Distributed Termination Detection in a Mobile Wireless Network

Jeff Matocha
jmatocha@cs.ua.edu
The University of Alabama
Tuscaloosa, AL 35487-0290

Abstract- In a mobile wireless network, restrictions exist which demand creative solutions to classical distributed problems. Distributed termination detection, the problem of deciding when the whole of a distributed computation has completed, is one such distributed problem. In this paper, an existing message optimal distributed termination detection algorithm is selected and modified to enhance its behavior in a mobile wireless network; the modifications entail reducing the number of messages received and transmitted by the mobile nodes involved in the distributed computation.

1 Introduction

A large percentage of the population today utilizes pagers and cellular telephones on a daily basis. Laptop computers have also seen a recent boom in sales. The union of these two technologies creates the need for protocols to maintain Internet connections while roaming. This new paradigm of networking presents many unique challenges.

As the surge in popularity of this new technology increases, "many experts are convinced that tomorrow's computers will all be mobile [13]!" Since a large number of computers will be mobile, researchers must examine all existing algorithms to meet the challenges which appear in the face of mobility. The research in this paper presents an approach to solving the problem of distributed termination detection (DTD) in a mobile wireless network. We begin in Section 2 with a presentation of the DTD problem as well as the Dijkstra and Scholten algorithm which achieves DTD [7].

Section 3 presents mobile computing and networking with an emphasis on the challenges which must be faced in order to create a mobile aware distributed algorithm. Our enhancements for making the algorithm by Dijkstra and Scholten mobile aware are presented in Section 4. Finally, Section 5 contains our conclusions and future work.

2 Distributed Termination Detection

Distributed termination detection is one of the classic problems in distributed systems research. Dijkstra and Scholten [7] and Francez [9] independently introduced and solved DTD in 1980. Throughout the 1980s, solutions to DTD frequently appeared in the literature [1, 4, 6, 10, 11, 12, 14, 15, 19, 20, 21, 23, 24, 25, 26, 28]. With proofs on the optimal bounds of the number of messages and concern over fault tolerance, DTD research has continued throughout the 1990s [3, 16, 17, 22, 27, 29].

Section 2.1 describes DTD and defines terms used in the remainder of this paper. We present the algorithm for Dijkstra and Scholten's solution to DTD in Section 2.2.

2.1 Problem Description

Termination is detected in a distributed system when a process determines that the distributed computation has completed. We define several terms in order to describe distributed systems and distributed computations in this section.

A distributed computation executes on a set of n processes, $P = \{p_0, p_1, \ldots, p_n\}$, which are distributed throughout a network. These processes are considered as nodes in a graph which are connected by a set of communication channels, E. We refer to the channels as edges due to their function in the definition of a distributed system as a graph.

The following three general assumptions hold for all distributed systems:

- There is no shared memory. Information must be transmitted between nodes via some channel in E.
- There is no common clock. Thus, there is no way to schedule a set of processes to perform an action at precisely the same moment.
- Communication takes arbitrary, but finite time. Since message transmission times are unpredictable, a process can not determine when a message is received by the recipient.

The above three assumptions complicate DTD. In fact, if any of the above assumptions did not hold, DTD would be trivial.

We view the processes in a distributed system as daemons that each perform a particular job. Thus, *active processes* are those currently working on a computation; *passive processes* are waiting. We will denote the set of messages as m; therefore, there are $|m|$ messages necessary for the computation and DTD. The messages for computation are called *basic messages* (m_B); the messages used for DTD are called *control messages* (m_C). Basic messages contain requests to the daemon at the intended process. The following restrictions, based on these process states and message types, describe the general action of a distributed system:

1. Initially, each process in the system is either active or passive.
2. An active process may become passive at any time.
3. Only an active process may send a basic message to another process.
4. A passive process becomes active only after receiving a basic message.

We can derive the necessary and sufficient conditions for DTD from these restrictions: termination has occurred when all processes in the system are passive and there are no basic messages in transit. DTD algorithms focus on the detection of such a state in which these necessary and sufficient conditions have been met.

Chandy and Misra proved that any DTD algorithm must use $O(|m_B|)$ control messages in its worst case execution in order to determine termination [5]. In choosing an algorithm for a mobile wireless network, a minimal number of messages necessary for DTD was preferred. For a taxonomy of DTD algorithms categorized by several features, see [18].

2.2 Dijkstra and Scholten's Solution

In 1980, Dijkstra and Scholten introduced the problem of DTD and proposed an elegant solution [7]. The algorithm of Dijkstra and Scholten (hereafter called the *DS algorithm*) performs similarly to the execution of a concurrent program; it is a call/reply or parental responsibility algorithm. In a concurrent program, the main procedure exits when it completes its main body and all of the procedures which it has called have exited. Furthermore, each called procedure may exit when it finishes the execution of its body and all of the procedures which it has called have exited. Similarly, a node in the DS algorithm exits when it has completed its computation and all the nodes to which it sent basic messages have completed. The DS algorithm is restricted to computations which begin at a single node, thus forming a computation tree.

In the DS algorithm, basic messages travel down the computation tree. Control messages are replies to basic messages and thus travel up the computation tree. Each node in the DS algorithm retains the total number of basic messages it has sent to its successors and the number of basic messages it has received from each of its predecessors. A basic message is *unreplied* if the receiver of the basic message has not returned a control message. The receiver returns a control message after it has completed the requested job and has no unreplied basic messages which were a result of the job. When the root node has no unreplied basic messages and is passive, termination is detected.

Pseudocode for the DS algorithm follows. Each node keeps `parent`, `passive`, and `pending` for each job.

Root node

Upon becoming `passive` or `pending` becoming 0:

```
if ((passive=True) and (pending=0))
  distributed termination has been detected;
```

Non-root nodes

Upon finishing job `m` or `pending` becoming 0:

```
if ((passive=True) and (pending=0))
  send control message to parent;
```

All nodes

Upon the arrival of a basic message `m` from `p_j`:

```
parent := p_j;
passive := False;
do job in m;
```

Upon sending a basic message to a child:

```
pending := pending+1;
```

Upon receiving a control message from a child:

```
pending := pending-1;
```

The DS algorithm requires exactly one control message for each basic message in the system. As stated in Section 2.1, optimality in terms of message complexity for a DTD algorithm is $O(|m_B|)$. The DS algorithm, which introduced the problem, is message optimal. Message complexity is important in choosing an algorithm for a mobile wireless network due to the expense of receiving and transmitting messages.

3 Mobile Computing and Networking

In a mobile wireless network, a computer capable of retaining attachment to a network while changing location is called a mobile node (*MN*). MNs wander throughout a set of *cells*, each of which is governed by a radio transmitter called a *Base*. Each Base is physically connected to the wired network and provides each MN within its cell a link-level point of attachment to the wired network. An example network illustrating these items is shown in Figure 1.

Forman and Zahorjan list three major challenges that must be considered before mobile computing and

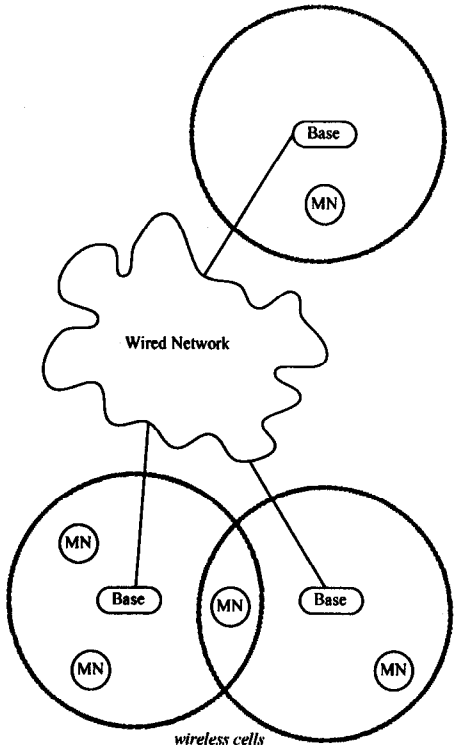

Figure 1: An example mobile network

networking become commonplace [8]. The three challenges are wireless issues (lower bandwidths, higher error rates, more frequent disconnections, and lower security than normal transmission media), portability issues (lower power and smaller storage capacity than non-mobile computers), and mobility issues (location management for the MNs). In a solution to DTD for a mobile wireless network, we consider the wireless and portability issues.

In any computation which includes MNs, there are three types of messages, each with a different cost. First, consider messages on the wired network (m_w); we consider wired network messages inexpensive since they travel on a physical medium. Next, consider messages which are sent to (received by) MNs (m_r); from an MN point of view, the power consumed by the receipt of a message is several times the power consumed by executing instructions. Last, consider messages which are transmitted by MNs (m_t); a MN transmission consumes several times the power consumed by receiving a message. In this paper we present modifications to the DS algorithm in order to reduce the number of m_t and m_r messages.

4 Mobile Enhancements to the DS Algorithm

Consider a distributed computation where all processes reside on MNs. A non-mobile aware implementation of Dijkstra and Scholten's DTD algorithm [7] requires $|m_r| = |m_t| = 2 \times |m_B|$. Each control message sent to an MN incurs one m_r; each control message sent by an MN incurs one m_t. Figure 2 depicts a simple distributed computation, the DTD of the simple distributed computation using the DS algorithm, and a timing diagram; solid arrows denote basic messages, dotted arrows denote control messages, the grey vertical line denotes a state of the system (such as DTD), and, with time progressing from left to right, a heavy line denotes activity and a thin line denotes passivity.

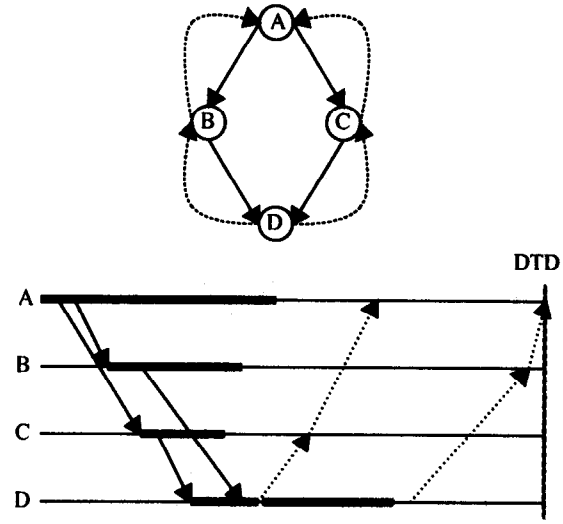

Figure 2: A graph and timing diagram for a simple distributed computation

In the DS algorithm, when node D completes node C's computation, node D immediately replies to node C. Since C has completed its job when it receives this control message from D, C immediately replies to A (its caller). Termination, however, can not be detected until D completes B's computation and a control message is sent to B and then to A. Since node A can not detect termination until D is completed, D could cache C's control message until it has completed all pending jobs. Therefore, our first improvement for converting the DS algorithm to a mobile wireless network allows MNs to cache control messages until completing their computations. Figure 3 illustrates the time diagram for our enhanced DS algorithm. Notice that the reply from D to node C and node B are combined into a single message. This message is sent to the Base for D where it is split into two m_ws. This improvement allows the MN to combine control messages to its Base which, in this example, reduces $|m_t|$ by one. In the best case, $|m_t|$ could be reduced to $n-1$, where n is the number of processes involved in the distributed computation.

Our first enhancement functions correctly when processes do not make recursive or mutually recursive calls. Consider, for instance, the computation and timing diagram in Figure 4. Node C is waiting on a reply from B, but B has cached its control messages until it is passive and has received replies from its children. Unfor-

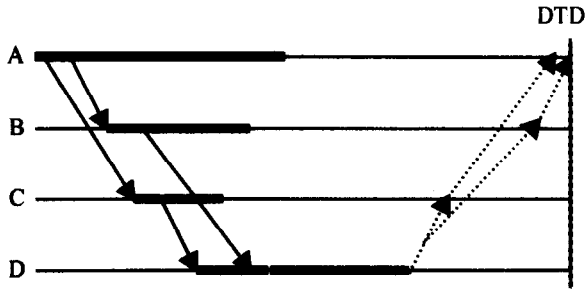

Figure 3: The enhanced time diagram for Figure 2

tunately, B is waiting on a reply from C; therefore, B is waiting (transitively) on its own reply. In the scheme developed thus far, if cycles exist, deadlock occurs. To prevent deadlock, we introduce a bitmap of size n which is added to each basic message (i.e., a small control message is piggybacked on each basic message). When node i sends a basic message, the node sets the ith bit in the bitmap it received for the current job and appends the bitmap to the message. At initialization, the root node creates a bitmap of size n with all zero entries. If node i receives a basic message with bit i set, it knows that a recursive (or mutually recursive) call has been made. When a node realizes it has received a recursive call, the job contained within the message is stored and the node immediately replies with a control message. The immediate reply does not remove the node from the computation tree, the immediate reply merely removes the cycle in the computation tree. Figure 5 exhibits the immediate reply to the example computation in Figure 4.

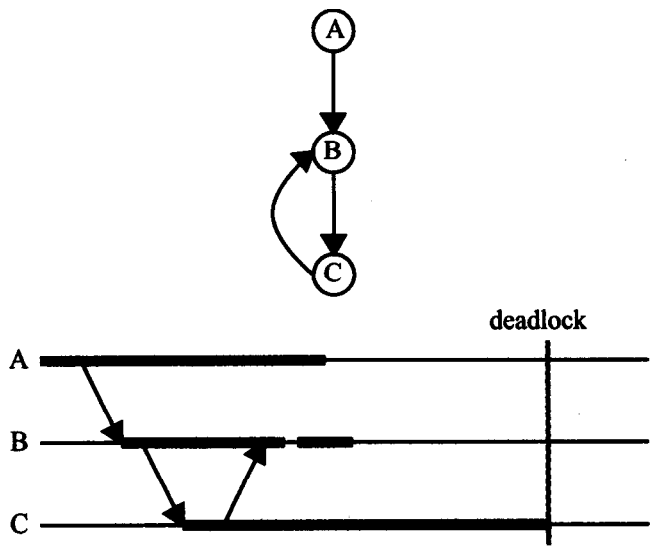

Figure 4: The effect of caching and recursive calls

Pseudocode for our first enhancement to the DS algorithm follows. Each node keeps `parent` and `replied` for each job as well as a single copy of `number_active` and `pending`.

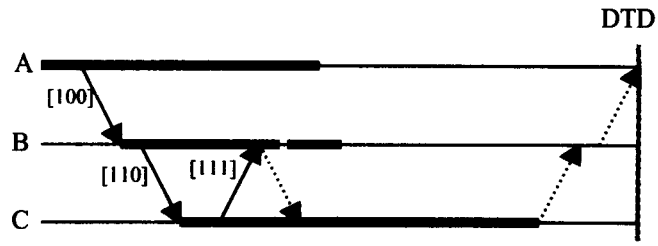

Figure 5: Immediate response of recursive calls illustrating bitmaps

Root node

Upon `number_active` becoming 0 or `pending` becoming 0:

```
if ((number_active=0) and (pending=0))
  distributed termination has been detected;
```

Non-root nodes

Upon finishing job `m`:

```
number_active := number_active-1;
```

Upon `number_active` becoming 0 or `pending` becoming 0:

```
if ((number_active=0) and (pending=0))
  send control message to all parents not
    already replied;
```

All nodes

Upon arrival of a basic message `m` with bitmap `b` for `p_j` from `p_k`:

```
replied := False;
parent := p_k;
number_active := number_active+1;
if (b has bit j set) {
  send control message to parent;
  replied := True;
}
do job in m;
```

Upon sending a basic message to a child:

```
pending := pending+1;
```

Upon receiving a control message from a child:

```
pending := pending-1;
```

Our first enhancement to the DS algorithm for a mobile wireless network never adds messages to the system. In the worst case, however, there is no reduction in the number of control messages transmitted. In other words, our first enhancement aids in practical matters only, not in theoretical bounds.

Although our first enhancement to the DS algorithm may reduce $|m_t|$, a penalty for this reduction is possible. Specifically, the caching of control messages may cause a

delay in the detection of termination by the root of the computation tree. Consider the computation graph and timing diagram in Figure 6. As in Figure 3, D caches its reply to C while D completes B's computation. Since B is a direct successor of A, a path of length two between D and A exists; since there are numerous intermediate nodes between A and C, the control message from D to C, which could have been sent while D was completing B's computation, traverse a path of length $n-2$ before reaching A. The issue at hand is the number of control messages which must be propagated between the time of actual termination and its detection. We call this distributed termination detection delay or *DTD delay*.

The original DS algorithm has a DTD delay between zero and $|m_B|$. In the best case, the last node to finish its computation is the root; in the worst case, the computation tree takes the form of a chain with cycles as in Figure 4. In this case, the calling chain must be unraveled, much like activation records are popped off the stack in the run-time memory of a typical procedural program. If recursive calls are not allowed, the maximum depth of a calling chain would be $n-1$, but since processes are allowed to recursively call one another, a chain with cycles may have a length of $|m_B|$.

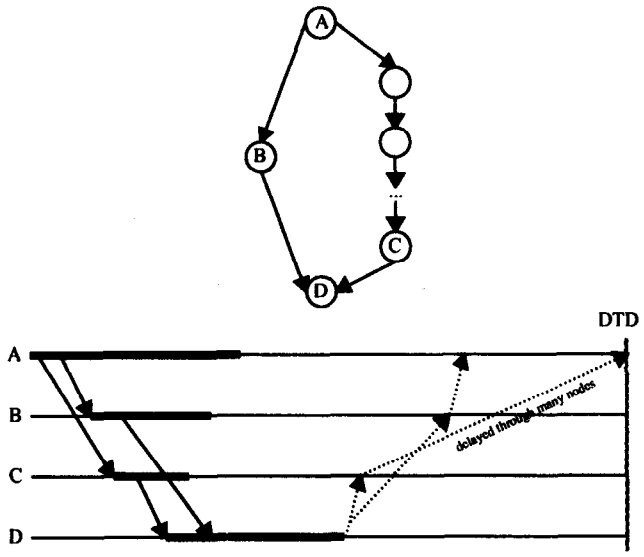

Figure 6: An example of a computation suffering from DTD delay

Although our first enhancement to the DS algorithm can cause a DTD delay greater than the original DS algorithm when the same distributed computation is run with both versions of the algorithm, the worst case DTD delay for the enhanced version of the DS algorithm is $O(n)$. Figure 6 depicts an example of a computation in which the difference in DTD delay between our enhanced algorithm and the original DS algorithm is $O(n)$. In our enhanced algorithm, D caches the reply to C while completing B's request. If D had not cached this control message (as in the original DS algorithm), the message may have reached the root (A) before D completed B's computation; in other words, the original algorithm may have DTD delay of two in this example. Our first enhancement to the DS algorithm has a DTD delay of $n-2$ in this example.

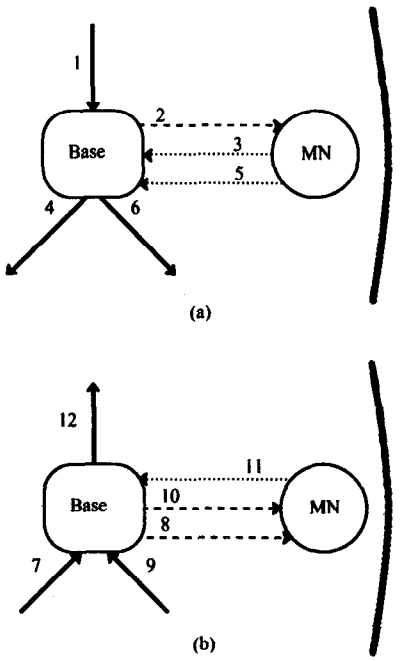

Figure 7: An execution example of the original DS algorithm
(a) basic messages
(b) control messages

The second enhancement to the DS algorithm reduces $|m_r|$. Figure 7 depicts an example of message transmissions at an MN which, upon receiving a basic message, sends basic messages to two other MNs; a heavy line denotes an m_w, a dashed lines denotes an m_r, and a dotted line denotes an m_t. If we assume the MNs involved in this distributed computation are in different cells, this scenario produces six m_ws (messages 1, 4, 6, 7, 9, and 12) which induce three m_rs (messages 2, 8, and 10) and three m_ts (messages 3, 5, and 11). A reduction in the number of control m_rs can be realized in an enhancement similar to that proposed by Badrinath, Acharya, and Imielinski [2]. These authors propose that as much of the work as possible is delegated to the Base. This delegation involves the Base retaining data structures for each MN under its care. In the DS algorithm, an MN can use its Base to retain the list of unreplied basic messages. The basic messages are still transmitted as m_ws, m_rs, and m_ts since the MN must obtain the information contained within the message. The Base peeks at the messages in order to retain the number of nodes from which the MN expects replies, as well as a list of nodes which are expecting a reply from the MN. When the MN becomes passive, it tells the Base. Figure 8 illustrates this enhancement with the simple example from Figure 7. Message 7 communicates to the Base that the MN has completed its computations; thus, after the responses arrive for the basic messages transmitted by the

MN, the Base transmits a control message to the MN's caller. In this example, two m_r messages are saved. Leaf nodes benefit from this enhancement only when they are called by more than one node, intermediate nodes benefit only when they are either called by more than one node or call more than one node, and the root node benefits only when it calls more than one node.

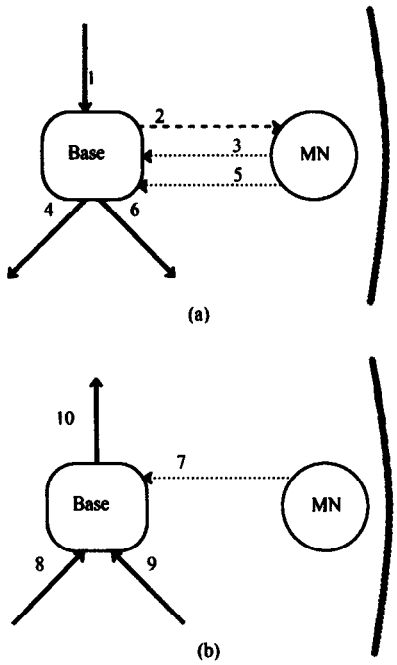

Figure 8: An execution example of the Badrinath enhancement to the DS algorithm
(a) basic messages
(b) control messages

Pseudocode for our second enhancement to the DS algorithm follows: Each node keeps `parent` and `replied` for each job as well as a single copy of `number_active` and `pending`. The variables `pending` and `number_active` are stored at the Base for each node and `parent` and `replied` are stored at the Base for each job.

Base of root node

Upon `number_active` becoming 0 or `pending` becoming 0:

```
if ((number_active=0) and (pending=0))
  distributed termination has been detected;
```

Base of non-root node

Upon `number_active` becoming 0 or `pending` becoming 0:

```
if ((number_active=0) and (pending=0))
  send control message to all parents not
    already replied;
```

Base of node p_j

Upon receiving a basic message `m` for `p_j` from `p_k`:

```
replied := False;
parent := p_k;
number_active := number_active+1;
if (b has bit k set) {
  send control message to parent;
  replied := True;
}
send job in m to p_j;
```

Upon receiving a control message from a child of `p_j`:

```
pending := pending-1;
```

Upon receiving a done message from `p_j`:

```
number_active := number_active-1;
```

Any MN

Upon receipt of a job from the Base:

```
do job;
```

Upon end of a job:

```
send done message to Base;
```

Upon sending a message to a child:

```
// nothing special
```

This second enhancement to the DS algorithm, like the first, does not change the message optimality of the algorithm. As stated above, the Badrinath enhancement does not always reduce the number of messages. There fore, the bounds on the number of messages continues to be $O(|m_B|)$. This enhancement, like our first enhancement, is a practical enhancement for DTD in a mobile wireless network.

5 Conclusions

As stated above, the Badrinath enhancement can be used in conjunction with the caching enhancement. Controversy exists, however, over whether Bases will provide the required service. If Bases do make such a service available, the use of the service may be based on a combination of the importance of the computation, the necessity of savings, and the cost of the service. The caching enhancement is available for use regardless of the service provided by the Base and should be utilized for a reduction of $|m_t|$.

The enhancements in this paper do not reduce the bound on the number of messages, but they do reduce the number of m_rs and m_ts required for termination detection in practice. Therefore, the bound on the number of messages continues to be $O(|m_B|)$; this bound has been proven as message optimal for DTD.

An area of future work includes converting other appropriate existing DTD algorithms to a mobile wire less network. DTD delay should be considered strongly since, in the mobile world, transmission delay due to error rates and disconnections expound DTD delay which exists in the algorithm.

References

[1] R. K. Arora and N. K. Sharma. A methodology to solve distributed termination problem. *Information Systems*, 8(1):37–39, 1983.

[2] B. R. Badrinath, A. Acharya, and T. Imielinski. Structuring distributed algorithms for mobile hosts. In *Proceedings of the 14th International Conference on Distributed Computing Systems*, pages 21–28, June 1994.

[3] S. Chandrasekaran and S. Venkatesan. A message-optimal algorithm for distributed termination detection. *Journal of Parallel and Distributed Computing*, 8:245–252, 1990.

[4] K. M. Chandy and J. Misra. An example of stepwise refinement of distributed programs: Quiescence detection. *ACM Transactions on Programming Languages and Systems*, 8(3):326–343, 1986.

[5] K. M. Chandy and J. Misra. How processes learn. *Distributed Computing*, 1(1):40–52, 1986.

[6] E. W. Dijkstra, W. H. J. Feijen, and A. J. M. van Gasteren. Derivation of a termination detection algorithm for distributed computations. *Information Processing Letters*, 16(5):217–219, 1983.

[7] E. W. Dijkstra and C. S. Scholten. Termination detection for diffusing computations. *Information Processing Letters*, 11(1):1–4, August 1980.

[8] G. Forman and J. Zahorjan. The challenges of mobile computing. *IEEE Computer*, pages 38–46, April 1994.

[9] N. Francez. Distributed termination. *ACM Transactions on Programming Languages and Systems*, 2(1):42–55, January 1980.

[10] N. Francez and M. Rodeh. Achieving distributed termination without freezing. *IEEE Transactions on Software Engineering*, 8(3):287–292, 1982.

[11] N. Francez, M. Rodeh, and M. Sintzoff. Distributed termination with interval assertions. In *Formalization of Programming Concepts: Lecture Notes in Computer Science 107*, pages 280–291, Pensicola, Spain, 1981. Springer-Verlag.

[12] S. Huang. A fully distributed termination detection scheme. *Information Processing Letters*, (28):13–18, September 1988.

[13] C. Huitema. *Routing in the Internet.* Prentice-Hall, 1995.

[14] D. Kumar. A class of termination detection algorithms for distributed computations. In *5th Conference on Foundations of Software Technology and Theoretical Computer Science*, volume LNCS 206, pages 73–100, New Delhi, 1985. Springer-Verlag.

[15] T. Lai. Termination detection for dynamically distributed systems with non-first-in-first-out communication. *Journal of Parallel and Distributed Computing*, 3:577–599, 1986.

[16] T. Lai, Y. Tseng, and X. Dong. A more efficient message-optimal algorithm for distributed termination detection. In *Proceedings of the 4th IEEE Symposium on Parallel and Distributed Processing*, pages 274–281, Arlington, Texas, December 1992.

[17] T. Lai and L. Wu. An $(n-1)$-resilient algorithm for distributed termination detection. *IEEE Transactions on Parallel and Distributed Systems*, 6(1):63–78, January 1995.

[18] J. Matocha and T. Camp. A taxonomy of distributed termination detection algorithms. *Journal of Systems and Software*, 1998. to appear.

[19] F. Mattern. Algorithms for distributed termination detection. *Distributed Computing*, 2:161–175, November 1987.

[20] F. Mattern. Experience with a new distributed termination detection algorithm. In *Proceedings of the 2nd International Workshop on Distributed Algorithms*, pages 127–143, Amsterdam, 1987.

[21] F. Mattern. Global quiescence detection based on credit distribution and recovery. *Information Processing Letters*, 30:195–200, 1989.

[22] J. Mayo and P. Kearns. Distributed termination detection with roughly synchronized clocks. *Information Processing Letters*, 52:105–108, 1994.

[23] J. Misra. Distributed termination of distributed computations using markers. In *Proceedings of the 2nd Annual ACM Symposium on Principles of Distributed Computing*, pages 290–194, Montreal, August 1983.

[24] J. Misra and K. M. Chandy. Termination detection of diffusing computations in communicating sequential processes. *ACM Transactions on Programming Languages and Systems*, 4(1):37–43, 1982.

[25] S. P. Rana. A distributed solution to the distributed termination problem. *Information Processing Letters*, 17:43–46, July 1983.

[26] R. W. Topor. Termination detection for distributed computations. *Information Processing Letters*, 18:33–36, January 1984.

[27] Y. Tseng. Detecting termination by weight-throwing in a faulty distributed system. *Journal of Parallel and Distributed Computing*, 25:7–15, 1995.

[28] S. Venkatesan. Reliable protocols for distributed termination detection. *IEEE Transactions on Reliability*, 38(1):103–110, April 1989.

[29] X. Ye and J. A. Keane. A distributed termination detection scheme. Technical Report UMCS-91-3-1, University of Manchester Department of Computer Science, Manchester, M13 9PL, England, 1991.