



A Termination Detection Protocol for Use in Mobile Ad Hoc Networks

GRUIA-CATALIN ROMAN

roman@wustl.edu

JAMIE PAYTON

payton@wustl.edu

*Department of Computer Science and Engineering, Washington University in St. Louis, Campus Box 1045,
One Brookings Drive, St. Louis, MO 63130-4899, USA*

Abstract. As devices become smaller and wireless networking technologies improve, the popularity of mobile computing continues to rise. In today's world, many consider devices such as cell phones, PDAs, and laptops as essential tools. As these and other devices become increasingly independent of the wired infrastructure, new kinds of applications that assume an ad hoc network infrastructure are being deployed. Their development forces software engineers to revisit well understood problems in a setting in which existing solutions are no longer working. This paper illustrates one such attempt by focusing on an important problem in distributed computing, termination detection in diffusing computations, in an ad hoc network environment. We formulate an algorithmic solution amenable to usage in mobile ad hoc networks. Along the way, we highlight several important software engineering concerns one must address and design strategies one might employ in a mobile setting.

Keywords: mobile computing, ad hoc network, termination detection, diffusing computation, algorithm

1. Introduction

In the recent past, mobile computing may have evoked images of a businessperson using a laptop in a conference room, a PDA on a business trip, or a cell phone in a taxicab. Today, mobile computing encompasses a diverse set of devices for a wide range of applications. Retail store employees use mobile devices to check inventory, runners wear devices that compute their heart rate and speed while on the track, and music lovers have MP3 players that can be used to store and play music anytime and anywhere. Increasingly, schools are equipping their students and teachers with mobile devices to enhance the learning experience. Presently, the military is outfitting its soldiers with wearable devices to assist in urban warfare strategy and communication. Automobile manufacturers are including guidance systems to assist travelling motorists.

The strong demand for mobile computing devices in our society warrants the development of new, innovative software that is designed for wireless communication in any setting. We expect some of these applications to be used in situations in which it is not possible to access a wired infrastructure, yet communication among devices is still required. In such settings, reliance on ad hoc networks becomes essential. Ad hoc networks are formed opportunistically among devices equipped with wireless communication capabilities. Links are established as devices move within communication range and are broken as hosts move away. All of this takes place without assistance from any wired resources.

Numerous distributed applications rely on detecting the completion of a computation to ensure correct operation. The same is true for applications developed for use in ad hoc networks. In a field command and control application, for instance, the network infrastructure may be under enemy control. A commander preparing to issue an attack directive will need to rely upon an ad hoc network formed by devices that soldiers carry with their gear. Before issuing the order, the commander may need to request position information from all soldiers in the area to ensure that all units are in their proper positions. Each soldier responds with position coordinates as soon as the request is received. Termination of the entire computation must be detected to ensure that every soldier is in position before the attack directive is issued. Similarly, in a wireless anywhere classroom setting, the instructor and students may form an ad hoc network using their laptops and PDAs. Using this network, a class assignment may be distributed among all students. Students complete the assignment, and then turn in their homework by the same means. All students must submit their assignments before the instructor opens up the discussion of the solution. This again entails the reliance on a termination detection protocol.

The examples above illustrate the importance of termination detection across a diverse range of mobile applications. This paper is concerned with the development of a termination detection algorithm for use in ad hoc network environments. In the remainder of the paper, we focus on detecting termination in diffusing computations. Such computations are characterized as those in which initially a single node is active, and awakens other nodes to perform some computation. These awakened nodes can spread the computation to other nodes, which can spread the computation further, and so on. An important advantage of diffusing computations is that only participating nodes need to carry out the termination protocol. A well-known solution to termination detection in diffusing computations is the algorithm proposed by Dijkstra and Scholten (1980). In this algorithm, a spanning tree of active nodes is constructed by starting with a single active node (the root) that gradually spreads the work to other nodes in the network, awakening idle nodes as work requests are passed along. Upon activation, a node becomes the child of the activating node, causing a transition from idle to active status. The activating node is always part of the tree. If a node that is already in the tree receives a request for activation, it notifies the sender that the tree's topology need not change. A node can be removed from the tree when it is an idle leaf node by notifying its parent which will remove it from the list of active children. Termination is detected when the tree contains only one idle node—the root node. The algorithm relies upon constant connection between the child and its parent node.

In a mobile setting, however, it is likely that a parent will become disconnected before a child can deliver the termination notice. What is needed is a solution that allows for propagation of termination messages to be independent of connectivity to the parent node. Eventual delivery of termination reports to the root node should be the only requirement. In this paper, we show that it possible to provide such an algorithm for termination detection in ad hoc networks. Key to our solution is the use of a partial ordering across all active nodes in the network for delivery of termination notices. Thus, a node's termination notice need not be delivered through the node which activated it. One interesting feature of our algorithm is its modular design; different methods of constructing the partial ordering can be used without making changes to the algorithm itself. Also, delivery can be restricted to

only activated nodes or may allow participation by all willing hosts. Finally, it should be noted that the algorithm can be directly applied in settings that exhibit physical mobility of hosts or logical mobility of agents.

The remainder of the paper is organized as follows. In Section 2, we specify the problem of termination detection in diffusing computations for mobile ad hoc networks. Section 3 includes a discussion of an algorithm for termination detection. The algorithm focuses on peer-to-peer communication. In Section 4, we give an example to provide a more concrete understanding of the algorithm. Section 5 contains a discussion of the results. In Section 6, we present related work. Concluding remarks appear in Section 7.

2. Problem specification

A mobile ad hoc network is formed when a collection of mobile hosts equipped with wireless capabilities become connected without assistance from any wired resources. Connections are established when devices move within communication range. The network topology changes as hosts join or leave the ad hoc network. Connections within mobile ad hoc networks are affected by power limitations of devices, resource availability, physical locations of hosts, etc. Typically, mobile hosts communicate directly in a point-to-point fashion, but protocols for ad hoc routing have been and continue to be developed. In this paper, we need to consider only the case of pair wise connectivity.

Our goal is to examine the issue of termination detection in mobile ad hoc networks. Because the ad hoc network is often composed of small, resource-constrained devices, it is desirable to keep the number of participants in a computation as small as possible. When thinking about a diffusing computation, we simply assume that the computation spreads out as hosts “bump” into each other, i.e., a host passes work to another host as they come together within communication range and establish a communication link. The challenge, of course, will be to let the source of the computation know that all hosts touched by the computation actually terminated, despite the fact that connectivity becomes available in an opportunistic and unpredictable manner.

Making termination detection of diffusing computations possible in this setting is not just an interesting problem, but is one that must be solved in order to produce certain types of applications. For instance, termination detection is important to applications that perform multiphase processing. A practical example is epidemic software updates. In the first phase of processing, the software update is propagated to all other hosts. The goal of this phase is for the host that originated the update to know that all hosts received the update. In the second phase of processing, the originator begins to notify hosts that the old software can be dropped. Another example is encountered when a new encryption key for secure communication is generated and sent to others in the group. The old encryption key is kept until all participants have received the new key. The originator of the new key, then, must know that all others have received the new key so that the old one can be destroyed. The originator can then send a message to indicate that the old key should be used no longer.

Informally, the problem we are trying to solve can be captured as follows. All mobile hosts are initially idle except for a single host, called the source, which initiates the

computation and is charged with ultimately detecting termination. The source node can activate a connected idle host in the mobile ad hoc network by sending a request for a task to be performed. The first such request to be received by a node becomes its activation message. A host that becomes active may, in turn, activate other hosts. Notice that an idle host can become active only when it receives an activation message from an active node in the computation. The active nodes later become idle once their processing is complete. To detect termination, it is necessary to ascertain at the source node that, at some point after the start of the computation, all mobile hosts in the ad hoc network that were once activated are idle again. It should also be noted that, throughout this process, connections are established and dissolved very frequently, which often results in leaving a node completely isolated, or in the formation of ever-changing, disjoint subnets. In the general case, the problem is clearly unsolvable since a disconnected active node could simply depart, never to return. As such, examining what might be reasonable ways to constrain the problem will be an integral part of the proposed solution and a guide for a general design strategy.

Formally, let V be the set containing all nodes representing mobile hosts. V is fixed, i.e., the set of all nodes may be very large but closed. Let v_0 be a distinguished node, representing the source of the diffusing computation. Let E be a set containing all communication links that are up at any instant in time. Clearly E changes over time, as hosts change physical location and connections are dropped or established. The combination of V and E forms a logical connectivity graph, $G(V, E)$. Each connected subgraph of G represents an ad hoc network. Furthermore, let us assume that each node u has an associated Boolean variable called $u.idle$. Another variable called $u.done$ is needed to log the fact that termination was detected. Since detection is carried out only at the source node v_0 , $u.done$ remains false for all other nodes. Given these assumptions, the termination detection problem assumes its classical formulation:

Given

$$W \equiv \langle \forall u : u \in V :: u.idle \rangle$$

stable W

construct a protocol such that

$$v_0.done \text{ detects } W.$$

This formulation is given in the UNITY notation (Chandy and Misra, 1988), which provides precise definitions of stable and detects, and can be explained as follows. We define a predicate W to be true if and only if all nodes in the network are idle. The stability requirement states that, once established as true, W will continue to hold forever. In other words, once all nodes in the network become idle, they will stay that way. Our goal is to build a protocol that can detect termination given this stability requirement. The detection requirement above captures two properties. First, if all nodes become idle, the fact is eventually recorded by setting the termination flag, $v_0.done$. Second, once set, the termination flag $v_0.done$ guarantees that all nodes are idle. The first is a progress property, while the second is a safety property.

3. Algorithmic solution

In this section, we introduce our algorithm for termination detection in diffusing computations over mobile ad hoc networks. For purposes of exposition, we assume that the set of mobile hosts is closed and communication is point-to-point (ad hoc routing is not used). Furthermore, we allow a node to be activated at most once. We will be able to eliminate this last restriction later in the paper. The details of the algorithm are shown in figure 1. In the figure, the symbols “+” and “−” represent set union and difference, respectively.

3.1. Protocol description

The diffusing computation begins with a single node being active, the source of the computation. Henceforth, we refer to this node as the root because it serves as the starting point for the construction of an activation tree that will keep track of all the nodes participating in the computation. Once active, a node may request help from other nodes that are within communication range, still in the idle state, and are known to have work to do (action *IssueActivatingRequest_B* in figure 1). The function *work_to_be_done_by_B* is an abstraction for the decision process by which a node determines whether it has work to be delegated to some other node.

Since (for now) nodes can be activated at most once, the recipient of an activation request must not have been previously activated. This is the reason why the activation action is guarded by a condition, which in practice would only be known by the other party in the communication. The use of the guard simplifies the algorithm presentation, but hides the presence of an additional synchronous message exchange between the two nodes. As a matter of fact, even if we were to remove the restriction, the information of whether or not a work request activated a new node or it involved a node already active would still need to be communicated to the requester since the activating node must keep track of all the nodes it activated throughout its lifetime. Work requests circulating among active nodes (action *IssueAnotherRequest_B*) are simply carried out without affecting the bookkeeping process associated with detecting termination.

When a node is activated, both participants record the fact but in different ways. The activating node places the activated node in its list of children (variable *ActivatedChildren*), while the node being activated transitions from idle to active (action *AcceptWorkRequest*). Implicitly, the newly activated node is added as a leaf to a tree rooted at the source of the diffusing computation. The tree is stored in a distributed manner by having each parent keep track of its own children. Ignoring node termination for the moment, every active node is reachable (in principle) along a path from the root but (in fact), many of the links may no longer be up since nodes may have moved out of range with respect to each other.

An activated node that no longer has a task to perform may terminate (action *NodeTerminates*) at any time by changing its state from active to idle—the record of having been activated already (auxiliary variable *activated* and variable *activatedChildren*) remains unchanged and, because of the technical restriction that a node can be activated at most once, the node is effectively removed from the computation. The transition to the idle state is accompanied by the generation of an idle report, which is stored locally as part of a

State characterization for node A	
$idle$	– Boolean, true if and only if the node is in an idle state, initially true except for the initiator of the diffusing computation
$root$	– true if and only if the node is the initiator of the diffusing computation
$activatedChildren$	– a set of activated nodes, initially empty
$idleReports$	– a set of pairs of the form (idle node, activated nodes)
id	– unique node identifier
$done$	– Boolean, true if the root detected termination, false for all other nodes, initially false
$channel(A, B)$	– Boolean, true if communication link between A and B is up
<u>Auxiliary Variables</u>	
$activated$	$\equiv (activatedChildren \neq \emptyset)$
$never_activated$	$\equiv \neg activated$
$active$	$\equiv \neg idle$
<u>Actions at A</u>	
<u>DetectTermination</u>	
Precondition:	$root \wedge idle \wedge idleReports = \emptyset$
Effect:	$done := true$
<u>IssueActivatingRequest$_B$</u> – A sends an activation message to B	
Precondition:	$active \wedge channel(A, B) \wedge B.never_activated \wedge work_to_be_done_by_B$
Effect:	$activatedChildren := activatedChildren + \{B\}$ $send\ job(task)\ to\ B$
<u>IssueAnotherRequest$_B$</u>	
Precondition:	$active \wedge channel(A, B) \wedge B.activated \wedge B.active \wedge work_to_be_done_by_B$
Effect:	$send\ job(task)\ to\ B$
<u>AcceptWorkRequest</u> – activation message arrives at A from B	
Let the message be $job(task)$	
Effect:	if $never_activated$ then $idle := false$ end $perform_the_task$
<u>NodeTerminates</u> – node A terminates	
Precondition:	$active \wedge no_work_to_do$
Effect:	$idle := true$ $idleReports := idleReports + \{(A, activatedChildren)\}$
<u>PropagateIdleReports</u> – node A meets node B	
Precondition:	$channel(A, B) \wedge (B.id < A.id)$
Effect:	if $idleReports \neq \emptyset$ then $send\ nodeInfo(idleReports)\ to\ B$ $idleReports := \emptyset$ end
<u>AcceptIdleReports</u> – node A receives computation states from B	
Let the message be $nodeInfo(new_reports)$	
Effect:	$idleReports := idleReports + new_reports$
<u>RemoveIdleLeaves</u>	
Precondition:	$((x, z) \in idleReports) \wedge (y \in z) \wedge (y, \emptyset) \in idleReports$
Effect:	$idleReports := idleReports - \{(x, z)\} + \{(x, z - y)\} - \{(y, \emptyset)\}$

Figure 1. Termination detection algorithm for mobile ad hoc networks.

completion history (variable *idleReports*). This may appear at first to have broken the activation tree, but this is not the case. The parent/child relation is captured by the local list of children while the node is active and by the idle report once the node becomes idle. The distinction is important. First, a node that generated an idle report is, in fact, idle. Second, idle reports stored in the local completion histories need not remain with the node that generated them but can travel from one node to another according to a set of rules that maintain the integrity of the tree and, eventually, make it possible for the root to declare the computation as being finished. Idle nodes maintain a virtual presence in the activation tree even though they may be long gone and out of reach. A path from the root to the idle node still exists but the information about the tree structure is scattered among nodes that may or may not have been involved with the computation. Nevertheless, premature termination detection cannot happen because a decision at the root cannot be taken before all the idle reports are collected. If any report is missing there is always a node present at the root that will point to it, directly or indirectly.

If the system enters a state in which all nodes are idle and all idle reports have reached the root, termination detection becomes a trivial exercise. One simply removes each leaf of the tree, representing an idle node with no children that could possibly be still active (action *RemoveIdleLeaves*), and repeats the process until the only node left in the tree is the idle root (action *DetectTermination*). The fact that all nodes eventually terminate is one of the assumptions made in the definition of the problem. But how will the idle reports reach the root? In the static network, connections stay up and idle reports could be funneled to the root along the paths in the tree. In the ad hoc setting, we could make the assumption that every node eventually meets the root and transfers the idle report, but this would be much too strong. In the algorithm as presented, we abstract the notion that an idle report eventually reaches the root by postulating the existence of a partial order over the universe of nodes, having a lowest bound, the root node. Under this assumption, a node carrying some portion of the completion history simply passes all it knows to any node lower with respect to the partial order.

We discussed earlier the method by which the root recursively prunes the activation tree. This process, however, need not wait to be carried out by the root. Any node that has sufficient information should carry out the pruning (action *RemoveIdleLeaves*). If the (incomplete) completion history stored at the node contains a leaf and also the idle report associated with the parent, the information about the leaf may be eliminated in the node's completion history and the parent's idle report without any negative impact on the detection process or on the integrity of the activation tree.

3.2. *Partial ordering of nodes*

We postulated the existence of a partial ordering for message delivery purposes. An obvious question is, how can nodes involved in the computation be arranged in a partial ordering? An obvious answer is to assign numerical identifiers to hosts. One possibility is to statically assign identifiers to nodes based upon the location at the start of the computation. However, hosts move a great deal making static assignment too inflexible. Dynamic assignment of identifiers is another possibility. In thinking about the manner in which a partial order for

message delivery can be dynamically constructed in our algorithm, it is interesting to note that it is possible to capture a rich set of assumptions about the delivery policy:

- (1) If the ad hoc network is self-organizing in some hierarchical fashion, it is possible to direct the data to the lowest common ancestor of a given node and the root node and, once reaching the ancestor, to redirect the information down to the root node.
- (2) If the root has a fixed location, a node that is heading in that particular direction may be viewed as being closer to the root. This suggests that the geometry of the space can play a role in the data transfer policy. Also, as mentioned before, it highlights the fact that the partial order need not be defined in a static manner but in a way that takes into consideration motion patterns.
- (3) If we desire to involve in the delivery process only nodes that participated already in the computation, we need to make available to each activated node some notion of distance relative to the root, e.g., depth in the activation tree. Initially, only the root is active and has a valid identifier value; all other nodes have a pre-defined default value that indicates non-participation in the computation. Identifiers that indicate depth in the activation tree can be assigned as nodes are activated. Using their identifiers, two nodes can easily determine whether they both participated in the computation and their relative ordering relation. It is important to note that with this method of establishing a partial order, only the root node is guaranteed to have a unique identifier. As such, some nodes may have identifiers that are not comparable. In this case, no idle reports would be passed between them. Nodes can, however, exchange information with any “partial order ancestor”, i.e., any node with a lesser identifier. In Section 4, we provide an example demonstrating the use of our algorithm in which this method is adopted for building a partial order.

Although the methods described above for constructing a partial order are adequate for many ad hoc networks, a developer may want to choose other methods of building the partial order to suit the needs of the application. Fortunately, our algorithm is modular in that detecting termination does not rely on knowledge of the method used to construct the partial order. Many strategies that capture the construction and maintenance of some partial ordering over the nodes of the ad hoc network are equally suitable.

Several schemes already exist for destination-oriented message delivery in ad hoc networks, and many are based on the notion of associating numerical values to hosts based on relative distance to a destination host and maintaining an ordering on nodes. In epidemic routing (Vahdat and Becker, 2000), the goal is to deliver a message to a particular host in an ad hoc network in which ad hoc routing is used, relying only on temporary pair wise connections between hosts. In this work, the order is controlled by the last bits in the hosts’ IP addresses. The induced order is used to control message delivery. A more dynamic approach is captured in the Disconnected Transitive Communication (DTC) model (Chen and Murphy, 2001), which uses a value called a *utility* to determine which host in the neighborhood is most desirable for carrying a given message. If a host is closer to the destination, it means that the probability that it will bump into the destination host within a given time frame would be higher. A node sends a utility probe to connected nodes, which calculate their respective utility on demand. Utility responses are collected, and the message is sent to

the node with the highest utility (the node that is most likely to meet the destination node). In effect, a partial ordering is constructed using the inverse of the utility value such that the destination node is the least element. The transient nature of the ordering offers only statistical delivery guarantees. In other work, algorithms for ad hoc routing (Gafni and Bertsekas, 1981; Park and Corson, 1997), token based mutual exclusion (Dhamdhere and Kulkarni, 1994; Walter et al., 2001a, 2001b), and leader election (Malpani et al., 2000) capture their own routing strategies based on the construction and maintenance of a total ordering. In these algorithms, the total ordering of host identifiers reflects the relative location of hosts with respect to the destination.

3.3. Proof sketch

In order to prove that termination is detected, we must make certain assumptions. First, we assume the establishment of a partial ordering among node identifiers, with a unique least element. Second, we assume that once an ordering is established, the distance of an idle report from the root (in terms of the partial ordering) cannot increase. Third, as stated before, we assume that all activated nodes terminate and generate an idle report. Finally, we must assume that a node carrying idle reports eventually meets another node that is logically closer to the root. Without this assumption, no solution exists, since nodes may go away never to come back. The information they hold would be lost, and no final determination of the termination status would be possible. This is one of the realities of mobile ad hoc networks.

Relying on these assumptions, we can prove that once all nodes have terminated, our algorithm detects termination. To do so, we must first show that idle reports move closer to the root with each transfer. This means that we should have a strictly decreasing variant function describing the position of idle reports. Second, we must show that the activation tree always remains valid. Finally, we must show that once the root receives all idle reports, the activation tree can be pruned in a finite number steps to contain a single node. We begin a proof sketch below by defining necessary terms, giving the variant function, and capturing the integrity of the algorithm in an invariant.

When all nodes have terminated, each idle report r_i is at some distance d_i from the root. Distance is defined as the number of unique node identifiers in the partial order between the holder of the report and the root node. To show that idle reports reach the root node in a finite number of steps, a variant function can be defined as the sum of distances over all idle reports, i.e., $\sum_{i \in T} d_i$, where T is the activation tree. To help show that the activation tree is always valid, we capture the integrity in an invariant, which states that from the set of all idle reports, one can construct the activation tree T .

In carrying out the proof, we must consider the entire range of possible state transitions that could affect the integrity of the activation tree and variant functions.

- (1) Nodes move, possibly causing existing communication links to break or new ones to form.
- (2) Nodes meet, and idle reports may be transferred and coalesced.
- (3) A node may prune the activation tree.

We must ensure that our algorithm still detects termination, even in the face of the above situations. We use the definitions, variant function, and invariant given above to do so, addressing each situation in turn.

Movement. Since we assume that the distance d_i of an idle report r_i does not increase due to node mobility, the variant function is not increased due to mobility. The activation tree T is still valid since it is constructed from idle reports, which remain unchanged.

Pruning. An idle report r_i is not transferred during pruning, nor are node identifiers altered. However, since an idle report can be removed, the distance d_i for that report becomes zero. Therefore, the sum of all distances will decrease, thus preserving the validity of the variant function. The pruning mechanism removes a node from the activation tree only when it is an idle leaf, i.e., a node in the activation tree that is idle and has no children. In the pruning action, the connection between the idle leaf and its parent is eliminated, as well as the idle leaf entry itself. This is accomplished by removing the leaf from the parent's list of children in the parent's entry in the idle report, and by removing the idle leaf's entry in the idle report. This results in an activation tree with one less leaf node, with no dangling references. The activation tree is still valid.

Information Transfer. When an idle report is transferred, the node with the lower identifier updates its idle report to also contain the idle report of the transferring node, by simply taking the union of two reports. In the meantime, the transferring node erases the contents of its idle report. The invariant states that the activation tree is made up of all idle reports. It is easy to see that the activation tree is the same after the transfer as it was before (since no pruning has taken place between the start and end of the transfer). We can conclude, then, that the transfer of an idle report does not affect the validity of the activation tree. Since an idle report can only be transferred upon meeting a node with a lower identifier, the distance of a transferred idle report r_i must be less than its previous distance, i.e., $d'_i < d_i$, where d'_i is the distance after the transfer of report r_i . Thus, for every idle report transfer, the variant function $\sum_{i \in T} d_i$ is guaranteed not to increase; in fact, it is easy to see it will decrease with each idle report transfer.

The sum of distances does not increase in any of the situations described above. To ensure that termination is eventually detected at the root, we must ask ourselves if the variant function actually decreases with each transfer and if the variant function is bounded. Information transfer is the only situation in which the sum of distances changes, and the value can only decrease. Transfer of idle reports occurs only to nodes with lower identifiers, so the root never transfers idle reports. Coupled with the assumption that a (non-root) node will eventually meet another node with a lower identifier, we can prove that all idle reports reach the root. Therefore, for every idle report r_i , there is a lower bound of zero on the distance d_i , and so the variant function $\sum_{i \in T} d_i$ is bounded at zero as well.

Finally, we must prove that the termination is detected, i.e., the only node remaining in the activation tree is the root. Once the root has received all idle reports (i.e., $\sum_{i \in T} d_i = 0$), it begins pruning the activation tree. We have shown above that pruning decreases the distance d_i of an idle report r_i , so the variant function is valid. We have already shown that each pruning results in a valid activation tree. Now we must show that the activation

tree is pruned to a single node (the root), given that all idle reports are present at the root. Each time the pruning method is called, a single node is pruned from the activation tree. Thus, the number of nodes in the activation tree decreases by one each time the pruning function is called. This action is repeated until eventually only a single node, the root, remains in the activation tree. At this point, the termination flag is set and termination is detected.

We assumed a particular scheme for assigning identifiers in the proof outlined above, one in which the distance (with respect to the partially ordered identifiers) between an idle report and the root node cannot increase. There are other ways of shaping delivery paths to the root, e.g., in a scheme that utilizes motion patterns, idle reports are transferred when encountering a node with a velocity vector with a direction component pointed toward the location of the root node. Note that the above proof does not accommodate this. To prove that all idle reports reach the root and termination is detected with a motion-sensitive method of maintaining a partial order, a different variant function is needed to show that the idle reports are guaranteed to reach the root node. The variant function would be the sum of physical distances from all nodes to the root node. However, it is possible to encounter scenarios in which it is very difficult to prove that the variant function always decreases. For instance, a node moving away from the root meets a node that is physically closer, and it passes its idle report. The latter receives the idle report and merges it with its own, only to change direction to begin moving further and further away from the root. In such scenarios, proving that the distance of idle reports using the variant function suggested is impossible without making certain assumptions about the behavior of the nodes. For instance, one solution may constrain the movement patterns of nodes by not allowing nodes that hold idle reports to increase their physical distance from the root.

4. Example

In order to provide a more concrete understanding of the algorithm's mechanics, let us consider the example of an epidemic software update. In the scenario that we present, updating software in this manner is instrumental in minimizing downtime in factory production. A factory relies on robots to complete quality assurance tasks. The robots move within the factory floor space, using information gathered from sensors at various points in the factory to complete each task. The space is large, and there are many robots.

Robots may be assigned to collect information from only a subset of the sensors. The subset of sensors with which a robot communicates is determined by the robot's job function. Figure 2 illustrates a possible interaction between robots and sensors on the factory floor. The rectangles represent various pieces of machinery in the factory. The robots are represented by shaded concentric circles, and sensors are represented by plain circles. Sensors are either located on pieces of machinery or on a wall of the factory. Communication between robots is represented by solid lines and communication between robots and sensors is represented by dashed lines.

Robots share a common communication protocol, PV1.0. As robots move within range of other robots or sensors, they are able to communicate various pieces of information using

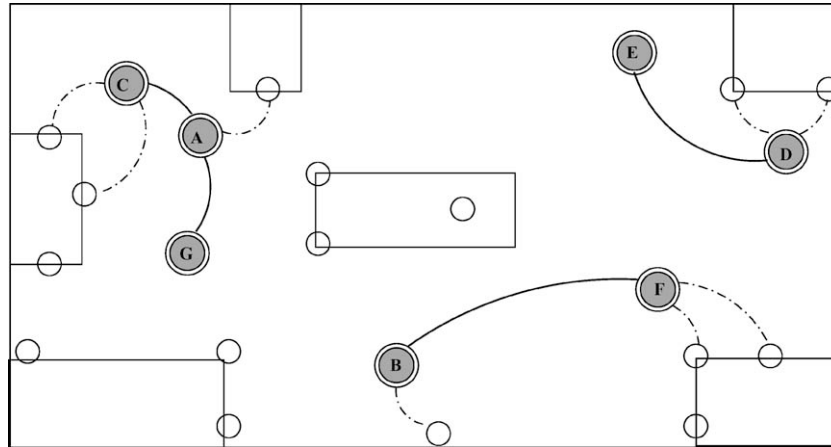


Figure 2. Communicating robots and sensors on a factory floor.

PV1.0. At some point it is determined that an upgrade from PV1.0 to PV2.0 is needed. The factory cannot operate without the robots, yet it is not reasonable to shut down factory production in order to upgrade the software on all of the robots. To update the software without factory downtime, an epidemic software update will be used, in which the update starts at one robot and spreads to all others.

Both robots and sensors must be upgraded to support the new software, PV2.0, while continuing to support the previous version. For presentation purposes, sensor updating is ignored. The epidemic software update is used only to update robot software. Actually, two diffusing computations will be used: one to distribute the new software, and another to remove the previous version of the software. The initial diffusing computation is started by installing the new software, PV2.0, on a single robot, which will be assigned a unique lowest identifier with respect to the software upgrade task.

As the robot moves around the factory floor, it moves within communication range of another robot and passes along the software upgrade. As the computation is spread, a newly activated robot will be associated with a robot identifier that indicates its position in the activation tree, such that the identifiers form a partial ordering with a unique least element (at the root). This process of spreading the computation and assigning identifiers is repeated by the receiving robot, and so on. Since we assume that each robot carries a list of the names of other robots with which it customarily cooperates, each robot will consider itself to have completed its update activity when all its acquaintances have been encountered. The process of passing updates will be complete when all robots have received the software upgrade. At this point, the robots must support both PV1.0 and PV2.0. The robot that originated the request must receive status information about all robots before completing the update such that the previous protocol can be dropped. Note that in our algorithm, robots do not have to contact the originator of the software upgrade. Rather, responses are passed along to other robots that have lower identifiers in the partial ordering of robots.

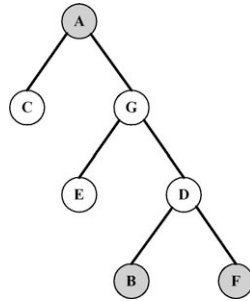


Figure 3. Snapshot of the logical activation tree.

Because responses are passed to robots with lower identifiers and the originator has the lowest identifier, eventually, the responses reach the robot that originated the update. When the originator robot has collected responses concerning all robots involved in the upgrade, termination of the first diffusing computation is detected. The same robot can now issue a request for the other robots to complete the software upgrade, starting the second diffusing computation. The request tells the other robots to drop PV1.0. Eventually, the originator robot should receive confirmation that these requests were received and processed, and the second diffusing computation is terminated.

An activation tree our algorithm for termination detection might produce in completing the software update at time t is shown in figure 3. Active nodes representing robots in the activation tree are shaded. Non-shaded nodes represent robots that were once active, but are currently idle. While a snapshot of the activation tree gives a history of which robots have been connected in the past, it does not necessarily reflect the current physical connectivity between nodes. To illustrate this point, a corresponding snapshot of a possible ad hoc network topology formed between communicating robots at time t is shown in figure 4. Due to the movement of the robots on the factory floor, connections are established and dropped as time elapses. Thus, the network topology may become reconfigured at any point in time.

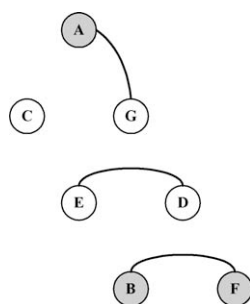


Figure 4. Corresponding snapshot of the physical connectivity.

Initially, all robots have default identifiers of a predefined “infinity value,” which indicates non-participation in the computation. Robot A is the originator of the computation, and becomes the root of the activation tree (figure 3). As such, it is assigned a numerical robot identifier of one. Robot A passes the software upgrade request to robot G. A robot identifier value for robot G is piggybacked on the upgrade request, such that the value is $root.id + 1$, or two. If robot G has not been previously activated, then it records this robot identifier in its id variable. Robot A moves to another position on the factory floor, becomes connected with robot C, and passes the software upgrade request, with a robot identifier of two piggybacked on the request. The names of robots C and G are added to robot A’s list of activated children, and so the associated nodes for robots C and G are added to the activation tree (figure 3). Robot G activates two more robots, robot D and robot E, and passes them robot identifiers of $G.id + 1$, or three. Nodes are accordingly added to robot G’s list of activated children and, thus, to the activation tree. At some point, robot C finishes its work and becomes idle (figure 3), adding its activation history to its idle report. Robot D activates robot F and robot B, which are given robot identifiers of $D.id + 1$ (i.e., four) and are added to the activation tree. In the meantime, robot G terminates (figure 3). Notice that robot G is the parent of robot D in the activation tree, and is terminated, but not removed from the tree. Later, robot E terminates.

After a robot becomes idle, it will pass its idle report when it meets a node with a lower identifier. Figure 5 illustrates how passing and storing idle reports affects a robot’s local view of the activation tree, from the perspective of robot A. In the figure, a bold edge between nodes indicates that robot A knows the relationship between the connected nodes. Grayed edges indicate that robot A has no knowledge of the relationship between the connected nodes. Heavy outlining around a node indicates that A knows the status of the node, i.e., whether it is active or idle.

From the figure, we see that robot A knows the following from its list of activated children and idle reports:

- (1) Robot A knows that it is the parent of robots C and G because they are in its list of activated children.
- (2) At some point after becoming idle, robot C established a connection with robot A, and passed its idle report. Robot A now knows that C is idle and has no children.

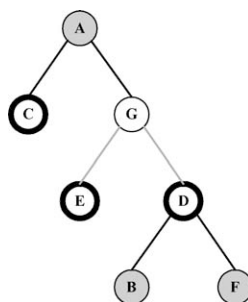


Figure 5. Node A’s knowledge about the status of the computation.

- (3) At some point after becoming idle, robot E moves into communication range of robot D and passed its idle report. Additionally, robot D later becomes idle and adds its information to the idle report. Robot D eventually establishes communication with robot A and passes its idle report. Robot A now knows that D and E are idle, that E has no children, and that D is the parent of robots B and F.

Note that robot A does not know that G is the parent of robots D and E. Also, robot A does not know the status of robots G, B, and F. At this point, A still cannot prune robots C, D, or E from the activation tree because none of these are listed as a child of some robot in A's completion history. While C and D are in A's list of activated children, they are not listed as children in A's completion history because A has not yet terminated. Notice, however, that if robot G were to become idle and its idle report reached robot A, E would be listed as a child in A's completion history. Since robot E has no children, it is an idle leaf and could be pruned from the activation tree.

5. Discussion

In this section, we revisit some of the assumptions made earlier and discuss some subtle aspects of the solution. One assumption was that a host may be activated at most once. This was only to help us explain the essence of the algorithm. A simple modification to the code offers an easy way out. If each node keeps an activation counter, we can treat each new activation as if it involves a completely new node whose identity is defined by a pair consisting of the node identifier and the activation counter. The space taken by the activation counter may be reclaimed, if so needed, when the termination is known by the participant, not just at the root.

Throughout, we assumed that hosts communicate in a point to point fashion. However, it is likely that in the future, most hosts will utilize ad hoc routing. When ad hoc routing is introduced, clusters of connected nodes will form and dissipate opportunistically as the hosts enter and leave the communication range. In this setting, a cluster can divide into multiple clusters and/or merge with another cluster. How can our algorithm be modified to accommodate these changes? Clusters can be thought of as containing all active nodes. Clusters shrink as active nodes become idle, and grow as nodes become active. Each cluster has a "leader." One of the leaders is the originator of the diffusing computation. Each leader would be responsible for communicating with other clusters, and updating and passing on the information needed to maintain the activation tree. An idle node can freely depart knowing that the leader will have all the data required to carry out the algorithm. The leader is also responsible for negotiating cluster merging and partitioning. If a computation spans several clusters, each leader must detect termination in its cluster before the originator of the computation can detect termination. This leader node is similar to a clusterhead (Baker and Ephremides, 1981; Steenstrup, 2001; Zavgren, 1997) or a cluster representative used in ad hoc routing protocols.

An interesting feature of our algorithm is that it can also be applied to diffusing computation applications that are built with mobile agents. No modification to the termination detection algorithm is necessary for it to be useful in the presence of logical mobility.

Instead, the problem specification is altered to describe the underlying model of computation. In our computational model for logical mobility, an agent communicates only with other connected agents. Agents are considered connected when they reside on the same host. To communicate with an agent located on a remote host, one of the agents must migrate so that both agents will be located on the same host. Logical connectivity changes as agents are created, migrated, and destroyed. We can describe a diffusing computation in this setting as follows. A single agent is designated as the source of the diffusing computation, and is responsible for detecting termination. The source agent spreads the computation by creating new agents, which in turn can further spread the computation by creating more agents. The set of all agents participating in the diffusing computation are represented as nodes in the activation tree. Agents are assigned agent identifiers, such that they form a partial ordering with the root having a unique least agent identifier (the source). When agents finish their tasks, they die. The source detects termination when it is the only live agent in the activation tree. Using this problem description, it is possible to directly apply the termination detection algorithm as presented in this paper.

6. Related work

Distributed termination detection has been extensively studied (Chandrasekharan and Venkatesan, 1990; Dijkstra and Scholten, 1980; Filali et al., 2000; Francez, 1980; Lai and Wu, 1995; Mattern, 1987; Misra, 1983; Matocha and Camp, 1998). One approach to the problem is to record snapshots of the global state at several points in time in order to find a point at which the termination condition is met (Chandy and Lamport, 1985; Sato et al., 1996). Another approach to detecting termination is to count the number of control messages sent and received by a node in the computation, such as in a credit/recovery scheme (Mattern, 1989). Yet another approach is to build an activation tree, keeping track of the relationships formed between an activating node and an activated node in a diffusing computation. This is the approach taken in Dijkstra and Scholten (1980), and is the approach employed in our algorithm.

While the aforementioned solutions for termination detection are applicable to distributed systems with a static network topology, they are not directly applicable when ad hoc mobility is introduced. In Tseng and Tan (2001), the authors adapt existing solutions to traditional distributed termination detection for a mobile environment. In this work, a diffusion-based scheme is used for mobile hosts, and a weight-throwing scheme is used for hosts on the wired network. In their diffusion-based scheme, the parent of each mobile host is a base station on the wired network. Like the algorithm in Dijkstra and Scholten (1980), a parent must be contacted when its child becomes idle. In the weight-throwing scheme, a host on the wired network calculates a weight that combines information about the idleness of the host and the emptiness of the host's message channel. A central component collects weight values that are reported by processes in the computation as they become idle. The collector uses the amassed weight values to detect termination. While this solution uses existing termination detection algorithms for a form of mobile computing, it does not address the challenges

associated with a mobile ad hoc network. Furthermore, it does not address termination detection in applications that exhibit logical mobility.

In Queinnec et al. (2001), the authors present an algorithm for termination detection of diffusing computations in the presence of logical mobility, similar to the solution presented in this paper. Both use local activation history data structures to amass information in order to build an activation tree that reflects the computation, and rely on the collection of this local information to detect termination. While these algorithms share similarities, there are also several key differences. One difference is in the underlying computational model for logical mobility. We assume that an agent may only communicate with other agents that reside on the same host, while they assume that an agent can communicate directly with any other agent on a remote host, as long as the hosts are connected. We believe that this is an important distinction. The former seems to provide a better representation of logical mobility, since in practical usage a mobile agent is often migrated to a location to obtain information. Furthermore, the algorithm in Queinnec et al. (2001) assumes that agents pass their termination messages directly to the root of the computation, an assumption that we earlier characterized as being too strong in the presence of frequent disconnection among agents or hosts.

7. Conclusions

In this paper, we have examined the problem of detecting termination in diffusing computations in ad hoc networks. While this problem is well-researched, it has been studied mostly in a distributed computing context. When studying termination detection in a mobile ad hoc network environment, we found that frequent disconnections among hosts render the standard solution ineffective and that no strategy exists for this setting. In turn, we devised a new algorithm to achieve termination detection in the presence of physical or logical mobility, one that does not rely on persistent connections among hosts to achieve the termination condition. In investigating other algorithms for ad hoc networks and while developing our own, we found that a common solution to handling message delivery in the face of unpredictable communication is to rely on goal-directed routing, which is emerging as an important problem in mobile computing. With this knowledge, we constructed an algorithm that is independent of the manner of construction and maintenance of a logical network delivery structure, a partial order. Providing such a modular algorithm allows the application developer to select the partial order construction and maintenance scheme that is best suited for the application. Finally, we have hinted at a formal, yet pragmatic way to think about problems in the mobile computing domain by relying in our correctness arguments on informal application of well-established proof techniques.

Acknowledgments

This research was supported in part by the National Science Foundation under Grant No. CCR-9970939 and the Office of Naval Research under ONR MURI research contract N00014-02-1-0715. Any opinions, findings, and conclusions or recommendations

expressed in this paper are those of the authors and do not necessarily reflect the views of the sponsoring agencies.

References

- Baker, D. and Ephremides, A. 1981. A distributed algorithm for organizing mobile radio telecommunication networks. In *Proceedings of the Second International Conference on Distributed Computer Systems*, pp. 476–483.
- Chandrasekharan, S. and Venkatesan, S. 1990. A message-optimal algorithm for distributed termination detection. *Journal of Parallel and Distributed Computing*, 8:245–252.
- Chandy, K. and Lamport, L. 1985. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 8(3):326–343.
- Chandy, K. and Misra, J. 1988. *Parallel Program Design: A Foundation*. Addison-Wesley.
- Chen, X. and Murphy, A. 2001. Enabling disconnected transitive communication in mobile ad hoc networks. In *Proceedings of the Workshop on Principles of Mobile Computing*, pp. 21–27.
- Dhamdhere, D. and Kulkarni, S. 1994. A token based k-resilient mutual exclusion algorithm for distributed systems. *Information Processing Letters*, 50:151–157.
- Dijkstra, E. and Scholten, B. 1980. Termination detection for diffusing computations. *Information Processing Letters*, 11(1):1–4.
- Filali, M., Mauran, P., Padiou, G., Philippe, Q., and Thirioux, X. 2000. Refinement based validation of an algorithm for detecting distributed termination. In *Proceedings of the International Workshop on Formal Methods for Parallel Programming: Theory and Applications*, volume 1800 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 1027–1036.
- Francez, N. 1980. Distributed termination. *ACM Transactions on Programming Languages and Systems*, 2(1):42–55.
- Gafni, E. and Bertsekas, D. 1981. Distributed algorithms for generating loop-free routes in networks with frequently changing topology. *IEEE Transactions on Communications*, 29(1):11–18.
- Lai, T. and Wu, L. 1995. An (N-1)-resilient algorithm for distributed termination detection. *IEEE Transactions on Parallel and Distributed Systems*, 6(1):63–78.
- Malpani, N., Welch, J., and Vaidya, N. 2000. Leader election algorithms for mobile ad hoc networks. In *Proceedings of the Fourth International Workshop on Distributed Algorithms and Methods for Mobile Computing and Communications*, pp. 96–103.
- Matocha, J. and Camp, T. 1998. A taxonomy of distributed termination detection algorithms. *Journal of Systems and Software*, 43(3):207–221.
- Mattern, F. 1987. Algorithms for distributed termination detection. *Distributed Computing*, 2:161–175.
- Mattern, F. 1989. Global quiescence detection based on credit distribution and recovery. *Information Processing Letters*, 30(4):195–200.
- Misra, J. 1983. Detecting termination of distributed computations using markers. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*, pp. 290–294.
- Park, V. and Corson, M. 1997. A highly adaptive routing algorithm for mobile wireless networks. In *Proceedings of IEEE INFOCOM*.
- Queinnec, P., Filali, M., Mauran, P., and Padiou, G. 2001. Describing mobile computations with path vectors. In *Proceedings of the Fourth International Conference on Principles of Distributed Systems*, pp. 221–234.
- Sato, Y., Inoue, M., Masuzawa, T., and Fujiwara, H. 1996. A snapshot algorithm for distributed mobile systems. In *Proceedings of the 16th International Conference on Distributed Computing Systems*, pp. 734–743.
- Steenstrup, M. 2001. *Ad Hoc Networking*, chapter Cluster-Based Networks. Addison-Wesley, pp. 75–138.
- Tseng, Y. and Tan, C. 2001. Termination detection protocols for mobile distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 12(6):558–566.
- Vahdat, A. and Becker, D. 2000. Epidemic routing for partially-connected ad hoc networks. Technical Report CS-2000-06, Duke University.

- Walter, J., Cao, G., and Mohanty, M. 2001a. A k-mutual exclusion algorithm for wireless ad hoc networks. In *Proceedings of the Workshop on Principles of Mobile Computing*, pp. 29–39.
- Walter, J., Welch, J., and Vaidya, N. 2001b. A mutual exclusion algorithm for ad hoc mobile networks. *Wireless Networks*, 7(6):585–600.
- Zavgren, J. 1997. NTDR mobility management protocols and procedures. In *Proceedings of the IEEE Military Communications Conference*.