Detecting Termination in Pervasive Sensor Networks

Habel Kurian, Abhishek Rakshit and Gurdip Singh
234 Nichols Hall, Computing and Information Sciences
Kansas State University
Manhattan, KS 66506

Abstract—With the increased deployment of pervasive systems, there has been an explosive growth in the number of applications being developed for such systems. Distributed applications typically depend on the underlying middleware infrastructure to provide services to perform their tasks. Many applications rely on a service which can detect the termination of a distributed activity being performed by a set of entities. Existing algorithms for termination detection are based on the layering paradigm wherein the algorithm can monitor application level communication. Pervasive applications, however, may not be structured as strictly layered systems. This paper proposes algorithms for termination detection of distributed applications in pervasive systems. We propose two algorithms for this problem, and show that each performs better than the other under certain conditions. Subsequently, we propose an hybrid algorithm which combines the features of the two algorithms and provides performance comparable to the better of the two algorithms under different conditions.

I. INTRODUCTION

Advances in communication and computing technologies are enabling the development and deployment of deeply embedded, pervasive systems. This has led to an explosive growth in the number of applications for sensor-network based pervasive systems. Numerous distributed pervasive applications are being developed in many different domains such as health monitoring, bio-surveillance, manufacturing automation and defense. For example, a military application may involve a group of soldiers using a combination of wearable and ground sensors and devices to accomplish a surveillance task. Pervasive systems are typically heterogeneous in nature, employing computing entities with varying capabilities and using a number of different communication technologies. Developing middleware algorithms to implement services to support such systems is a challenging task.

Many distributed applications rely on a service which can detect the completion of a distributed task being performed by a set of application entities. For example, consider a scenario where a group of humans and robots are collaborating for performing some task. The entities in the group may interact by exchanging messages or directly observing each other. An entity may become passive on its own (that is, it can determine when it is done with an assigned sub-task), but may become active again after interacting with another active entity. Termination of the application in this case would imply that the overall task has been accomplished and no further application actions are required from any of the entities. As another example, consider a network of wireless sensing modules deployed in an area, where each module is capable of monitoring an attribute such as temperature or motion. An operator may be interested in using this network to detect the termination of a physical phenomena. For instance, a fire can be viewed as an application activity which can spread from the sensing neighborhood of one module to its neighbors (a module reporting fire in its neighborhood is considered as active). In this case, a termination detection algorithm can be used to detect whether the fire has been extinguished.

The problem of termination detection has been extensively studied and many algorithms have been proposed to solve this problem [3], [7], [4], [8], [1]. One class of algorithms is based on detecting termination of diffusing computations. In a diffusing computation, there is a single active node initially which can wake up other nodes to perform computation. The computation can then be spread to other nodes in the network. [3] presented an algorithm to detect termination of such diffusing computations. This algorithm dynamically constructs an activity tree of active nodes, rooted at the initiator node, such that a node $v_1$ is the parent of $v_2$ if $v_1$ initiated computation at $v_2$. A passive leaf node can remove itself from the tree by notifying its parent. A non-leaf node can remove itself from the tree only after all of its children have removed themselves from the tree. Another class of termination detection algorithms is based on a central node polling other nodes for determine their current state [7], [4]. In these algorithms, each node responds to the central node with its state. The central node must coordinate the collection of the states to ensure that application messages in transit are accounted for when determining termination.

The existing algorithms discussed above for traditional distributed systems assume a layered design wherein application communication can be monitored by the algorithm. That is, given an application $App = A_1, \ldots, A_n$, and an algorithm $P = P_1, \ldots, P_n$, each application process $A_i$ is layered on top of $P_i$, and the algorithm entities can monitor application level communication. For example, when $A_i$ sends a message to $A_j$, processes $P_i$ and $P_j$ are able to observe this application message and update any data structures maintained by the algorithm. In general, this capability allows algorithms to intercept and piggyback information on application messages and has been used in many algorithms [5], [2]. Applications in pervasive systems, on the other hand, may not follow the traditional layered model. For example, the application entities and the underlying infrastructure platforms (on which the algorithm executes) may use different forms of communication. In the application discussed above which consists of
physical entities such as humans and autonomous robots, it may be the case that the application communication (which for instance may be via direct visual observation) cannot be directly observed or intercepted by the infrastructure. Thus, existing algorithms cannot be directly applied to address the problem of termination in pervasive systems.

In this paper, we present algorithms for termination detection in pervasive systems. In our model, we assume that there exists a single consumer node which is interested in determining when an application activity has terminated. Our first termination detection algorithm for pervasive systems, \textit{TDAP1}, is an adaptation of the diffusing computation algorithm. In \textit{TDAP1}, each node reports its state directly to the consumer node. In addition to reporting its state, a node also reports the set of currently active neighbors to the consumer node. This enables the consumer node to keep track of the active nodes in the computation. The second algorithm, \textit{TDAP2}, is based on the consumer node polling all nodes in the network regarding their current states. We present a comparative evaluation of the two algorithms using discrete event simulation. We show that each of the algorithms performs better than the other under certain conditions. For example, in \textit{TDAP1}, only those nodes which are involved in the application computation participate in termination detection, whereas in \textit{TDAP2}, all nodes in the network participate irrespective of their involvement in the application. Hence, \textit{TDAP1} requires fewer messages when the application is local to a region in the network. However, the number of messages in \textit{TDAP1} is proportional to the number of times a node changes state from active to passive (as it reports the state changes to the consumer node). Our simulation results show an increase in the number of messages sent in \textit{TDAP1} as the number of state changes increase, whereas we find that \textit{TDAP2} is less sensitive to this factor. Finally, we present \textit{TDAP3} which is a combination of \textit{TDAP1} and \textit{TDAP2}. In this algorithm, we divide the network into sub-groups and use \textit{TDAP2} to detect termination in each sub-group. At the higher level, we view each sub-group as a node in the super-graph and use \textit{TDAP1} to detect termination in the super-graph. This ensures that \textit{TDAP2} is executed only in those sub-groups which are involved in the application computation. We show that \textit{TDAP3}’s performance is comparable to the performance of the better of the two algorithms for various network scenarios.

### II. Problem Definition

In the problem of termination detection, we are given an application \textit{App} with entities \(A_1, A_2, \ldots, A_m\), whose termination has to be detected. Each entity can perform a set of possible actions which may include local actions as well as interactions with others. An entity can be in an active or passive state with respect to termination. An active entity can perform application actions and can change its state to passive at any time. A passive entity, on the other hand, does not perform any application actions as long as it remains in that state (that is, it does not initiate any interactions on its own but may participate in interactions initiated by other active entities). A passive entity can change its state to active after interacting with (or observing) an active entity. The problem is to design a termination detection (TD) algorithm to identify when the application has reached a state in which no application entity will ever become active again. The TD algorithm is executed on a set of infrastructure entities which may consist of wireless sensing modules, gateways and other computing devices. One can view the TD algorithm as a set of processes \(P_1, \ldots, P_n\) where \(P_i\) executes on node \(i\) of the infrastructure.

TD algorithms for traditional distributed systems have been based on the layering approach. As shown in Figure 1, in this approach, each application process \(A_i\) is layered on top of a protocol process \(P_i\). This allows each message sent by \(A_i\) to its peer \(A_j\) to be monitored by both \(P_i\) and \(P_j\). For instance, \(P_i\) may piggyback data on this message or \(P_j\) may delay its delivery. Many distributed algorithms rely on this type of control to monitor and enforce synchronization. In existing TD algorithms, layered structuring has been used to monitor channel states for any messages that might activate a process. Once \(P_i\) learns that \(A_i\) is passive, it can monitor the messages sent to \(A_i\). As long as no messages are received by \(A_i\), \(P_i\) knows that \(A_i\) will remain passive. Another approach that has been used is for \(P_i\) to count the number of messages sent and received along each outgoing and incoming channel respectively. By comparing these counters, the TD algorithm can determine whether or not a channel is empty.

In pervasive systems, applications may not be structured as strictly layered systems and the nature of the application entities may be different from those in traditional distributed systems. For example, as shown in Figure 2, consider a system of entities (e.g., robots) tasked with a goal, in which each entity may perform a set of actions on its own or on observing another entity perform an action. For this application, we may want to use a TD algorithm to determine when the system has reached a quiescent state. In this case, the TD algorithm may be able to use sensing devices to determine whether an application entity is active or passive. However, since application entities interact via direct observation, the TD algorithm will be unable to monitor this communication. As another example, consider a set of infrastructure entities which...
are capable of detecting an attribute such as temperature or motion. In this case, we may want to use the TD algorithm to detect the termination of a physical phenomena such as a fire. In this case again, while the protocol can sense whether the phenomena is active or passive in its neighborhood, it cannot directly observe the spreading of the fire. In general, as shown in Figure 2, the application and protocol entities may interact independent of one another, and protocol entities may not be able to monitor application level communication in pervasive applications.

For example, in Figure 3, node 4 and 8 are initially active. Subsequently, 5 and 9 become active and 4 becomes passive. In this case, either 4 or 8 could have caused 5 and 9 to become active (for example, 5 may have observed that 4 is active and before 5 changes its state to active, 4 could have become passive). Figure 3(c) shows a later state in which 6 and 10 are also active. Let $e.time$ denote the real-time at which $e$ occurs. In the following, we list a set of assumptions used in our algorithms (the correctness of each proposed algorithm depends on a subset of these assumptions):

- **Assumption A1**: There exists an upper bound $Propagate\_Max$ such that the following is true. In every execution of the application, for each event $active_i^t$ which could have been caused by another active event, there exists $active_j^y$ such that $active_j^y \rightarrow active_i^t$ and $(active_i^t.time - passive_j^y.time) \leq Propagate\_Max$.

  Informally, A1 states that if $i$ is not initially active then there must exist a neighboring process $j$ such that $i$ became active within $Propagate\_Max$ time unit of $j$ becoming passive. This places an upper bound on the time taken by an application-level interaction and ensures that if $i$ becomes active due to $j$, then it does so within $Propagate\_Max$ time units of observing the active state of $j$. This prevents $i$ from becoming active after an arbitrary amount of time.

- **Assumption A2**: $IG$ is a subgraph of $NG$.

  Informally, A2 states that if the application in the neighborhood of $v1$ can directly influence the application behavior in the neighborhood of $v2$ then $v1$ and $v2$ are able to directly communicate with each other.

- **Assumption A3**: Each node $P_i$ can detect every state change of the application that happens in the neighborhood of $P_i$.

  Intuitively, A3 ensures that an application entity cannot become active for a brief period of time in the neighborhood of $P_i$ without $P_i$ detecting it.

- **Assumption A4**: There exists an upper bound $Rtd$ on the round trip delay between any two neighboring nodes in $NG$.

  This assumption is useful in reducing the number of messages, and can be eliminated from our algorithms at the expense of additional messages.

- **Assumption A5**: We assume that there exists a tree spanning the entire network rooted at the consumer node. Although our algorithms only require that each node be able to route messages to the consumer node, A5 allows us to simplify the presentation and the implementation.

### III. TERMINATION DETECTION ALGORITHMS

In this section, we present our algorithms for termination detections. We first present two algorithms which are simple variations of existing termination detection algorithms. Subsequently, we present the results of our simulation experiments to compare the performance of the two algorithms. Finally, we present a hybrid algorithm which combines the features of the first two algorithms.
A. Algorithm TDAP1

Our first TD Algorithm for Pervasive systems, TDAP1, is based on the algorithm to detect termination of diffusing computations [3]. In [3], the computation is assumed to start at a single node, and the current state of the application is tracked by maintaining an activity tree. Initially, the initiator is the only node in the tree. If $A_i$ sends an application message to a passive process $A_j$, then $A_j$ is added to the tree with $A_i$ as its parent. The example in Figure 4 shows different various stages of the activity tree during a computation. In Figure 4(a), the computation has spread from node 1 to nodes 2 and 3. In (b), the computation has further spread to a larger set of nodes. Each message sent by $A_i$ to $A_j$ is responded to by an acknowledgement message. When a leaf node becomes passive, it sends an acknowledgement message to its parent and is removed from the tree. Figure 4(c) shows the stage when some of the acks (dashed edges) have been received. Finally, all acks have been received in (d).

The algorithm in [3] cannot be directly applied to pervasive systems for two reasons. First, as discussed earlier, it assumes that application level communication can be monitored. Secondly, the root of the tree initiates the computation and must participate in the algorithm until termination is detected. Similarly, other nodes such as the children of the root must also participate until the termination is detected. In pervasive systems, the application activity may move from one part of the network to another, and nodes in a previously active region may be unable to participate in the algorithm (for example, to conserve battery power, we may want to put them into a sleep state or nodes may get disconnected from their children due to link failures). Hence, it may not be desirable to rely on the participation of previously active nodes until the application terminates. Hence, the algorithm in [3] may not be directly applicable. [8] addressed a similar issue in the context of termination detection in wireless networks where network connectivity may be disrupted due to movement of nodes. The algorithm in [8] however assumes that the application level communication can be monitored.

We adapt the algorithm described above for pervasive systems by maintaining the application information at the consumer node. In the beginning, we assume that the consumer knows the set of nodes which are initially active. Whenever node $i$ becomes passive, it sends a Report message to the root indicating that it has become passive. However, before sending the report message, $i$ queries its neighbors to determine their current state. Node $i$ then includes the state information for the neighbors in the Report message. For example, in Figure 4(a), if node 1 becomes passive, then it would report that nodes 2 and 3 are active to the consumer node. This allows the root to update the application information which contains the set of nodes which can be potentially active at any given time.

We now present some more details of TDAP1. Since a node may change between active and passive states several times, each node $i$ maintains a variable $participation_i$, where $participation_i = x$ implies that $active_i^x$ has occurred. In addition, $i$ maintains an array $neighbor_active$, where $neighbor_active[j] = y$ implies that $active_j^y$ has occurred. When $passive_i^x$ occurs, $i$ waits for $Propagate_{Max}$ time units and then sends a $Iampassive(participation_i)$ message to all of the neighbors. This ensures that if a neighbor $j$ becomes active due to $i’$ active state, then the state change at $j$ would have already happened. When $j$ receive $Iampassive(x)$ message, it acts
as follows: First, it updates $\text{neighbor_active}[j]$ to $x$. If $j$ is active, then it will respond with an $Iamactive$ message with its participation number. Otherwise, it ignores the message since $j$ would have already sent an $Iampassive$ message earlier to $i$ (if $j$ did become active once). After sending the $Iampassive$ message to its neighbors, $i$ waits for $Rtd$ time to receive responses, following which $i$ sends a $Report$ message to its parent containing the participation number of each neighbor. Since a $Report$ message may have to be sent several times (each time the state changes to passive), we only include the incremental changes to the states of the neighbors in the $Report$ message. For example, as shown in Figure 5, when 4 becomes passive, it will report 5 as active (assuming nodes can only communicate with their horizontal and vertical neighbors). Similarly, in Figure 5(c), when 8 becomes passive, it will report 9 as active. The $Report$ messages are then further propagated to the consumer node via the parent nodes. Intermediate nodes in the tree aggregate the $Report$ messages to remove duplicate information before forwarding them (each node waits for a timeout interval to allow other report messages to be received and aggregated). The consumer nodes maintain the set of nodes which have been reported as active with their current participation number and removes nodes from this set as the corresponding $Report$ messages are received. When this set is empty, the consumer reports termination.

We have the following lemma for our first algorithm:

Lemma 1: TDAP1 correctly detects termination provided $A1, \ldots, A5$ are satisfied.

The proof depends on the following property: For any node $i$ which is not active initially, if $\text{participation}[i] = x$ then there exists a neighbor $j$ such that for some $y$, either $\text{passive}^y$ occurs after $\text{active}^x$ or occurs at most $\text{Propagate}\_\text{Max}$ time prior to $\text{active}^x$. This ensures that every active event will be reported to the consumer node, which in turn will wait for the reception of the corresponding report message.

B. Algorithm TDAP2

Another class of termination detection algorithms is based on an initiator node polling all of the other nodes to determine their states using a logical tree or a ring topology [4], [7], [6]. For example, in a ring based algorithm, the initiator node sends a message around a logical ring. Each node waits until it becomes passive and then forwards the message to the next node in the ring. When the initiator node receives the message back, it sends a message again around the ring to determine whether or not all nodes have remained passive. TDAP2 is a simple variation of this algorithm and uses a tree topology.

In this algorithm, the consumer node polls all of the nodes to find the current state of the system. We assume that there exists a tree node spanning the entire network (assumption $A5$). When a node becomes active, it sends an $Iamactive$ message to the root via its parent. This active message is forwarded to the root by a node provided it has not already forwarded one. On receiving this message, the root performs the following computation periodically: The root sends $Are\_you\_passive$ message to all children in the tree. Each node waits until it is passive and then propagates this message to its children in the tree. When a leaf node receives this message, it waits until it becomes passive. Subsequently, it sends $Iampassive$ message to its parent. These messages are aggregated as they move up the tree. When a node has received responses from all of its children, it waits until it becomes passive and then sends an $Iampassive$ message to its parent. After the root has received an $Iampassive$ message from all of its children, then it waits for $\text{Propagate}\_\text{Max}$ time and sends an $Are\_you\_still\_passive$ message to its children. This is required to ensure that any potential nodes which can become active due to their neighbors have already become active. The $Are\_you\_still\_passive$ message is propagated to all nodes in the tree. A leaf node responds with a $\text{Confirm}$ message if it has remained passive since the last time it sent an $Iampassive$ message. Otherwise, it sends an $\text{Active}\_\text{Again}$ message. If a non-leaf node receives an $\text{Active}\_\text{Again}$ message from a child or it has become active since the last time it sent an $Iampassive$ message, then it sends $\text{Active}\_\text{Again}$ to its parent. Otherwise, it sends $\text{Confirm}$ to its parent. If the root receives $\text{Active}\_\text{Again}$ from any child, it must repeat the cycle again after a timeout interval (the second phase in a cycle can be used to replace the first phase of the next cycle; hence, we only need to repeat the second phase). Otherwise, it declares termination.

We have the following lemma for this algorithm:

Lemma 2: TDAP2 correctly detects termination provided $A1,$
A3, A4 and A5 are satisfied. Note that this algorithm does not require assumption A2 as it polls all nodes in the network.

C. Comparison of TDAP1 and TDAP2

To compare the relative performance of the two proposed algorithms, we evaluated them using discrete event simulation. The performance of the algorithms depends on several factors which include: (a) the number of nodes involved in the computation at any given time, (b) the location of the computation with respect to the consumer node, (c) value of Propagate_Max, (d) the duration for which a node remains active before becoming passive, and (e) the number of times nodes change state from active to passive. We used a grid topology (of size 8x8) to model NG in our simulation experiments. In our experiments, we assumed that the application spans an area of size $k*l$ (spanning $k$ rows and $l$ columns). The simulation uses a configuration file as input which specifies the state of the computation in different stages. That is, we assume that the computation moves through a sequence of stages and the configuration file specifies the set of nodes active in each stage. This allows us to control the various parameters influencing the performance. For example, for the example in Figure 6, both $k$ and $l$ are 2 and the configuration file specifies four stages: In the first stage, 4, 5, 8 and 9 are active. The next stage has 5, 6, 9 and 10 as active. In the final state, nodes 5 and 9 again change state to active.

As can be seen, the number of messages in TDAP1 is directly proportional to the density whereas in TDAP2, this parameter has a smaller impact. However, even when the density is low, TDAP2 incurs a higher cost. Next, we changed the location of the computation with respect to the consumer node. Figure 8 shows the relative performance of the algorithms as the average distance of the application nodes to the consumer node is changed. As expected, the number of messages sent in TDAP1 increases as the number of hops travelled by the Report messages increases. There is a lower increase in the messages for TDAP2 as all messages (other than Iamactive message) are sent to all nodes in the network. Finally, we evaluated the impact of Propagate_Max. In

![Fig. 8. Performance of algorithms by varying the distance to the sink](image)

TDAP2, the number of messages is linked to the number of times the consumer node has to poll the nodes. When Propagate_Max is increased gradually, we found that there were points at which there is a step increase in the number of messages sent in TDAP2 (due to an additional round of polling). However, the number of messages in TDAP1 did not change significantly as Propagate_Max is increased (the only change observed was a reduction in the amount of aggregation of Report messages).

A drawback of TDAP1 is that it is sensitive to the number of times a node becomes active. Each time a node changes its state from active to passive, a report message has to be sent to the root. We conducted a set of experiments in which the number of times a node becomes active is varied. We start with a set of nodes as active and make them change state at different stages of the computation (vis appropriate entries in the configuration file). Figure 9 shows the relative performance of the algorithms as the active count (which denotes the number of times each node becomes active) is increased. As can be seen, TDAP1 is more sensitive to this parameter and its cost increase more rapidly as the frequency of nodes becoming active is increased while the number of nodes is kept constant.
As shown by the experiments above, each of the algorithms performs better than the other under certain conditions. Algorithm TDAP3 combines the features of TDAP1 and TDAP2. This algorithm assumes that the network is divided into a set of regions, and there exists a rooted tree in each region. The root node in each region is responsible for detecting termination in its region. We also assume that each root node can communicate with the root nodes of its neighboring regions (two regions are neighboring if they have nodes which can communicate directly with one another). For example, Figure 10 shows a network with six regions, each consisting of 8 nodes. In this topology, Region 1 has Regions 2 and 4 as its neighboring regions. The roots of each region (shown as shaded nodes) and the consumer node are organized in a logical tree rooted at the consumer node. One of the reasons for studying this type of organization is that we have built a sensor testbed with a similar structure. As shown in Figure 11, each group (which is a plexi-glass board) consists of 8 TelosB motes arranged in a 2*4 grid, one Stargate Netbridge and a USB-hub. Currently, we have 12 such boards to enable experimentation with a grid of 96 motes. The base station of each group is connected to a Stargate Netbridge gateway, and the gateways communicate via TCP. We have written a program which can communicate with the Stargates to configure into a given tree structure via TCP connections.

Algorithm TDAP3 is a combination of TDAP1 and TDAP2. Within each region, termination is detected using TDAP2. At the higher level, we view each region as a node in a super-graph, and use TDAP1 to detect termination for
the entire computation in the super-graph. This allows us to use TDAP2 in a smaller area and hence, each region’s root has fewer nodes to poll. Furthermore, since TDAP1 is being used between regions, only those regions which are involved in the computation are polled to detect termination. When a node becomes active, it behaves as in TDAP2 and sends an *I am active* message to the root in its region. When this message is received, the root sends an *Are you passive* message to the nodes in its region as in TDAP2. In addition, it also becomes active with respect to TDAP1. When TDAP2 detects termination in a region, the root node of that region becomes passive in TDAP1 and sends an *I am passive* message to the root nodes of its neighboring regions. The roots of the neighboring regions respond back with their state as in TDAP1. On receiving this information, a Report message is sent to the consumer node. Note that a region may become active again after TDAP2 has reported termination. Hence, we may have to execute TDAP2 several times. [9] also presented an algorithm which assumes a hierarchical organization similar to the one in TDAP3. However, it was targeted for traditional wireless networks with layered design and combined a different set of algorithms.

V. CONCLUSION

We have studied the problem of termination detection in pervasive sensor networks. The proposed algorithms address this problem in systems where the termination detection algorithm is unable to monitor application level communication. We presented two algorithms for this problem, one of which is based on termination detection of diffusing computations and the other on polling of nodes. We showed via simulation that each of these algorithms performs better than the other under certain conditions. Subsequently, we presented an algorithm which combined the features of the two algorithms. Our experimental results showed that this combined algorithm performs better than the component algorithms.

ACKNOWLEDGEMENT

This work was supported by NSF grants CNS0615337 and CNS0551626 and DARPA contract F33615-00-C-3044.

REFERENCES


