Report Number: WUCS-01-29

2001-01-01

A Termination Detection Protocol for Use in Mobile Ad Hoc Networks

Gruia-Catalin Roman and Jamie Payton

As computing devices become smaller and wireless networking technologies improve, the popularity of mobile computing continues to rise. In today's business 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 will need to be deployed. Such a setting poses new challenges for the software developer, e.g., the lack of an established network topology, bandwidth limitations, and frequent disconnections. In this paper, we begin to explore design strategies for developing applications over... **Read complete abstract on page 2.**

A Termination Detection Protocol for Use in Mobile Ad Hoc Networks

Gruia-Catalin Roman and Jamie Payton

Complete Abstract:

As computing devices become smaller and wireless networking technologies improve, the popularity of mobile computing continues to rise. In today's business 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 will need to be deployed. Such a setting poses new challenges for the software developer, e.g., the lack of an established network topology, bandwidth limitations, and frequent disconnections. In this paper, we begin to explore design strategies for developing applications over ad hoc networks. The study of termination detection in diffusing computations, along with the formulation of an algorithmic solution amenable to usage in mobile ad joc networks, gives us the opportunity to bring to light several important software engineering concerns and design strategies one might employ in a mobile setting. We view this effort as a first step towards creating a repertoire of commonly used design solutions for frequently encountered problems in the development of applications over mobile ad hoc networks.

A Termination Detection Protocol for Use in Mobile Ad Hoc Networks

Gruia-Catalin Roman and Jamie Payton

WUCS-01-29

October 2001

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis MO 63130

A Termination Detection Protocol for Use in Mobile Ad Hoc Networks

Gruia-Catalin Roman and Jamie Payton
Washington University in St. Louis
Campus Box 1045, One Brookings Drive
St. Louis, MO 63130-4899, USA
{roman, payton}@ cs.wustl.edu

ABSTRACT

As computing devices become smaller and wireless networking technologies improve, the popularity of mobile computing continues to rise. In today's business 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 will need to be deployed. Such a setting poses new challenges for the software developer, e.g., the lack of an established network topology, bandwidth limitations, and frequent disconnections. In this paper, we begin to explore design strategies for developing applications over ad hoc networks. The study of termination detection in diffusing computations, along with the formulation of an algorithmic solution amenable to usage in mobile ad hoc networks, gives us the opportunity to bring to light several important software engineering concerns and design strategies one might employ in a mobile setting. We view this effort as a first step towards creating a repertoire of commonly used design solutions for frequently encountered problems in the development of applications over mobile ad hoc networks.

Keywords

Mobile computing, ad hoc network, termination detection, diffusing computation, design.

1. INTRODUCTION

Mobile computing evokes images of a businessperson using a laptop in a conference room, a PDA on a business trip, or a cell phone in a taxicab. Mobile computing is not limited to these applications. Exciting, yet practical, uses of mobile devices are already in existence, and new mobile devices and uses for them are emerging. Today, retail store employees use mobile devices to check inventory, runners have devices that compute their heart rate and speed while on the track, and music lovers have MP3 players that can store and play music anytime and anywhere. In the near future, the U.S. Army will equip its soldiers with wearable devices to assist in urban warfare strategy and communication.

The strong demand for mobile computing devices warrants the development of new, innovative software that is designed for communication over ad hoc networks. A good software development practice is to apply established design strategies to the problem at hand. Few, if any, are currently available in the ad hoc setting even though collections of design strategies exist and are widely used in distributed computing. The latter range from high levels of abstraction, e.g., design patterns [2, 6], to specific algorithms for standard problems encountered in distributed system implementation, e.g., [7]. Our goal is to establish a repertoire of design strategies for mobile computing applications over ad hoc networks.

Our first objective is to identify important problems in mobile computing and to offer practitioners reasonable solutions that will eventually be routinely included in delivered software. One such problem is termination detection in diffusing computations, which we study in this paper. Our second objective is to show that the development of mobile applications demands a new way of thinking, one rooted in formal methods even when applied informally. In presenting our algorithmic solutions to the termination detection problem, we hope to demonstrate a pragmatic style of employing basic formal thinking skills to this new problem domain. Finally, we seek to bring to life some of the complexities of mobile computing for the benefit of mobile application developers.

We focus on termination detection because it is an important practical problem in distributed processing, important in a mobile ad hoc network as well. In a field command and control application, for instance, the commander must be sure that all participants are in position and are ready before issuing an attack directive. Similarly, in a wireless classroom setting, all students must submit their test before the teacher opens up the discussion of possible problem solutions.

A well-known solution to termination detection in diffusing computations is the algorithm proposed by Dijkstra and Scholten [5]. 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 depends upon constant connection among the child and parent nodes. In a mobile setting, however, it is likely that a parent will become disconnected from the network before a child can deliver a termination notice. When designing applications in a mobile ad hoc network, coordination issues such as discovery of other nodes, synchronization of actions, and exchange of information must be viewed differently because of the high frequency of disconnections. Solutions should allow for propagation of termination messages to be independent of the connectivity to the parent node, and cannot depend on the existence of a wired infrastructure. Eventual delivery of termination signals to the root node should be the only requirement.

The termination detection protocol for mobile systems presented in [9] takes a hybrid approach that combines two existing protocols, a weight-throwing scheme and a diffusion-based scheme. This solution is defined for nomadic computing, in which mobile devices are connected to the wired infrastructure. No connection to the wired network exists in a mobile ad hoc network. Ideas borrowed from termination detection in diffusing computations have been used in mobile ad hoc networks for constructing a message delivery algorithm [8]. Thus, while termination detection in mobile distributed systems has been studied, new solutions to this particular problem are still needed for use in ad hoc networks.

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. Section 4 contains a discussion of the results. Concluding remarks are presented in Section 5.

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 get established when devices move within communication range. The network topology changes as hosts join or leave the ad hoc network. Connections within such networks are affected by power limitations, resource availability, physical locations, etc. Typically, the mobile hosts communicate directly in a point-to-point fashion, but protocols for ad hoc routing are being developed. In this paper, we assume only the case of pair wise connectivity.

Our interest is to examine the issue of termination detection in such networks. We focus on diffusing computations because, in the ad hoc setting, it is desirable to keep the number of participants in a computation as small as possible. We simply assume that the computation spreads out as nodes "bump" into each other. 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 epidemic updates, the software is updated locally on a host. In the first phase of processing, the update is propagated to all other hosts. The originator of the update must know that all hosts received the update. In the second phase of processing, the originator begins to notify hosts that the software can be activated. This phase does not require the host to obtain knowledge of completion of processing. An instance of this type of software update might occur when a new communication key is generated and sent to others in the group. The old communication 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 key 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 root, which initiates the computation and is charged with ultimately detecting the termination. The root node can activate idle hosts in the mobile ad hoc network by sending a request for a task to be performed, while connected. 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. 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. To detect termination, it is necessary to ascertain at the root node that, at some point after the start of computation, all mobile hosts in the ad hoc network that were once activated are idle again. 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 problems 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 mobile nodes may be very large but closed. Let v_0 be a distinguished node, representing the root 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 root node v_0, $u.done$ remains false for all other nodes. Given these assumptions, the termination detection problem assumes its classical formulation:

Given[1]

$$W \equiv < \forall u : u \in V :: u.idle >$$

stable W

construct a protocol such that[2]

$v_0.done$ **detects** W.

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). Finally, a node may 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. Note that the algorithm assumes synchronous communication. In Figure 1, the symbols "+" and "-" represent set union and difference, respectively.

The diffusing computation begins with a single node being active. 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 and still in the idle state (action $IssueActivatingRequest_B$). 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 it hides the presence of some additional message exchanges 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 latter must keep track of all the nodes it activated throughout its life time. 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*) 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 an completion history (variable *idleReports*). This may appear at first to have broken the activation tree but this is not so. 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 they 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 if all idle reports reach 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) and repeats the process until the only node left in the tree is the idle root. 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 mobile 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. It can be easily shown that the distance between the current location of an idle report and the root (measured in terms of the shortest path in the partial order) decreases with each such transfer of information. With each encounter the idle reports get closer and closer to the root. By induction we can establish that they eventually reach the root. Even with the partial order in place, the proof still needs to rely on the assumption that a node carrying idle reports eventually meets another node that is logically closer to the root. Without this, no solution exists since nodes may go away never to come back. The information

[1]The stability requirement indicates that, once established, W will continue to hold forever.

[2]Detection is defined as a combination of two properties. First, once set, the termination flag $v0.done$ guarantees that all nodes are idle. Second, if all nodes become idle, the fact is eventually recorded by setting the termination flag. The first condition is a safety property while the second is a progress property.

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

Auxiliary Variables

$activated \equiv (activatedChildren \neq \emptyset)$
$never\_activated \equiv \neg activated$
$active \equiv \neg idle$

Actions at A

DetectTermination
Precondition:
$root \wedge idle \wedge idleReports = \emptyset$
Effect:
$done := true$

IssueActivatingRequest$_B$ – 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

IssueAnotherRequest$_B$
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

AcceptWorkRequest – activation message arrives at A from B
Let the message be $job(task)$
Effect:
if $never\_activated$ then
$idle := false$
end
$perform\_the\_task$

NodeTerminates – node A terminates
Precondition:
$active \wedge no\_work\_to\_do$
Effect:
$idle := true$
$idleReports := idleReports + \{(A, activatedChildren)\}$

PropagateIdleReports – 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

AcceptIdleReports – node A receives computation states from B
Let the message be $nodeInfo(new\_reports)$
Effect:
$idleReports := idleReports + new\_reports$

RemoveIdleLeaves
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

they held would be lost and no final determination of the termination status would be possible. This is one of the realities of mobile computing in ad hoc networks.

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. 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 both places without any negative impact on the detection process and the integrity of the activation tree.

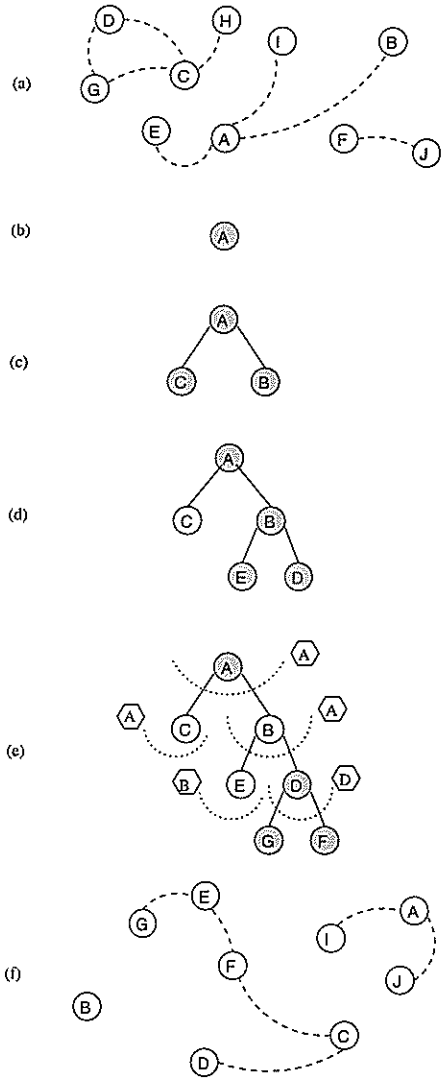

Figure 2: Example Activation Tree

An example of an activation tree that our algorithm might produce is shown in Figure 2. Active nodes in the activation tree are shaded. A snapshot of the ad hoc network topology at the time that the diffusing computation begins is shown in Figure 2a. Node A is the originator of the computation, and becomes the root of the activation tree (Figure 2b). Node A activates node B and node C. These nodes are added to the activation tree (Figure 2c). Node B activates two more nodes, node D and node E. At some point, node C finishes its work and becomes idle (Figure 2d). Node D activates node F and node G, which are added to the tree. In the meantime, node B terminates (Figure 2e). Notice that B is the parent of D, and is terminated, but not removed from the tree. The labeled hexagons indicate where the information about that parent/child relation is actually stored. For instance, node E is idle and its idle report resides at node B. Similarly the idle report of B is stored at A. By contrast, the active node D still stores its relation to its children. A snapshot of the network topology after termination is detected is shown in Figure 2f.

4. DISCUSSION

In this section, we revisit some of the assumptions made earlier and some subtle aspects of the solution.

The first concern we need to address is the assumption that a node carrying idle reports eventually meets another node closer to the root. In truth, nodes may leave the network at any time, never to return. The question we must ask is, how do we ensure that termination is detected at the root if a node never returns its idle report? One possible solution is to employ some timeout mechanism and to "push out" of consideration the wayward node. In other settings, we may employ probabilistic analysis or may be able to take advantage of known movement patterns, e.g., carts moving back and forth on a given set of tracks.

We also assumed 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 by the root.

We also postulated the existence of a partial ordering for message delivery purposes. An obvious question is, how can nodes involved in the compuation be arranged in a partial ordering? An obvious answer is to assign numerical identifiers to hosts. There are several possibilities. 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 assingment too inflexible. Dynamic assignment of identifiers is another possibility. One approach of particular interest dynamically assigns numerical values to hosts according to their relative distance to the destination. This approach, the Disconnected Transitive Communication (DTC) model [3], uses a value called a *utility* to determine if one host is closer to another. 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 set 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 existence of other bodies of work with similar goals is en-

couraging, e.g., research on epidemic routing [10]. The goal of epidemic routing 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, host identifiers are assigned according to the last bits in the hosts's IP address. Algorithms for token based mutual exclusion dynamically assign numerical identifiers to hosts in such a way that a total ordering is formed [4, 11].

Returning to the notion of using a partial order for message delivery 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 we assume that every node will eventually encounter the root in its travels, all nodes may be viewed as being immediately greater than the root node. Two non-root nodes that meet would not exchange any data since, according to this partial order; they are not comparable with each other.

(2) 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. Two nodes can easily determine whether they both participated in the computation and their relative ordering relation.

(3) 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.

(4) 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.

These ideas clearly deserve further exploration and we plan to examine them in future work, outside the scope of this paper.

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 accomodate 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 [1, 12] or cluster representatives that are used in ad hoc routing protocols.

Finally, we note that the activation tree is what we often refer to as a global virtual data structure. It is an abstract representation of the global system state that has an explicit, concrete, but distributed, representation which is being manipulated by actions at the local level. The semantics of these actions are associated with constrained changes in the local data but can also be assigned global semantics that capture their impact on the overall system. Local semantics relate to coding the programs while global semantics facilitate reasoning about the computation.

5. CONCLUSION

In this paper, we have examined the problem of detecting termination in diffusing computations. 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 the frequent disconnections among hosts renders the standard solution ineffective. In turn, we devised a new algorithm to achieve termination detection in mobile ad hoc networks, one that does not rely on persistent connections among hosts to achieve the termination condition.

By identifying termination detection as a useful problem and offering a solution well suited to mobile ad hoc networks, we have begun the journey toward achieving our goals. First, we took the first step towards providing a practical solution to an important problem faced in developing applications for mobile computing. Second, in working on the termination detection problem, we put forth additional reasons why goal-directed routing is emerging as an important problem in mobile computing. 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 and on the notion of global virtual data structures.

Acknowledgements

This research was supported in part by the National Science Foundation under Grant No. CCR-9970939. 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 National Science Foundation.

6. REFERENCES

[1] D. Baker and A. Ephremides. A distributed algorithm for organizing mobile radio telecommunication networks. In *Proceedings of the Second International Conference on Distributed Computer Systems*, pages 476–483, Apr. 1981.

[2] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *A System of Patterns: Pattern Oriented Software Architecture.* Wiley, 1996.

[3] X. Chen and A. Murphy. Enabling disconnected transitive communicaiton in mobile ad hoc networks. In *Proceedings of Workshop on Principles of Mobile Computing*, pages 21–27, August 2001.

[4] D. Dhamdhere and S. Kulkarni. A token based k-resilient mutual exclusion algorithm for distributed systems. *Information Processing Letters*, 50:151–157, 1994.

[5] E. Dijikstra and B. Scholten. Termination detection for diffusing computations. *Information Processing Letters*, 11(1):1–4, 1980.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, October 1994.

[7] N. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers, 1996.

[8] A. Murphy, G.-C. Roman, and G. Varghese. Tracking mobile units for dependable message delivery. Technical Report WUCS-99-30, Washington University, Department of Computer Science, St. Louis, Missouri, 1999. To appear in *IEEE Transactions on Software Engineering.*

[9] Y. Tseng and C. Tan. Termination detection protocols for mobile distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 12(6), June 2001.

[10] A. Vahdat and D. Becker. Epidemic routing for partially-connected ad hoc networks. Technical Report CS-2000-06, Duke University, July 2000.

[11] J. Walter, J. Welch, and N. Vaidya. A mutual exclusion algorithm for ad hoc mobile networks. In *Proceedings of 2nd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, Oct. 1998.

[12] J. Zavgren. Ntdr mobility management protocols and procedures. In *Proceedings of the IEEE Military Communications Conference*, Nov. 1997.