Termination Detection Protocols for Mobile Distributed Systems

Yu-Chee Tseng, *Member*, *IEEE Computer Society*, and Cheng-Chung Tan

Abstract—This paper studies a fundamental problem, the *termination detection* problem, in distributed systems. Under a wireless network environment, we show how to handle the *host mobility* and *disconnection* problems. In particular, when some distributed processes are temporarily disconnected, we show how to capture a *weakly terminated state* where silence has been reached only by those currently connected processes. A user may desire to know such a state to tell whether the mobile distributed system is still running or is silent because some processes are disconnected. Our protocol tries to exploit the network hierarchy by combining two existing protocols together. It employs the *weight-throwing scheme* [9], [16], [21] on the wired network side, and the *diffusion-based scheme* [5], [13] on each wireless cell. Such a hybrid protocol can better pave the gaps of computation and communication capability between static and mobile hosts, thus more scalable to larger distributed systems. Analysis and simulation results are also presented.

Index Terms—Distributed computing, distributed protocol, mobile computing, operating system, termination detection, wireless network.

1 Introduction

One major breakthrough in computer communication recently is the extension from wired to wireless transmission. Wireless communication products ranging from LAN, MAN, to WAN are available commercially [7], [8]. Another breakthrough in computing devices is the maturity of light-weight, economic, hand-held laptop and palmtop computers. This has made *mobile computing* (or *nomadic computing*) possible [11]. Users can carry computers (or *mobile hosts*) while moving around and remain in touch with networks.

Designing a distributed system with wireless communication components does pose some new challenges. First, distributed processes in mobile hosts have mobility. Second, mobile hosts are inherently weaker in computing capability than fixed hosts. Third, wireless communication has less (about an order or two) bandwidth and higher error rate compared to wired communication. These have interested a lot of researchers. Issues that have been considered on mobile distributed environments include message causal ordering [2], [18], [24], fault tolerance [1], distributed snapshot [23], and distributed checkpointing [12], [19].

In this paper, we study one fundamental problem, the *termination detection problem*, in a mobile distributed system. As a basic problem in operating system design, termination detection refers to the necessity of determining whether a set of distributed processes has entered a "silent" status where all processes are idle and no further computation is possible, taking the unpredictable message delays into account [20]. This problem was first identified by Dijkstra and Scholten [5], which has then inspired a lot of studies [4], [6], [9], [13], [14], [16], [17], [21], [22]. It has applications in diffusion computation [5], distributed workpool [15], and distributed garbage collection. It also serves a part in checking *stable states* (such as deadlock and token loss) in a distributed system [10], [20].

Under a wireless network environment, we propose a termination detection protocol that can handle the host mobility and disconnection problems. We assume that the mobility problem is not directly supported by the underlying protocol, and should be taken care of by the termination detection protocol. In particular, when some distributed processes are temporarily disconnected, we newly define a *weakly terminated state* where "silence" has been reached only by those currently connected processes in the distributed system. Our protocol can catch such a state so that a user will not be caught in the dilemma of wondering whether the mobile distributed system is still running or is silent because some mobile processes are disconnected. To exploit the network hierarchy, our protocol is in fact a combination of two existing protocols designed for static distributed systems. It employs the *weight-throwing scheme* [9], [16], [21] on the wired network side, and the *diffusion-based scheme* [5], [13] on each wireless cell. It is shown that such a hybrid approach can better pave the gaps of computation and communication capability between static and mobile hosts, thus more scalable to larger distributed systems. Through analyses and simulations, we demonstrate that our protocol places less demand of space and computing power on mobile hosts and of communication bandwidth on wireless links.

The rest of this paper is organized as follows: In Section 2, the distributed termination detection problem is formally defined and two existing protocols are reviewed. Our hybrid protocol is presented in Section 3, followed by

• *Y.-C. Tseng is with the Department of Computer Science and Information Engineering, National Chiao-Tung University, Hsin-Chu 30050, Taiwan. E-mail: yctseng@csie.nctu.edu.tw.*
• *C.-C. Tan is with the Department of Computer Science and Information Engineering, National Central University, Chung-Li 32054, Taiwan. E-mail: cctan@iii.org.tw.*

Manuscript received 14 July 1999; revised 26 Sept. 2000; accepted 7 Feb. 2001.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 110239.

comparisons and simulation results in Section 4. Conclusions are drawn in Section 5.

2 Preliminaries and Reviews

2.1 Distributed Termination Detection

A distributed system consists of a set of autonomous processes $S = \{P_1, P_2, , P_n\}$ which cooperate with each other to complete a job. Processes can communicate with each other by message-passing. Logically, from each P_i to each P_j, there is a communication channel $C_{i,j}$. A process may switch between two states: *active* and *idle*. A process, when performing some computation, is said to be in the active state. An active process is free to send/receive messages and may become idle spontaneously. On idle state, a process does not perform any computation, but can passively receive messages, on which event it becomes active immediately and starts computations. For distinction, computation carried out and messages transmitted by the system are called *basic computation* and *basic messages*, respectively.

A distributed system is said to be *terminated* iff 1) P_i is idle and 2) $C_{i,j}$ is empty, for all $1 \leq i, j \leq n$. (Condition 2 is necessary because message delays are unpredictable and any "hidden" message will wake the system up later.) When terminated, no distributed process can become active and perform any further computation. Due to the variation of processor speeds and the unpredictability of message delays, detecting such a status is usually nontrivial. In the literature, this is known as the *distributed termination detection* problem. Extra messages (typically called *control messages*) are sent and/or extra information is associated with basic messages to detect such a state.

2.2 Review: The Diffusion-Based Scheme

The diffusion-based scheme [5], [13] detects termination following the expansion of the basic computation. Below, we present the version by [13]. The basic idea is to maintain a logical tree connecting all active (and perhaps some idle) processes. The basic computation is assumed to start from process P_1, which is regarded as the root of the tree. The tree may expand or shrink dynamically. When the tree shrinks to P_1 only and P_1 is idle, the system is terminated.

The tree is maintained by the following data structures in each process P_i:

- par_i: the parent of P_i. Initially, it is NULL.
- $in_i[1..n]$: an array of integers, where $in_i[j]$ is the number of basic messages that have been received from P_j but have not been acknowledged. Initially, this value is 0.
- out_i: an integer indicating the number of basic messages that have been sent by P_i but have not been acknowledged. Initially, $out_i = 0$.

The detection scheme consists of four event-driven rules of the format "*event* $\Rightarrow$ *action*," where *action* is the program segment to be executed when *event* occurs.

```
A1. On $P_i$ sending a basic message to another process
            $\Rightarrow$ $out_i := out_i + 1$;
A2. $P_i$ receiving a basic message from $P_j$ $\Rightarrow$
        $in_i[j] := in_i[j] + 1$;
        if $(par_i = NULL) \wedge (i \neq 1)$ then $par_i := j$;
A3. On $P_i$ turning idle $\Rightarrow$
        $reply\_minor(i)$;
        if $(out_i = 0)$ then $reply\_major(i)$;
A4. $P_i$ receiving an $ACK(k)$ $\Rightarrow$
        $out_i := out_i - k$;
        if ($P_i$ is idle) $\wedge$ $(out_i = 0)$ then $reply\_major(i)$;
```

For the tree to shrink, acknowledgments should be sent. We call a message a *major* message if it was from P_i's parent, otherwise, it is a *minor* message. Minor messages can be acknowledged whenever P_i turns idle, but major messages can only be acknowledged when P_i is idle and all P_i's outgoing messages have been acknowledged. The following procedures are for this purpose:

```
Procedure $reply\_minor(i)$
begin
        for each $j \neq par_i$ such that $in_i[j] \neq 0$ do
        else
                send an $ACK(in_i[j])$ to $P_j$;
                $in_i[j] := 0$;
        end for;
end.
Procedure $reply\_major(i)$
begin
        if (i = 1) then report termination
                send an $ACK(in_i[par_i])$ to process $par_i$;
                $in_i[par_i] := 0$;
                $par_i := NULL$;
        end if;
end.
```

2.3 Review: The Weight-Throwing Scheme

According to the problem definition, to detect termination, we mainly need to collect two kinds of information: 1) the idleness of processes and 2) the emptiness of communication channels. The *weight-throwing* scheme [9], [16] collectively represents them using one notion called *weight*. A weight is simply a real number, which can either be held by a process or be appended at a basic message while transmitted.

Initially, a predesignated process P_c (called *weight collector*) holds a weight $w_c = 1$ and all other process $P_i, i \neq c$, holds a weight $w_i = 0$. Process P_c is typically the one who starts the basic computation. It also serves as the central coordinator for termination detection. The protocol is derived based on a weight-invariant concept. It consists of four rules:

```
B1. On $P_i$ sending a basic message to another process $\Rightarrow$
        partition $w_i$ into two positive reals $x$ and $y$ such that
                $x + y = w_i$;
        append the weight $x$ to the basic message;
        $w_i := y$;
B2. On $P_j$ receiving a basic message with weight $x$ $\Rightarrow$
                $w_j := w_j + x$;
```

B3. On $P_i, i \neq c,$ switching from active to idle $\Rightarrow$
send a weight-report message $WGT(w_i)$ to P_c;
$w_i := 0$;

B4. On P_c receiving $WGT(w_i)$ from $P_i \Rightarrow$ $w_c := w_c + w_i$;

Throughout, two important invariants are preserved by the protocol:

1. Every active process holds a positive weight. Every in-transit basic and control message also holds a positive weight.
2. The sum of weights held by P_c, all active processes, all in-transit basic, and all in-transit control messages, is equal to 1 at the same time.

Therefore, when P_c becomes idle and finds a weight of 1 at its hand, no processes can be active and no messages can be in-transit. So termination of the system can be announced.

2.4 Discussion

Below, we compare the strength and weakness of these two schemes, intending to motivate the work (a hybrid protocol) in this paper.

Space Complexity. The diffusion-based scheme needs an array of size $O(n)$ (array $in[\,]$) in each process, where n is the size of the distributed system, while the weight-throwing scheme has a space complexity of $O(1)$ in each process. So the latter is more scalable in system size.

Computing Compelxity. Integers are used to maintain the tree T in the diffusion-based scheme, while floating numbers are used to represent weights in the weight-throwing scheme. Note that the regular hardware-supported floating-point arithmetic (which usually incurs truncating/rounding errors) should not be used with the weight-throwing scheme. Weights must be lossless and must be precisely represented to ensure correctness. This implies a software-emulated floating-point arithmetic and sometimes weight borrowing to take care of the repeated weight-split and combine problems [9], [16]. The weight calculation is computationally more costly.

Time Complexity. Let's define *detection delay* to be the time that a protocol takes to correctly report the termination status after a distributed system is terminated. In the diffusion-based scheme, this is reflected by the height of T and thus the detection delay is $O(n)$. The delay for the weight-throwing scheme is $O(1)$ as weights are always sent directly to the weight collector.

Communication Bandwidth. Both schemes were proved to be asymptotically optimal in terms of the number of control messages sent per basic message [13]. Looking in more details, we see that the weight-throwing scheme needs to append a weight on each basic message while there is no such cost for the diffusion scheme. However, counting the number of control messages being sent each time a mobile process turns idle, we see that exactly one $WGT()$ message is sent by the weight-throwing scheme, while the diffusion scheme will send n_{maj} and n_{min} $ACK()$'s to acknowledge major and minor messages, respectively, where $n_{maj} =$ 0 or 1, $n_{min} \geq 0$, but $n_{maj} + n_{min} \geq 1$.

3 A Hybrid Termination Detection Protocol

3.1 System Model

A distributed system consists of a set of processes $\{P_1, P_2, \ldots, P_n\}$. For distinction, a process P_i on a static host will be called a *static process* and denoted as P_i^s and that on a mobile host called *mobile process* and denoted as P_i^m. We consider a network architecture consisting of a wired part as the backbone connecting to a set of base stations $\{P_1^b, P_2^b, \ldots, P_m^b\}$. Communications to and from a mobile process must be relayed by a base station. The transmission range covered by a base station is called a *cell*. Each base station serves as a mobile-support station (MSS) [1], [2], [3], [19], [24] and can cooperate with the distributed system to detect the terminated status of the system. In general, any P_i^s, P_i^m, and P_i^b can be referred to as a "process." Message transmission between any two processes is reliable, but unpredictable, and follows a FIFO (first-in-first-out) model.

A mobile process can roam around any base station, and may roam off its current base station and become *temporarily disconnected*. A disconnected process can still perform computation, but all communication jobs to and from it should be suspended for future processing. We thus define two kinds of termination statuses as follows:

Definition 1. *A distributed system is strongly terminated if all processes are idle and there is no in-transit basic message. It is called weakly terminated if such a state is reached by all processes except disconnected processes.*

Knowing a system has entered a weakly terminated state is better than keeping a user waiting uncertain of the system's current state. (Still running? Or some mobile hosts being disconnected?) Our yet-to-be-presented protocol can even determine which processes have reached such a state.

3.2 Basic Idea and Data Structures

Our protocol is a hybrid of the diffusion-based and weight-throwing schemes. On each wireless cell, the diffusion-based scheme is used. While on the wired networks, the weight-throwing scheme is used. The base stations work in between to bridge these two protocols together. Our intention is to impose less burden (including storage, computation, and communication) on mobile processes, but not to sacrifice efficiency in the termination detection job. Intuitively, the weight-throwing scheme is better in both space and detection latency when the network scale is large, but it has to deal with real numbers. The diffusion scheme deals with only integer and, when applied on a wireless cell, will be a fine choice because all mobile hosts within a cell can talk to each other directly in one-hop. This is why we make such combinations.

As the weight-throwing scheme is used on the wired network, each static process P_k^s should keep a weight w_k. We assume that there is a special static process called P_c^s serving as the weight collector (alternatively, we can let some base station P_c^b play this role). Process P_c^s has an initial weight of $w_c = 1$. It also serves as the starter of the distributed computation (or, any process intending to begin a distributed computation cannotify P_c^s to "start" the computation by sending a virtual basic message to it). In

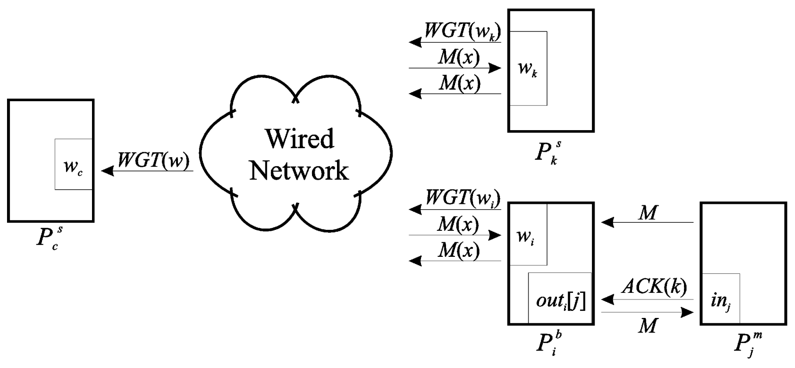

Fig. 1. The data structures and messages used by our hybrid termination detection protocol.

addition, to handle the mobility issue, P^s_c needs three new variables:

- Ψ: the set of temporarily disconnected processes.
- ψ: the total weight held by the processes in Ψ.
- $out_c[1..n]$: to store the numbers of unacknowledged basic messages for those temporarily disconnected processes in Ψ.

On each mobile process P^m_j, a simplified diffusion-based scheme is used. Only one variable is kept by P^m_j:

- in_j: the number of basic messages that have been received (from base stations) but have not been acknowledged.

Note that the pointer to P^m_j's parent and the variable out used in the original diffusion-based scheme are not needed any more. Further, the array $in_j[1..n]$ is now reduced to a scalar. Intuitively, this is because a mobile process always regards its current base station as its parent, to which all communications will be addressed.

Each base station P^b_i needs to run both protocols and, thus, will keep the following variables:

- w_i: the weight P^b_i holding at hand.
- Φ_i: the set of mobile processes currently supported by P^b_i.
- $out_i[1..n]$: an array of integers, where $out_i[j]$ is the number of basic messages that have been sent to $P^m_j \in \Phi_i$, but have not been acknowledged.

Note that on the contrary, the scalar out_i in the original scheme is now extended to an array. Intuitively, this is for the purpose of detecting the weakly terminated state.

Some of these variables are illustrated and related by messages in Fig. 1. The message types to be used in our protocol are listed below.

- M: a basic message; if M is appended with a weight w, we will write $M(w)$.
- $WGT(w)$: a message reporting a weight of w.
- $ACK(k)$: an acknowledgment of k basic messages.
- HF: handoff-related message which contains four subtypes: $HF.req, HF.ind,$ $HF.reply,$ and $HF.ack.$
- $DISC(P^m_k, i, w)$: a message associated with a weight w to report that a mobile process P^m_k is disconnected and has i unacknowledged basic messages.

3.3 The Protocol

3.3.1 Static and Weight-Collecting Process

Each static process, including the weight collector P^s_c, runs the original weight-throwing rules B1–B4 in Section 2.3. (How P^s_c announces termination is in Section 3.3.7.)

3.3.2 Mobile Processes

Each mobile process P^m_j runs a simplified diffusion-based scheme by always assuming its current base station as its parent and imagining all basic messages as being issued from or addressed to the base station.

C1. On P^m_j sending a basic message to another process $\Rightarrow$ do nothing;
C2. On P^m_j receiving a basic message from its base station $\Rightarrow$ $in_j := in_j + 1$;
C3. On P^m_j turning idle $\Rightarrow$ *reply*();

Compared to the original diffusion-based scheme in Section 2.2, A1 in fact turns to a null action (C1 is presented here only for clarity), A2 is simplified, and A3 and A4 are merged into a simpler rule. There is no concept of major or minor messages within a cell, so $reply\_major()$ and $reply\_minor()$ are merged into a simpler $reply()$ as follows:

```
Procedure $reply()$
begin
        Send an $ACK(in_j)$ to its base station;
        $in_j := 0$;
end.
```

3.3.3 Base Stations

Each base station P^b_i serves as a bridge between the wired and wireless networks. It first tries to detect the termination of all its local mobile processes based on the diffusion protocol. On finding this being true, it reports to P^s_c following the weight-throwing protocol:

```
D1. On $P^b_i$ receiving an $M(x)$ from the wired networks
                        destined to a $P^m_j \in \Phi_i$ $\Rightarrow$
        $w_i := w_i + x$;
        $out_i[j] := out_i[j] + 1$;
        Forward $M$ (without appending $x$) to $P^m_j$;
D2. On $P^b_i$ receiving an $M$ from a local mobile process $\in \Phi_i$
                        destined to a process $P_j$ $\Rightarrow$
        if $P_j \in \Phi_i$, then        /* for a local mobile process */
```

```
            $out_i[j] := out_i[j] + 1;$
            Forward $M$ to $P_j^m$;
    else        /* for a nonlocal process */
            Forward $M(w_i/2)$ to $P_j$;
            $w_i := w_i/2;$
    end if;
D3. On $P_i^b$ receiving an $ACK(k)$ from a $P_j^m \in \Phi_i \Rightarrow$
        $out_i[j] := out_i[j] - k;$
    if ($out_i[x] = 0$ for all $P_x^m \in \Phi_i$) then
            Send a $WGT(w_i)$ to the weight collector $P_c^s$;
            $w_i := 0;$
    end if;
```

The above rules guarantee two important properties:

Property 1. $(out_i[j] = 0) \Longrightarrow$ *(*$P_j^m \in \Phi_i$ *is idle)* $\wedge$ *(there is no in-transit basic message between* P_i^b *and* P_j^m*).*

Property 2. $(w_i = 0) \Longrightarrow$ *(each* $P_j^m \in \Phi_i$ *is idle)* $\wedge$ *(there is no in-transit basic message between* P_i^b *and each* $P_j^m \in \Phi_i$*).*

To prove Property 1, first observe that when $out_i[j] = 0$, all basic messages sent to P_j^m have been acknowledged, so P_j^m must be idle and the channel from P_i^b to P_j^m is empty. By C3, when P_j^m turns idle, the last message sent out by it must be an $ACK()$. As the channel is FIFO, this $ACK()$ will flush all basic messages sent from P_j^m to P_i^b. So the channel from P_j^m to P_i^b must be empty and this proves Property 1. Property 2 can be proven by examining each $P_j^m \in \Phi_i$ based on Property 1. It follows that D3 will lead to Property 2.

3.3.4 Hand-Off of Mobile Processes

When a mobile process P_k^m is handed off from its current base station P_i^b to another P_j^b, the following rules are used: Fig. 2 illustrates these steps.

```
E1. On $P_k^m$ intending to be hand-off from $P_i^b$ to $P_j^b \Rightarrow$
        $P_k^m$ sends a $HF.req(P_i^b)$ to $P_j^b$ and waits for $P_j^b$'s
                $HF.ack$ until it times out;
        $P_k^m$ stops sending and receiving any message until
                $HF.ack$ is received;
E2. On $P_j^b$ receiving $HF.req(P_i^b)$ from $P_k^m \Rightarrow$ $P_j^b$ sends a
                $HF.ind(P_k^m)$ to $P_i^b$;
E3. On $P_i^b$ receiving $HF.ind(P_k^m)$ from $P_j^b \Rightarrow$
        $\Phi_i := \Phi_i - \{P_k^m\};$
    if ($out_i[k] = 0$) then /* $P_k^m$ idle /*
            Send a $HF.reply(P_k^m, out_i[k], 0)$ to $P_j^b$;
    else if ($out_i[x] = 0$ for all $P_x^m \in \Phi_i$) then /* processes
                in $\Phi_i$ idle */
            Send a $HF.reply(P_k^m, out_i[k], w_i)$ to $P_j^b$;
            $w_i := 0;$
    else /* some processes in $\Phi_i$ active */
            Send a $HF.reply(P_k^m, out_i[k], w_i/2)$ to $P_j^b$;
            $w_i := w_i/2;$
    end if;
E4. On $P_j^b$ receiving $HF.reply(P_k^m, out_i[k], w)$ from $P_i^b \Rightarrow$
        $\Phi_j := \Phi_j \cup \{P_k^m\};$
        $out_j[k] := out_i[k];$
        $w_j := w_j + w;$
        Send a $HF.ack$ to $P_k^m$;
```

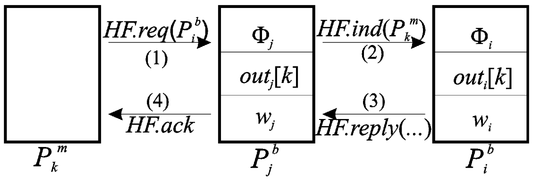

Fig. 2. The hand-off procedure.

Note that the above four steps may not always complete successfully. That is, P_k^m, after performing E1, may leave base station P_j^b before the $HF.ack$ returns. Thus, a *timeout* parameter should be set to deal with this problem. In this case, we regard that P_k^m has been handed off to P_j^b, although P_k^m has not been committed of this event. Subsequent steps E2 and E3, which only involve static processes, can complete successfully and, thus, migrate P_k^m's data structure from P_i^b to P_j^b. After the timeout period, if P_k^m fails receiving $HF.ack$ (due to mobility), it is free to hand-off to another base station by restarting E1. It should indicate its original base station as P_j^b instead of P_i^b. This will prevent P_k^m from being blocked due to high mobility. Also note that rule E1 ensures that P_k^m can only exchange messages with a base station that is formally committed (by $HF.ack$) as its current base station. Hence, the message exchange history will be recorded correctly (by array $out[\,]$) and mobility will not disrupt our protocol.

3.3.5 Temporary Disconnection and Rejoining of Mobile Processes

A reasonable assumption for disconnection is that such a status should be determined by each mobile host and base station *autonomously*. Specifically, when a mobile process finds that it is not connected to any base station, it can decide to enter a disconnected mode at its own will. Similarly, a base station determines that a mobile process has entered a disconnected mode if it finds 1) that it cannot hear from the process and 2) that there is no hand-off request from this mobile process for a predefined time.

A disconnected process should stop any message exchange (but local computation can continue). On the other side, a base station can return the disconnected process's data structure to the weight collector (in hope of reaching a conclusion that the distributed system is weakly terminated).

```
F1. On a $P_k^m$ deciding to enter a disconnected mode $\Rightarrow$ $P_k^m$
                suspend all message exchange;
F2. On a $P_i^b$ deciding that a $P_k^m \in \Phi_i$ has entered a
                disconnected mode $\Rightarrow$
        $\Phi_i := \Phi_i - \{P_k^m\};$
    if ($out_i[k] = 0$) then /* $P_k^m$ idle */
            Send a $DISC(P_k^m, out_i[k], 0)$ to $P_c^s$;
    else if ($out_i[x] = 0$ for all $P_x^m \in \Phi_i$) then /* processes
                in $\Phi_i$ idle */
            Send a $DISC(P_k^m, out_i[k], w_i)$ to $P_c^s$;
            $w_i := 0;$
    else /* some processes in $\Phi_i$ active */
            Send a $DISC(P_k^m, out_i[k], w_i/2)$ to $P_c^s$;
            $w_i := w_i/2;$
    end if;
```

```
F3. On $P_c^s$ receiving $DISC(P_k^m, out_i[k], w) \Rightarrow$
        $\Psi := \Psi \cup \{P_k^m\}$;
        $out_c[k] := out_i[k]$;
        $\psi := \psi + w$;
```

F2 manages the weights based on different combinations of processes' states. F3 appends the received weight to ψ. Depending on the value of ψ, the weight collector will try to determine the strongly/weakly terminated state of the system (refer to Section 3.3.7).

In our protocol, disconnection is handled very similar to hand-off. When a mobile process P_k^m in a disconnected mode hears any base station again, it can enter a connected mode again and execute the hand-off rule, E1, in the previous section to rejoin the system. Suppose P_k^m's original and new base stations are P_i^b and P_j^b, respectively. On hearing P_k^m's hand-off request, if P_i^b still has P_k^m's data structure (i.e., $P_k^m \in \Phi_i$), then the normal hand-off procedure takes action (by executing rules E2-E4). Otherwise, if $P_k^m \notin \Phi_i$, the hand-off request will be forwarded to the weight collector P_c^s and the following rule will be executed:

```
F4. On $P_c^s$ receiving the forwarded $HF.ind(P_k^m)$ from $P_i^b \Rightarrow$
        if ($P_k^m \in \Psi$) then
            $\Psi := \Psi - \{P_k^m\}$;
            if ($out_c[k] = 0$) then Send a
                    $HF.reply(P_k^m, out_c[k], 0)$ to $P_j^b$;
            else /* $P_k^m$ active */
                Send a $HF.reply(P_k^m, out_c[k], \psi/2)$ to $P_j^b$;
                $\psi := \psi/2$;
                if ($\Psi = \emptyset$) then $w_c := w_c + \psi$;
            end if;
        end if;
```

Note that the "if ($\Psi = \emptyset$) then ..." statement is to ensure that no weight is left in ψ if there is no disconnected process. Also, note that the above indices, i and j, do not have to be different. The modification is straight forward: We simply let P_i^b play both roles.

Finally, when the $HF.reply$ from the weight collector arrives at base station P_j^b, rule E4 will take action like a normal hand-off. A note similar to the hand-off procedure is that the above steps may not always complete successfully after P_k^m performs E1. Still, we regard that P_k^m has been hand-off to P_j^b, although P_k^m has not been committed of this event. No message exchange should be done with P_k^m in the meantime. The subsequent steps, which are on wired networks, will eventually deliver $HF.reply$ to P_j^b. If by the time P_k^m has migrated to another base station or become disconnected again, P_j^b simply serves as P_k^m's current base station and performs the above rules at its own determination. This ensures that temporary disconnection will not disrupt our protocol.

3.3.6 Dangling Messages

Dangling messages are those that cannot be delivered due to disconnection or host mobility. Since dangling messages may be delivered at any time later, care must be taken to avoid terminated states being falsely announced.

Dangling messages could exist in a mobile host when it is experiencing hand-off or disconnection. Since we always require that a base station obtain a mobile host's data structure before exchanging any message with it (observe the guarded conditions in D1-D3), these dangling messages will be kept locally in the mobile process until a formal hand-off is committed. So the message exchange history is always correctly recorded.

Dangling messages could also exist in a base station or a static process if the messages are destined to a hand-off or disconnected mobile process. Ensured by the weight-throwing rules, each process holding a dangling message must have a nonzero weight at hand. A reasonable approach is to forward dangling messages to the weight collector *as if* they are destined to it. A nonzero weight should be appended to each such message. The weight collector includes these weights into ψ and queues these messages for future processing:

```
G1. On $P_c^s$ receiving a dangling message carrying a weight
                    $x \Rightarrow \psi := \psi + x$;
G2. On $P_c^s$ sending a dangling message $M$ to a reconnected
                    mobile process $\Rightarrow$
        Append a weight $\psi/2$ to $M$;
        $\psi := \psi/2$;
        if (dangling message list becomes empty) then
            $w_c := w_c + \psi$;
            $\psi := 0$;
        end if;
```

3.3.7 Announcing Termination

The weight collector P_c^s can announce that the distributed system has entered a strongly or weakly terminated state once it has collected a sufficient weight of 1.

```
H1. On $P_c^s$ finding $w_c + \psi = 1 \Rightarrow$
        if ($\psi > 0$) then announce "weakly terminated
                    (excluding processes in $\Psi$)"
        else announce "strongly terminated";
```

A weakly terminated state guarantees that all currently connected processes are idle and there are no in-transit basic or dangling messages. It would be easier to design the application programs on top of the termination detection protocol because without the weight collector's permit (rule F4), no further computation in a weakly terminated system is possible. There are two possibilities for a weakly terminated state: 1) some dangling basic messages at P_c^s's hand and 2) some disconnected processes not acknowledging receipt of basic messages yet. In either case, the weight collector will have a nonzero ψ. On the other hand, note that the system can be strongly terminated even if some mobile processes are disconnected ($\Psi \neq \emptyset$), but the whole system is terminated for sure.

4 Comparisons and Simulation Results

A comparison based on a nonmobile system is in Table 1 (i.e., we assume that there are no host mobility and disconnection problems for the diffusion-based and weight-throwing schemes). Since we assume that mobile hosts are inherently weaker than static hosts, the comparison focuses on the space and computation complexity on mobile processes and the wireless bandwidth consumed.

TABLE 1
Comparison of Existing Hybrid and Our Hybrid Termination Detection Protocols for Mobile Processes

	space	computation	det. delay	ctl. msg.	comment
diff.-based	$O(n)$	integer	$O(n)$	$2n_{maj} + 2n_{min}$	
wgt.-throw	$O(1)$	lossless real weight	$O(1)$	2	append wgt. on basic msg.
hybrid	$O(1)$	integer	$O(1)$	1	

"Space" indicates the size of data structures used by each protocol. "Computation" indicates the data types to be processed for the purpose of termination detection (weight representation should be lossless, refer to Section 2.4). "Detection delay" is to count the maximum number of *communication hops* happening on the wireless links for our protocol to report termination once the system is terminated (the communication on wired links are considered to be much faster and, thus, is not counted in this metric). "Control message" is to count the number of control messages ($ACK()$ or $WGT()$) sent on wireless links each time when a mobile process turns idle. Note that our cost is 1 since we delegate the weight collector to a static process whereas the costs for the diffusion and weight-throwing schemes are doubled since each control message has to traverse on two wireless links.

A simulator was also developed to calculate the exact the *wireless* bandwidth consumed for the purpose of termination detection. We simulated a distributed system with m base stations, with m ranging from 1 to 10. Each base station owns four mobile hosts, each running one distributed process. As our focus is on the wireless part, no static processes were simulated. One process started the distributed system by injecting m basic messages into the system. On receiving a basic message, a mobile host will become active for A time units, where A is uniformly distributed in $[1, 49]$ with a mean of 25 time units. While active, a process could generate a new basic message every M time units, where M is also a random variable with a certain mean. The destination of a basic message was randomly selected from one of the other mobile processes. The distributed system was run until it naturally terminated or a preset timer of 2,000 expired.

Fig. 3 shows the number of control messages traveling on wireless links by the three protocols. We varied the ratio A/M (among $1/4, 1/2, 1, 2, 3, \dots, 10$) at different system sizes ($m = 2, 5, 8$). Because mobility cannot be handled by the other two protocols, hand-off and disconnection were not simulated. The weight-throwing scheme uses about two times the control messages used by the hybrid scheme. The diffusion-based scheme has similar performance as ours at low A/M ratios, but is getting worse and worse when the A/M ratio gradually increases over 1/2. Both weight-throwing and hybrid protocols are quite insensitive to the A/M ratio.

An important factor that has been ignored in the previous simulation is the extra weight that has to be carried by each basic message in the weight-throwing scheme. We adopted a simple assumption that each $ACK(k)$ costs 4 bytes (for an integer) and each $WCT(x)$ and basic message costs 8 bytes (for a real number). Byte counts are important for systems frequently delivering very short messages. Fig. 4 shows the average number of bytes required by a mobile host per unit time under different conditions (the other protocol overheads, such as packet header and error-checking code, which should be the same for different protocols, are not accounted in the simulation figures). Under this metric, the weight-throwing scheme is always most costly.

We also compare the three protocols by varying the system sizes ($m = 1..10$) at $A/M = 2, 5, 8$. The results are shown in Fig. 5 and Fig. 6. We observe that the curves for the weight-throwing and hybrid schemes are both insensitive to the system size, but the curves for the diffusion-based scheme are slightly going upward as the system size increases. So the weight-throwing and hybrid schemes are more scalable to systems sizes.

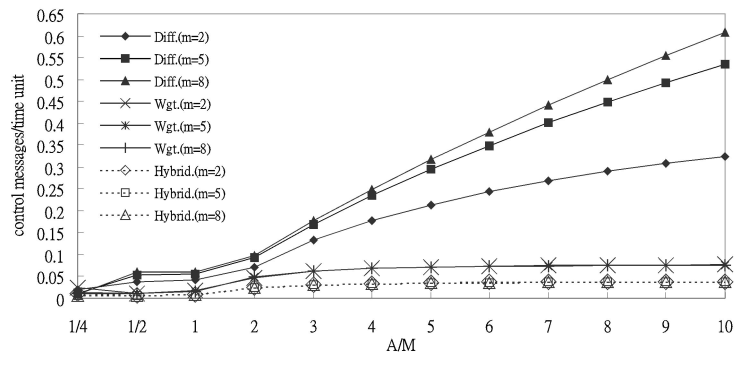

Fig. 3. Number of control messages consumed at different A/M ratios.

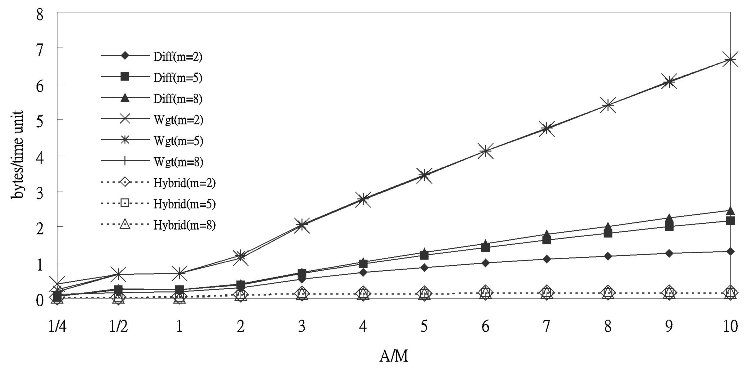

Fig. 4. Number of bytes consumed at different A/M ratios.

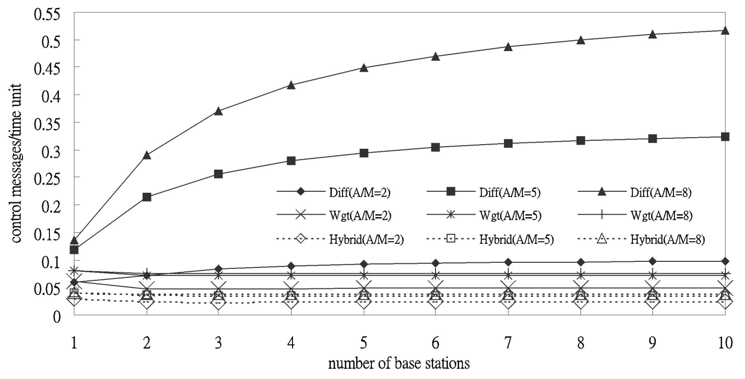

Fig. 5. Number of control messages consumed at different system sizes.

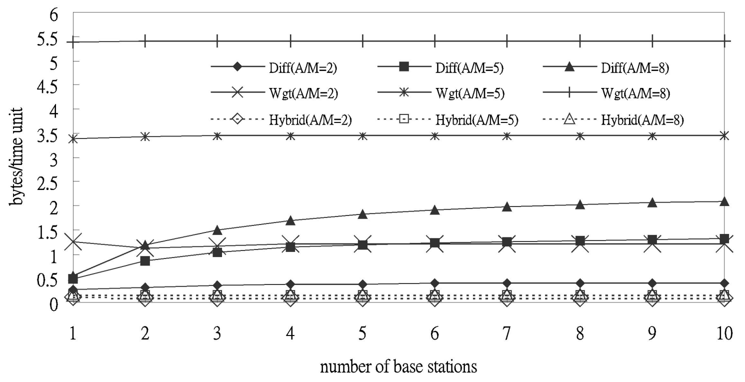

Fig. 6. Number of bytes consumed at different system sizes.

5 Conclusions

With the emergence of wireless networks, many distributed protocols well suited for static hosts have to be reevaluated for their appropriateness on wireless networks. Host mobility and disconnection problems are particularly important issues to be considered. In this paper, we have shown how to handle these problems for the termination detection problem. A newly defined weakly terminated state is provided for users to tell whether a mobile distributed system is still running or is silent because some processes are disconnected. By exploit the network hierarchy, our hybrid termination detection protocol can take advantage of the short detection delays of the weight-throwing scheme on the wired networks and the simplicity and bandwidth-efficient properties of the diffusion-based scheme on each wireless cell. It demands less space, computation, and wireless bandwidth on mobile processes and, thus, is more favorable.

Acknowledgments

This research is supported by the Ministry of Education, ROC, under grant 89-H-FA07-1-4.

References

[1] S. Alagar, R. Rajagopalan, and S. Venkatesan, "Tolerating Mobile Support Station Failures," *Proc. Int'l Conf. Fault-Tolerant Systems,* 1995.

[2] S. Alagar and S. Venkatesan, "Causal Ordering in Distributed Mobile Systems," *IEEE Trans. Computers,* vol. 46, no. 3, pp. 353-361, Mar. 1997.

[3] A. Archarys and B.R. Badrinath, "A Framework for Delivering Multicast Messages in Networks with Mobile Hosts," *ACM/Baltzer J. Mobile Networks and Applications,* vol. 1, no. 2, pp. 199-219, 1996.

[4] S. Chandrasekaran and S. Venkatesan, "A Message-Optimal Algorithms for Distributed Termination Detection," *J. Parallel and Distributed Computing,* vol. 8, pp. 245-252, 1990.

[5] E.W. Dijkstra and C.S. Scholten, "Termination Detection for Diffusing Computations," *Information Processing Letters,* vol. 11, pp. 1-4, 1980.

[6] N. Francez and M. Rodeh, "Achieving Distributed Termination Without Freezing," *IEEE Trans. Software Engr.,* vol. 8, no. 3, pp. 287-292, 1982.

[7] J. Geier, *Wireless Networking Handbook.* Indianapolis: New Riders Publishing, 1996.

[8] A. Hills and D.B. Johnson, "Wireless Data Network Infrastructure at Carnegie Mellon University," *IEEE Personal Comm.,* vol. 3, no. 1, Feb. 1996.

[9] S.-T. Huang, "Detecting Termination of Distributed Computations by External Agents," *Int'l Conf. Distributed Computing Systems,* pp. 79-84, 1989.

[10] S.T. Huang, "A Distributed Deadlock Detection Algorithm for CSP-Like Communication," *ACM Trans. Programming Language and Systems,* vol. 12, no. 1, pp. 102-122, Jan. 1990.

[11] L. Kleinrock, "Nomadic Computing—An Opportunity," *ACM Computer Comm. Review,* pp. 36-40, year???

[12] P. Krishna, N. Vaidya, and D. Pradhan, "Recovery in Distributed Mobile Environments," *Proc. Workshop Advances in Parallel and Distributed Systems,* pp. 83-88, 1993.

[13] T.-H. Lai, Y.-C. Tseng, and X. Dong, "A More Efficient Message-Optimal Algorithm for Distributed Termination Detection," *Int'l Parallel Processing Symp.,* pp. 646-649, 1992.

[14] T.-H. Lai and L.-F. Wu, "An $(n-1)$-Resilient Algorithm for Distributed Termination Detection," *IEEE Trans. Parallel and Distributed Systems,* vol. 6, no. 1, pp. 63-78, Jan. 1995.

[15] B.P. Lester, *The Art of Parallel Programming.* Prentice Hall, 1993.

[16] F. Mattern, "Golbal Quiescence Detection Based on Credit Distribution and Recovery," *Information Processing Letters,* vol. 30, pp. 195-200, 1989.

[17] J. Misra and K.M. Chandy, "Termination Detection of Diffusing Computations in Communicating Sequential Processes," *ACM Trans. Programming Language and Systems,* vol. 4, no. 1, pp. 37-43, Jan. 1982.

[18] R. Prakash, M. Raynal, and M. Singhal, "An Efficient Causal Ordering Algorithm for Mobile Computing Environments," *Int'l Conf. Distributed Computing Systems,* pp. 744-751, 1996.

[19] R. Prakash and M. Singhal, "Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 7, no. 10, pp. 1035-48, Oct. 1996.

[20] M. Singhal and N. Shivaratri, *Advanced Concepts in Operating Systems.* McGraw-Hill, 1994.

[21] Y.-C. Tseng, "Detecting Termination by Weight-Throwing in a Faulty Distributed System," *J. Parallel and Distributed Computing,* vol. 25, pp. 7-15, 1995.

[22] S. Venkatesan, "Reliable Protocols for Distributed Termination Detection," *IEEE Trans. Reliability,* vol. 38, no. 1, pp. 103-110, Apr. 1989.

[23] Y. Sato et al. "A Snapshot Algorithm for Distributed Mobile Systems," *Int'l Conf. Distributed Computing Systems,* pp. 734-743, 1996.

[24] L.-H. Yen, T.-L. Huang, and S.-Y. Hwang, "A Protocol for Casually Ordered Message Delivery in Mobile Computing Systems," *Mobile Networks and Applications,* vol. 2, no. 4, pp. 365-372, 1997.

Yu-Chee Tseng received the BS and MS degrees in computer science from the National Taiwan University and the National Tsing-Hua University in 1985 and 1987, respectively. He worked for the D-LINK Incorporated as an engineer in 1990. He obtained the PhD degree in computer and information science from Ohio State University in January of 1994. From 1994 to 1996, he was an associate professor at the Department of Computer Science, Chung-Hua University. He joined the Department of Computer Science and Information Engineering, National Central University in 1996 and has become a professor since 1999. Since Aug. 2000, he has been a professor at the Department of Computer Science and Information Engineering, National Chiao-Tung University, Taiwan. He served as a program committee member for the International Conference on Parallel and Distributed Systems, 1996, the International Conference on Parallel Processing, 1998, the International Conference on Distributed Computing Systems, 2000, and the International Conference on Computer Communications and Networks 2000. He was a workshop cochair of the National Computer Symposium, 1999. His research interests include wireless communication, network security, parallel and distributed computing, and computer architecture. He is a member of the IEEE Computer Society and the Association for Computing Machinery.

Cheng-Chung Tan received the BS and MS degrees in computer science from Tamkang University and National Central University in 1997 and 1999, respectively. He has served as an engineer at the Institute for Information Industry in Taiwan since 2000. His currently working on embedded operating systems and IA products. His research interests include mobile computing, networking protocol, and operating system.