Once the Distributed Termination Condition is established, the root process emits a termination-message wave, which spreads down the tree, for terminating the processes of the distributed program *P*.

The steps to be followed in terminating the processes are briefly given as below.

Step 1. **On detection of Distributed Termination Condition**

```
  begin
    for each p ∈ CH_i
    send a termination-message to p ;
    terminate
  end
```

Step 2. **on receipt of a termination-message by a non-root process p_i**

```
  begin
    If p_i is the internal-process
    then begin
        for each p ∈ CH_i
        send a termination-message to p;
        terminate
        end
    else
      terminate
  end
```

The State Transition Diagrams of leaf, internal and root processes are shown in *Figs. 1, 2 and 3* respectively.

3.3 Correctness of the Algorithm

To establish the correctness of the presented algorithm and to ensure that no deadlock situation arises and false termination of the distributed program *P* does not take place, we need to state and prove the following assertions.

1. The root process succeeds in initiating a detection-message wave.

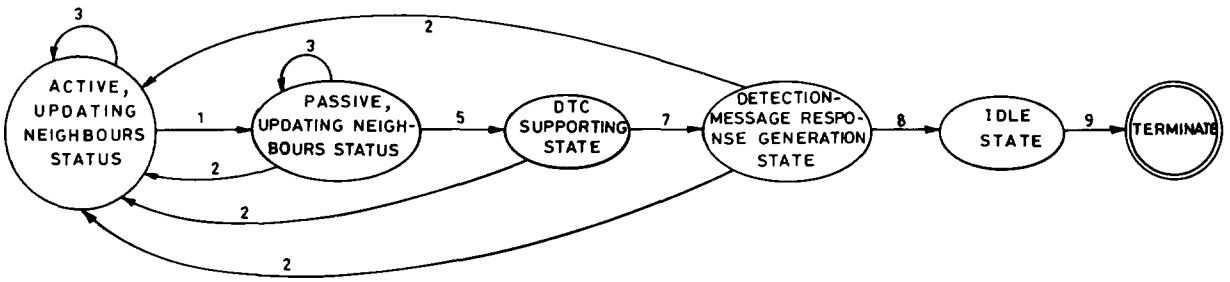

Fig. 1. State transition diagram of a leaf process p_i ($i \neq 1$).

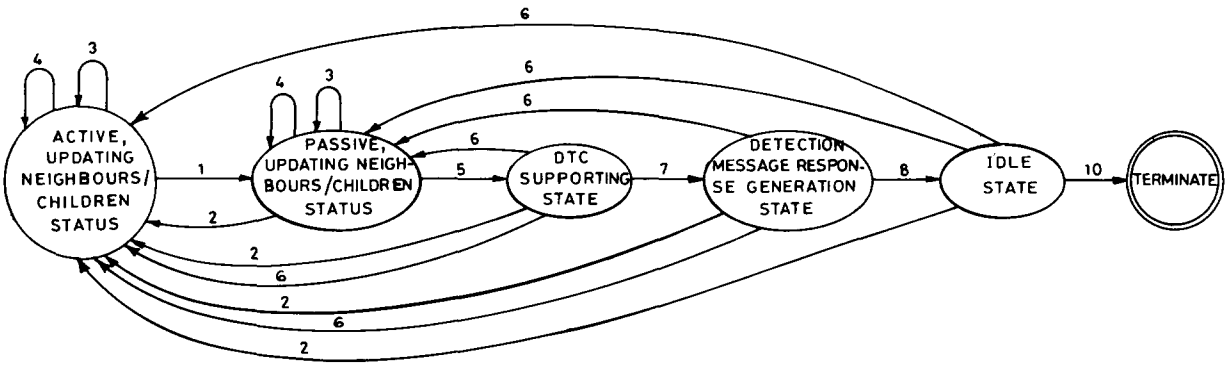

Fig. 2. State transition diagram of an internal process p_i ($i \neq 1$).

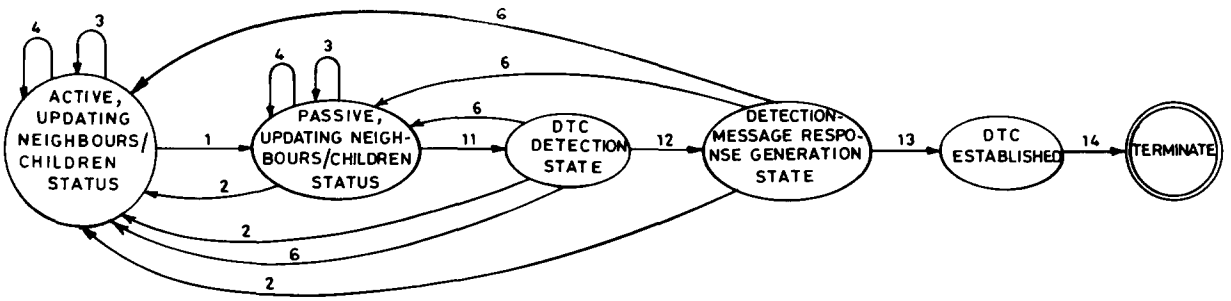

Fig. 3. State transition diagram of the root process p_1

The events representing edges in Figs 1, 2 and 3 are given as below:

1. $C_i \leftarrow$ True.
2. Receipt of basic communication (other than I-am-passive message) from an active process p_j.
3. Receipt of an I-am-passive message from its neighbours.
4. Receipt of I-am-through message from one of the children.
5. Sending of I-am-through message to its parent.
6. Receipt of I-am-up-again message from one of its children.
7. Receipt of detection message from the parent.
8. Sending of valid detection message to its parent.
9. Receipt of termination message from the parent.
10. Upon receipt of termination message from the parent & after sending the termination messages to each of its children.
11. Upon finding that the process, its neighbours and all its children are in passive-state.
12. Sending of detection-messages to its children.
13. On receipt of valid detection-message from each of its children.
14. After sending termination messages to each of its children.

2. The root process succeeds in entering the termination phase after detecting the truth of distributed termination condition.
3. There is no chance of detecting false distributed termination condition.

3.3.1 Proof of Assertion 1

Let p be the process to become passive last. It would then issue an I-am-passive message to each of its neighbours. Further, suppose that p_j is one of the neighbours of p. At some point of time, after p has become passive, p_j would have either received the I-am-passive message or the message would be in transit. The I-am-passive messages started by other neighbours of p_j may also be in transit. Eventually p_j would receive all its transit I-am-passive type messages, and then further action is taken as detailed below.

If p_j happens to be a leaf process, its local condition (viz. the process itself and its neighbours are passive) becomes true and this would in turn result in its sending an I-am-through message to its parent.

When p_j is an internal process, a check is made to find out whether it has already received I-am-through control messages from each of its children. If so, then an I-am-through message is sent straightway to its parent. Otherwise this is sent only when the transit I-am-through messages are eventually delivered to p_j.

On the other hand, when p_j happens to be the root process, a detection-message is issued to each of its children, immediately after p_j has received I-am-through messages from each of its children.

Finally, when p_j is not the root process, then as per stated sequence of events, I-am-through control messages are eventually delivered to the root process by each of its children processes and this results in the emition of a detection-message by the root-process.

Hence the assertion.

3.3.2 Proof of Assertion 2

Once the DTC becomes true, all processes of the distributed program P are in a passive state and the transit I-am-passive and I-am-through type messages eventually get delivered to their respective destination processes. Finally, when the root process discovers that it has received all I-am-through messages from each of its children and it has already received I-am-passive messages from each of its neighbours, then it increments the sequence number and issues a detection-message to each of its children.

Upon receiving a detection-message from a parent, an internal process notes down the sequence number associated with the message and spreads the detection-message downwards to each of its children. When such a message is received by a leaf process, it returns the detection-message to its parent.

When a valid detection-message is received by an internal process from one of its children, a note is kept and the message is purged. Subsequently, when the process discovers that it has received the valid detection-messages from all its children, it sends a detection-message to its parent. Continuing the argument like this, we find that the root process eventually receives the valid detection-messages from all its children and then enters the termination phase.

Thus we find that once the DTC becomes true, the root process eventually enters the termination phase.

3.3.3 Proof of Assertion 3

The assertion is proved by contradiction. Let the root process receive valid detection-messages from each of its children and conclude the truth of the Distributed Termination Condition. Further, suppose that there still exists an active process, say p_i which violates this stated conclusion.

Naturally, p_i would have become active only after emitting a detection-message to its parent after having received valid detection-messages from each of its children. This would mean that there is an active process, say p_j, which would have made p_i active after the latter already has emitted a detection-message to its parent. This situation cannot arise since according to our algorithm, a process sends an I-am-through message to its parent only after its local condition (viz. the process itself, its neighbours and all its children, if any, are passive) is satisfied. The root process, in turn, cannot start any detection-message wave until and unless it receives an I-am-through message from each of its children, which in turn obtain such information from their own re-

spective children, if any. Moreover, when a process which has already sent an I-am-through message to its parent, becomes active again as a result of basic communication from some other active process, it immediately sends an I-am-up-again message to its parent which, in turn, records its status as active. This process continues till the message finally reaches the root process. As a consequences of this, the detection-message wave initiated by the root process gets purged on the way. But according to our assumption, the root process receives valid detection-messages from its children; hence the contradiction.

3.4 Speed up of the Algorithm

The derived algorithm does not depend upon a particular form of a spanning tree. However, if the extracted spanning tree of the graph happens to be a balanced-tree, then the average time taken by the detection-message in travelling from the root to the leaves and back would certainly be less.

It is very likely that the connection of the graph is such that it is not possible to get a balanced-tree. In that case additional channels can be added to finally obtain a balanced-tree. However, in doing so, it is imperative to have a trade off between the overheads on account of addition of extra channels and the improvement in speed.

We propose to deal with the problem of complexity of the algorithm, depending upon the structure of the tree, separately.

4. Concluding Remarks

The solution reported by Topor [12] generates two waves of signals moving through the spanning tree. Initially a contracting wave of signals, called tokens, moves inwards from the leaves to the root. If this token wave reaches the root without discovering that termination has occurred, the root initiates a second, expanding, wave of repeat signals. As this repeat wave reaches the leaves, the token wave is gradually reformed and starts moving up again. This sequence of events is repeated until termination is determined.

The solution presented by us avoids such repetition and has been based upon the concept of communicating neighbours introduced in our earlier papers on ring based symmetric distributed termination algorithms [14, 15]. The detection-message in [14] is issued whenever a process changes state from active to passive, whereas in [15] less communication overheads have been incurred in allowing a process to issue a detection-message only when the process and all its communicating neighbours become passive. In the given algorithm, the root process issues a detection-message when the process itself is passive, and only after it obtains evidence that all its communicating neighbours including all its children have become passive. The detection-message wave once issued by the root process first spreads downwards and then contracts upwards. It either reaches back the root process, thereby determining the truth of the distributed termination condition, or gets purged on the way. The root process issues the next detection-message wave only after getting fresh evidence for suspecting the truth of the distributed termination condition. This results in considerable less control message traffic as compared to other reported strategies [4, 5, 12].

Further, no extra effort is required to maintain the local information in the control section of a process. The part of information in terms of keeping the status of neighbours of a process is automatically generated when basic communication takes place between the process and its neighbours, while the rest is generated when the control messages are received by the process.

References

[1] Arora, R.K., and Sharma, N.K.: A Methodology to Solve Distributed Termination Problem. *Information Systems*, Vol. 8, No. 1 (1983) 37–39.

[2] Dijkstra, E.W., and Scholten, C.S.: Termination Detection for Diffusing Computations. *Information Processing Letters*, Vol. 11, No. 1 (1980) 1–4.

[3] Dijkstra, E.W., Feigen, W.H.J., and Gasteren, A.J.M. van: Derivation of Termination Detection Algorithm for Distributed Computations. *Information Processing Letters*, Vol. 11, No. 5, (1983) 217–219.

[4] Francez, N.: Distributed Termination. *ACM-TOPLAS*, Vol. 2, No. 1 (1980) 42–45.

[5] Francez, N., and Rodeh, M.: Achieving Distributed Termination without Freezing. Tech. Rept. no. 180, Tech-

nion Israel Institute of Technology, Israel, 1980.

[6] Francez, N., Rodeh, M., and Sintzoff, M.: Distributed Termination with Interval Assertions. Tech. Rept. no. 186, Computer Science Department, Technion Israel Institute of Technology, 1980.

[7] Hoare, C.A.R.: Communicating Sequential Processes. *Comm. of ACM*, Vol. 21, No. 8 (1978) 666–677.

[8] Miller, L.L. and Liang, Tyne: The Role of Parallelism in File Organizations. *Proceedings of the 1985 ACM Computer Science Conference* (March 1985) 381–388.

[9] Misra, J., and Chandy, K.M.: Termination Detection of Diffusing Computations in Communicating Sequential Processes. *ACM TOPLAS*, Vol. 4, No. 1 (1982) 37–43.

[10] Lamport, L.: Time, Clock and Ordering of Events in a Distributed System. *Comm. of ACM*, Vol. 21, No. 7 (1978) 558–565.

[11] Rana, S.P.: A Distributed Solution of the Distributed Termination Problem. *Information Processing Letters*, Vol. 17, No. 1 (1983) 43–46.

[12] Topor, Rodney W.: Termination Detection for Distributed Computations. *Information Processing Letters*, Vol. 18, No. 1 (1984) 33–36.

[13] Apt, K.R., and Richier, J.L.: Real Time Clocks versus Virtual Clocks. Tech. Rept. no. 84–34, LITP, Université Paris 7, 1984.

[14] Arora, R.K., Rana, S.P., and Gupta, M.N.: Ring Based Termination Detection Algorithm for Distributed Computations. *Microprocessing and Microprogramming*, Vol. 19 (1987) 219–226.

[15] Arora, R.K., Rana, S.P., and Gupta, M.N.: Distributed Termination Detection Algorithm for Distributed Computations. *Information Processing Letters*, Vol. 22, 6 (1986) 311–314.

Dr. R.K. Arora is a Professor of Computer Science & Engineering and Head Computer Services Centre at IIT Delhi. Dr. Arora was associated with the software group of CDC-3600 and 160-A computer systems during 1963–67 at Tata Institute of Fundamental Research, Bombay. He has been a visisting research fellow at the Institute of Computer Science, University of London, London. He was a visiting professor to the University of Southampton, Oxford, Kent and Portsmouth Polytechnic, U.K. during 1978–79. Dr. Arora is author of various research publications in international journals in the area of distributed computing, operating systems, computer arithmatic and software engineering. His current research interests include distributed computing, computer networking, operating systems and database management systems. Dr Arora is a member of the British Computer Society and fellow of the Institution of Electronics & Telecommunications Engineers.

M.N. Gupta is currently associated with the Computer Center at the Indian Institute of Technology, Delhi. He is author of various research publications in international journals in the area of distributed computing. His current research interests are concurrent programming, computer networking and database management systems. He received his Master's degree in mathematics from Delhi University and DIIT from Indian Institute of Technology, Delhi.