Report Number: WUCS-98-01

1998-01-01

Search and Tracking Algorithms for Rapidly Moving Mobiles

Amy L. Murphy, Gruia-Catalin Roman, and George Varghese

With the advent of wireless technology and laptops, mobility is an important area of research. A fundamental problem in this area is the delivery of messages to a moving mobile. Current solutions work correctly only for slowly moving nodes that stay in one location long enough for tracking to stabilize. In this paper we consider the problem of message delivery to rapidly moving mobile units. With these algorithms, we introduce a new method for designing algorithms based on the paradigm of considering a mobile unit as a message, and adapting traditional message passing algorithms to mobility. Our first algorithm... **Read complete abstract on page 2.**

Search and Tracking Algorithms for Rapidly Moving Mobiles

Amy L. Murphy, Gruia-Catalin Roman, and George Varghese

Complete Abstract:

With the advent of wireless technology and laptops, mobility is an important area of research. A fundamental problem in this area is the delivery of messages to a moving mobile. Current solutions work correctly only for slowly moving nodes that stay in one location long enough for tracking to stabilize. In this paper we consider the problem of message delivery to rapidly moving mobile units. With these algorithms, we introduce a new method for designing algorithms based on the paradigm of considering a mobile unit as a message, and adapting traditional message passing algorithms to mobility. Our first algorithm is based on tracking and can be efficient when the path of the mobile node exhibits considerable locality. While it uses the Dijkstra-Scholten algorithm for diffusing computations as a point of departure, the final algorithm adds a number of mechanisms. Our second example is a search algorithm based on transforming the classic Chandy-Lamport snapshot algorithm. The algorithm generalizes to multicasting to a set of rapidly moving mobiles. Both our algorithms are based on the assumption that messages and mobiles travel through the same FIFO channels. We show how to enforce this assumption by modifying existing handover protocols.

Search and Tracking Algorithms for Rapidly Moving Mobiles

Amy L. Murphy
Gruia-Catalin Roman
George Varghese

WUCS-98-1

June 1, 1998

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

Search and Tracking Algorithms for Rapidly Moving Mobiles

Amy L. Murphy, Gruia-Catalin Roman, George Varghese

{alm, roman, varghese}@cs.wustl.edu

Washington University, Department of Computer Science

Campus Box 1045, Bryan 509/ One Brookings Dr.

St. Louis, MO 63130

June 1, 1998

Abstract

With the advent of wireless technology and laptops, mobility is an important area of research. A fundamental problem in this area is the delivery of messages to a moving mobile. Current solutions, work correctly only for slowly moving nodes that stay in one location long enough for tracking to stabilize. In this paper we consider the problem of message delivery to rapidly moving mobile units. With these algorithms, we introduce a new method for designing algorithms based on the paradigm of considering a mobile unit as a message, and adapting traditional message passing algorithms to mobility.

Our first algorithm is based on tracking and can be efficient when the path of the mobile node exhibits considerable locality. While it uses the Dijkstra-Scholten algorithm for diffusing computations as a point of departure, the final algorithm adds a number of mechanisms. Our second example is a search algorithm based on transforming the classic Chandy-Lamport snapshot algorithm. The algorithm generalizes to multicasting to a set of rapidly moving mobiles.

Both our algorithms are based on the assumption that messages and mobiles travel through the same FIFO channels. We show how to enforce this assumption by modifying existing handover protocols.

1 Introduction

Mobile computing is rapidly changing from being the exception to being the norm. Laptop computers are nearly everywhere people are, especially when people are traveling. Not only do people need to work in a disconnected mode of operation, they also need the advantages that come with remaining connected to the wired infrastructure without the bounds of a wire, and without concerning themselves with massive reconfiguration each time they move to a new area.

To some degree, cellular telephones already provide such services, and their success has led to the adoption of a similar model for mobile computing. Figure 1a shows a typical cellular telephone model with a single mobile support center (MSC) in each cell. The MSC is responsible for communication with the mobiles within its region and serves as a manager for handover requests when a mobile moves between MSCs. Figure 1b shows how the cellular telephone model is transformed into a graph of nodes and channels. For the purposes of this paper, we will assume that when a mobile unit moves between two MSCs, it is modeled as being on a channel and therefore disconnected from the network.

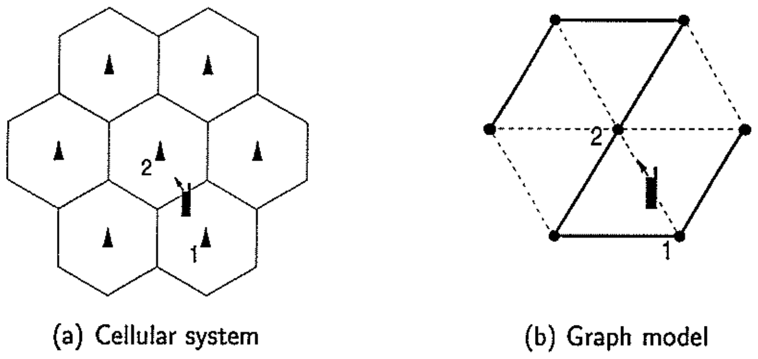


Figure 1: (a) Cellular system with one MSC per cell. All MSCs are assumed to be connected by a wired network. (b) Abstract model of a cellular system, as a graph of nodes and channels. Solid line indicates a spanning tree. A mobile unit moving across the border between two cells may miss a simple broadcast along a spanning tree if the handover occurs between the broadcast at MSC_2 and the broadcast at MSC_1.

One of the central problems in mobility is delivering a message to a mobile unit as it moves. Most applications in the Internet today rely on sending messages to particular locations. For example, e-mail assumes that the sender knows a particular address for the recipient. In a fixed network, routing protocols such as link state routing [12] work over a fixed topology collecting information about the direction of each host. When a message arrives, the router performs a table lookup to decide which port, or direction, to send the message. Although all routing protocols allow for topology changes, either due to link failures or policy changes, the length of time to propagate such changes is not adequate to support mobility.

One model of mobility currently being implemented fits on top of the fixed network, effectively providing specific points of connection for the mobiles. In Mobile IP [10], every mobile is associated with a fixed home agent. This agent provides a forwarding service for the mobile. When the mobile moves to a new location, it contacts its home agent with new contact information. Messages for the mobile are sent to the fixed address of the home agent. When a data message arrives, it is encapsulated by the new location of the mobile, and sent back on the network. Because the outer address is now the foreign location of the mobile, the fixed network successfully routes the message to the mobile, where it is decapsulated and processed.

In cellular telephones, a system similar to Mobile IP is employed when users roam outside their home region [13]. When the telephone is activated, the user registers with the home, indicating essentially a new area code to redirect calls to. The registration process occurs infrequently because the most common case is for the user to move within a single region. Within that region, another approach must be taken to locate the mobile each time a call arrives, and handovers are used to maintain connectivity when a user crosses a cell boundary during a session.

The Mobile IP and cellular telephone solutions work well with slowly changing systems where the mobile unit moves infrequently and the registration process has time to converge. This is especially true because mobiles are carried by people who move using mechanical devices (e.g., cars, planes) that are much slower than message transmission speeds.

However, if the mobile moves rapidly through different cells, the information at the home agent will reflect an old location, and messages sent to that location will be dropped. Clearly, a forwarding mechanism can be added to the foreign locations, having them send these otherwise lost messages to the mobile's next location. However, with rapid movement, the messages can keep chasing the mobile without delivery, following the trail of forwarding pointers. At the same time, the amount of forwarding information can increase dramatically. Although such rapid movement may seem unlikely, one of the trends in mobility is to reduce the size of the cell (e.g., nanocells) to increase the frequency reuse. As cell sizes decrease, the time that it takes to traverse a cell similarly decreases.

Another possible application that is characterized by rapid mobile movement is mobile code where it is not a physical component that moves, but rather a program moving along the fixed network doing computation at various network nodes. Rather than connecting to a foreign agent through a wireless mechanism, these mobile code agents actually execute at a foreign host. They have the ability to move rapidly from one host to another and may not register each new location with a home. Therefore, delivering a message to a mobile code agent becomes an interesting application area in which rapid movement is not only feasible, but is the common case.

Thus while rapid mobile movement may not be common today, it is likely to be a problem of practical interest in the future. Besides the potential practical impact, we believe that the following question is a fundamental one in the field of asynchronous network protocols: *Can we devise efficient protocols that can guarantee delivery to a node (or set of nodes) that can move at arbitrary speeds across a fixed network?* Clearly, trivial solutions exist that broadcast the message to all nodes and store the message at all nodes until the mobile arrives. By contrast, more efficient solutions should limit the broadcast range and/or the storage required.

Mobile-as-Message Paradigm and Paper Outline: In this paper, we propose adapting existing distributed algorithms to perform guaranteed message delivery in a mobile system with rapidly moving mobile units. In brief, our strategy is to take existing distributed algorithms that use message passing, substitute mobile units for messages, and find useful applications for the resulting algorithms.

We present two algorithms that follow this paradigm, each of which is applicable to the problem of message delivery. Because both of our algorithms rely on channels being with respect to both mobile units and messages, we first provide a simple mechanism to guarantee FIFO channel behavior in Section 2. Using this mechanism, our first algorithm tracks the current location of the mobile, and uses the tracking trail for efficient delivery. The algorithm starts with the main idea of the termination detection algorithm proposed by Dijkstra and Scholten [5], and adds considerable modifications. It is described in Section 3. Our second result adapts the snapshot algorithm by Chandy and Lamport [4] to search for a mobile unit. This is described in Section 4 together with an extension to multicasting to a set of mobiles. Section 5 describes related work while Section 6 states our conclusions and some future directions.

2 Enforcing the FIFO assumption

Before presenting our algorithms, we preemptively dispel one potential objection to the use of FIFO channels. Specifically, we model both the mobile units and the messages as traveling on the same channel. Although this is the case for mobile code units, with physically moving mobile units such as laptop computers, this assumption appears to be errant nonsense because mobile units move slowly through space while messages move rather quickly on a wire between MSCs. However, because many distributed algorithms use FIFO channels and our objective is to apply these algorithms to the mobile environment, the lack of FIFO channels in the mobile environment makes this difficult. Although flush primitives [2] can be used to make traditional non-FIFO channels FIFO, the separate channels used for mobiles and messages makes Ahuja's flush primitives inapplicable to mobility. To further understand this problem, we must explore a more detailed model of reality, and see exactly how the FIFO assumption can be broken. We then propose a simple fix and show why the resulting channel is indeed FIFO.

One of the American standards for analog cellular communication is AMPS [13]. In AMPS, cellular telephones tune to only one frequency at a time. When the signal from one mobile unit begins to degrade, MSC_1, communicating with it on frequency f_1, sends a message to all bordering cells requesting the signal strength of messages on f_1. After the responses are collected, MSC_1 selects the strongest signal as coming from the cell the mobile unit is moving into, MSC_2. At this point, a handover begins. Figure 2a shows the control messages exchanged among the two MSCs and the mobile unit. First, MSC_1 sends a *frequency request* to MSC_2. MSC_2 responds with an available frequency f. Next, MSC_1 sends a *switch* message to the mobile unit with the new frequency. The mobile unit changes to the new frequency and sends a *hello* to

MSC$_2$. Finally, MSC$_2$ sends a *handover-complete* message to MSC$_1$.

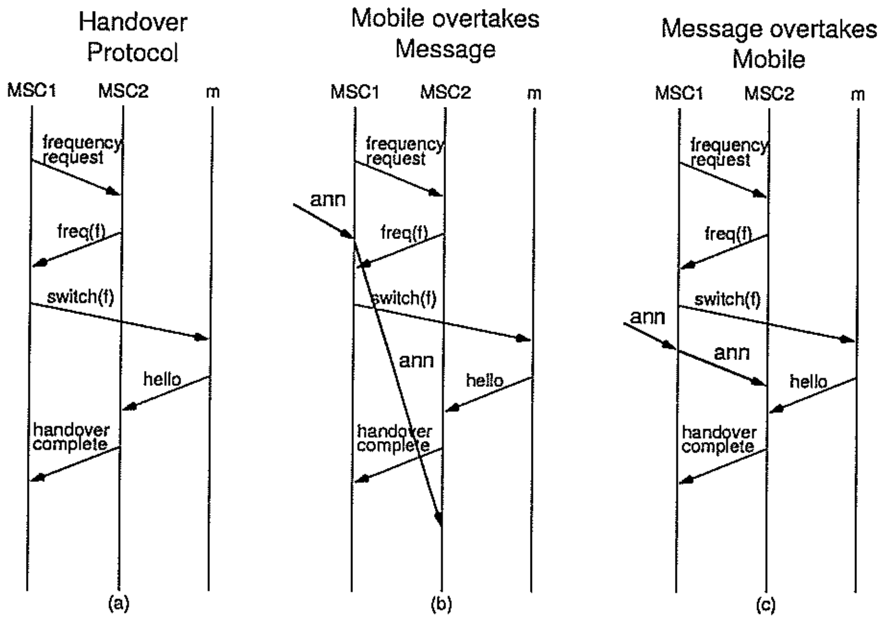


Figure 2: AMPS handover protocol (a). If messages are processed immediately upon receipt, it is possible (b) for the mobile to move faster than the message along the channel , or (c) for the message to move faster than the mobile, thus breaking the FIFO channel property. By delaying the message processing until after the *handover-complete*, both situations are avoided.

By using the AMPS approach, we know when a mobile unit is moving between cells, and which cells it is moving between. We also note that the mobile unit is not involved in the handover until the last moment when it changes the frequency it is tuned to. Although this is slightly different with digital technologies and soft handovers, we will leave that discussion for another paper.

Because we are concerned with keeping the channels FIFO with respect to the mobile unit and the data messages (henceforth referred to as *announcements* to maintain the distinction between data and control messages), we identify the two cases in which the channels are no longer FIFO. Figure 2b shows how an announcement is put into the channel prior the the mobile leaving, however the mobile arrives at the destination before the announcement. This is possible because the actual handover for the mobile is simply a series of messages and is not related to physical movement. The situation is as follows: a handover has already begun to transfer a mobile, m, from MSC$_1$ to MSC$_2$. Before the new frequency arrives, a new announcement arrives. The announcement is immediately put onto the channel from MSC$_1$ to MSC$_2$. The new frequency arrives and the switch message is sent to m, effectively putting the mobile onto the channel following the announcement. The messages to complete the handover move over the wireless channel and need not interact with the announcement, therefore it is possible for the mobile to arrive at MSC$_2$ before the announcement. Therefore, the mobile appears to have traveled across the channel faster than the announcement. In this case, we have effectively given the mobile multiple copies of the same announcement: once at MSC$_1$ and again at MSC$_2$. Although such duplicate delivery may not be a problem for the mobile to deal with (by keeping announcement identities), it is better to limit the power consumption of the mobile necessary to receive an announcement. Also, because we are separating the FIFO nature of the channel from the algorithm, we should not make any assumptions about the ability of an algorithm to handle this type of behavior.

Another more serious situation is that of the announcement overtaking a mobile, shown in Figure 2c. In this case, an announcement arrives at MSC_1 after the *switch* message has been given to the mobile and the mobile is already in the channel. At this point, the announcement is put into the channel, following the mobile. Unlike before, we assume that the announcement travels faster than the remainder of the messages of the handover, allowing the announcement to arrive at MSC_2 before the mobile. If we assume delivery as before, the mobile may not receive any copies of the announcement.

These problems arise because instead of having one channel on which both announcements and mobile units travel, we have, in effect, two channels which are not synchronized. We have also defined a mobile to be on the link after the *switch* message is sent and until the *hello* is received by the next MSC.

Our task is to use the details of the handover protocol and the arrival of the announcement to make the channel FIFO without regard for the underlying algorithm that depends on the FIFO channels. Intuitively, we accomplish this by treating any mobile unit that is involved in a handover as being on a channel and inaccessible during the handover. Because the sender initiates the handover, we modify its behavior by adding the mobile that is leaving to a handover set. Whenever an announcement arrives for a mobile in this set, its arrival is delayed until the mobile has left the MSC. As soon as the *handover-complete* is received, this delayed announcement can be processed and the mobile removed from the handover set. Nothing needs to change at the receiver.

In order to show that this provides us with FIFO channel behavior, we look at four conditions:

1. announcements arrive in FIFO order with respect to one another.

2. mobiles arrive in FIFO order with respect to one another.

3. announcements sent after mobiles arrive after mobiles.

4. mobiles sent after announcements arrive after announcements.

The first condition is trivial because we assume a reliable data link layer that ensures the ordering of messages on a single link. The second condition cannot be maintained because even though two mobile units, m_1 and m_2, are sent *switch* messages one after the other, the mobile units may send *hello* messages that arrive at the receiver in the reverse order. Therefore, we cannot guarantee FIFO behavior among mobiles, but this is not necessary for our message delivery algorithms.

The third condition is the most important. All announcements that are sent before the mobile unit is sent on the channel must arrive before the mobile unit because there is at least one message (the *frequency-request* message) that must follow the announcement down the channel, effectively flushing it out. During the period when a mobile is involved in the snapshot (after the *frequency-request* and before the *handover-complete*) all announcements received will be delayed. It follows that none will be on the channel until after the mobile has arrived at the destination, and therefore cannot arrive at the destination until after the mobile. For the final condition, we note that the handover requires at least one message to be sent (*frequency-request*) along the same channel as the announcement. Therefore, any announcements in the channel before a handover is initiated will be delivered before the handover completes since they will be flushed out of the channel by the *frequency-request*.

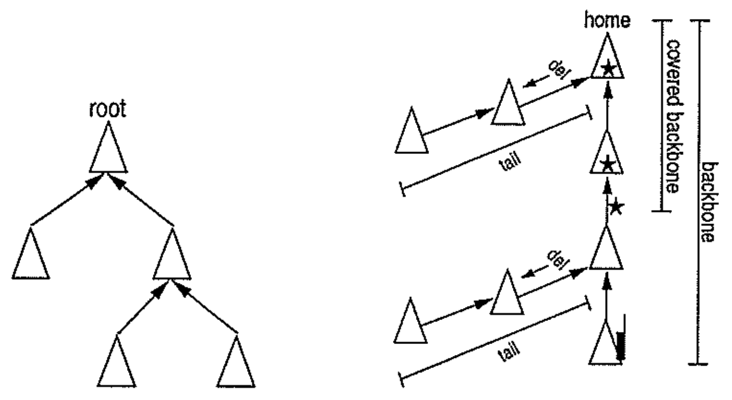
By delaying the processing of an announcement while a mobile is in transit, we have effectively reduced the handover process to an atomic operation and forced the channels to be FIFO without any delay on the movement of mobiles and only minor delays in the processing of announcements. Notice that we only delay announcements addressed to the mobile (either unicast or multicast) and not all messages. For the rest of the paper, we assume that that mobile nodes and announcements are FIFO with respect to each other.

3 Announcement Delivery through Tracking

For our first algorithm, we consider the case where a mobile's movement exhibits considerable locality and searching the entire network area is not necessary. Consider the situation of a cable repairman who travels from her office to a particular region. She stays within that region through the day; although the region is not known in advance, it is likely that it is highly localized (i.e. most repairs will be in a single suburb). Therefore, we would like to limit the number of MSCs involved in the delivery to those in that small region. To accomplish this, we turn to a tracking solution inspired by the termination detection algorithm for diffusing computations proposed by Dijkstra and Scholten [5].

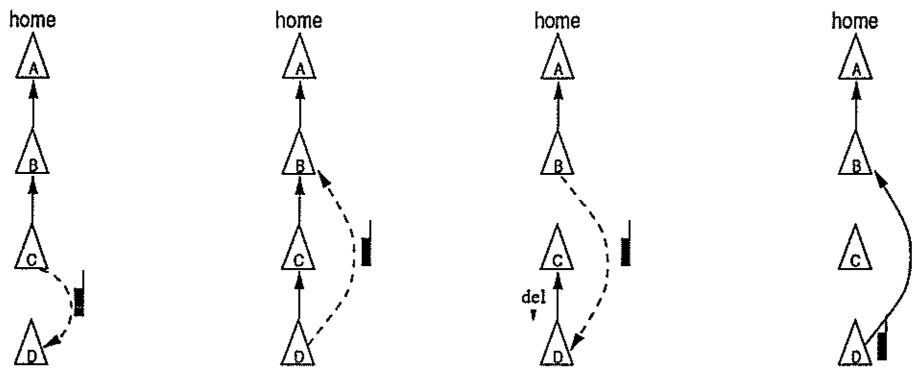
Figure 3 shows a general diffusing computation which starts from a single root node and forms a tree as messages are passed. A parent node keeps a count of the number of children it has, and the child keeps a pointer back to the parent. When a child goes idle, and it has no more children, it reports this to the parent and removes itself from the tree. When all children have been removed and the root is idle, the algorithm has terminated.

In the mobile model, we model a mobile unit as a message. For every mobile, we establish a home node and construct a separate tree rooted at the home node that reflects its movement. A node can remove itself from the tree when the mobile is not present and it has no children. A delivery algorithm that guarantees delivery can then be added on top of this tree by sending announcements first to the home and disseminating them through the tree. However, we note that this still uses too many nodes. The only nodes and channels that must be involved in the delivery are those on the path from the root to the mobile unit. We refer to this as the *backbone* (see Figure 3b). Therefore, we modify the graph growth algorithm to yield the backbone, and generate explicit delete messages to remove all nodes not on the backbone. We refer to non-backbone nodes as being part of a tail. There can be multiple tails, but only one backbone.



(a) Sample diffusing computation (b) Modified graph showing new structure.

Figure 3: By adapting diffusing computations to mobility, we construct a graph reflecting the movement of the mobile. In order to deliver an announcement, the only part of the graph we need is the path from the root to the mobile, the *backbone*. Therefore we adapt the Dijkstra-Scholten algorithm to maintain only this graph segment and delete all the others.



(a) Backbone extended (b) Backbone shortened (c) Tail node added (d) After movement completes

Figure 4: The backbone changes as the mobile moves to (a) a node not in the backbone, (b) a node higher in the backbone, and (c) a tail node. (d) shows the state after all channels have cleared.

To understand the overall approach, we will look at how the graph changes as the mobile unit moves. First, however, it is useful to note that because of the definition of the backbone, the mobile unit will always be located at the last node in the backbone or on a channel leading away from the last node of the backbone. The mobile cannot be on a tail. In Figure 4a, the mobile unit moves from the last backbone MSC, C, to an MSC not in the graph, D. In this case, the backbone extends to include D by both the child pointer of C pointing to D and the parent pointer of D pointing to C. Child and parent pointers will be used primarily in the announcement delivery portion of the algorithm.

In Figure 4b, the mobile moves to a node already in the backbone, B. Because of B's position, we do not want to change any of the parent pointers. But we do wish to cause the deletion of the tail formed by MSCs C and D. Therefore a delete message is sent to C. When C receives the delete from its parent, it will delete its parent pointer, send the delete to its child, D, and nullify its child pointer. If at this point, however, the mobile moves from B onto D (See Figure 4c), D still has a parent pointer (C) and we cannot distinguish this case from the previous case (where B also had a parent pointer).

In the previous case we did not change the parent pointer, but in this case, we wish to have D's parent pointer set to B so that the backbone is maintained. To distinguish these two cases, we require the mobile unit to carry a copy of the identifiers of the nodes in the backbone. In the first case where the mobile arrives at B, B is in the list of backbone nodes maintained by the mobile, so B keeps its parent pointer unchanged, but prunes the backbone list to remove C and D. However, when the mobile arrives at D, only A and B are in the backbone list, therefore the parent pointer of D is changed to point to B.

But, what happens to the delete message heading for D from C? Because C is no longer D's parent, the delete is simply dropped and the backbone is not affected. Keeping the backbone list is similar to routing protocols passing complete paths to destination in BGP [11] to avoid loops. It has been argued that keeping such information in the packet greatly increases its size. However, in our case, the information is being kept by the mobile unit and we assume there is sufficient storage on such a device for this additional information.

The delivery algorithm is then superimposed on top of the generated graph. It is not sufficient to send the announcement down the spanning tree created by the backbone because the mobile is free to move from a region below the announcement to one above it. Therefore, to guarantee delivery, as the announcement propagates down the backbone, a copy is stored at each backbone node until delivery is complete. We refer to the portion of the backbone with an announcement as the *covered backbone*, see Figure 3b. Delivery can occur by the mobile unit moving to a location in the covered backbone, or the announcement catching up with the mobile at an MSC. In either case, an ack is generated and sent up via the parent pointers toward the root. If the announcement is delivered by the mobile moving on to the covered backbone, a delete is generated toward the child and an ack is generated toward the parent. Therefore any extra copies of the announcement on the newly created tail will be deleted with the nodes.

3.1 Details

The code for the tracking algorithm is shown in Figure 5. As before, we model arbitrary movement of the mobile by an action that allows a mobile at a node to move non-deterministically to any outgoing channel. It is important to realize that, in this model, the tree formed by the movement is superimposed onto the graph of nodes. The existence of a channel does not mean the channel already exists as part of the movement path, but rather simply that it is possible for a mobile to move from one cell to another.

MOBILEARRIVES shows the bulk of the processing and relates closely to Figure 4. When the mobile unit arrives at a node, we must decide how the backbone changes. If the mobile is doubling back onto the backbone, we leave the parent pointers alone and simply change the path carried by the mobile to reflect the shorter backbone (as in Figure 4b). If the node is not in the backbone (Figure 4a) or is part of a tail (Figure 4c), then the parent pointers must change to add this node to the backbone, and the node must be appended to the backbone list carried by the mobile. In any case, the children of this node are no longer necessary for announcement delivery, so a delete message is sent to the child and the child pointer is cleared.

If the announcement is at the node and the mobile has not already seen it, delivery occurs and the mobile records the message identifier to prevent future redelivery. Because delivery is complete, we send an ack message toward the root to clear the announcement copies. The node then deletes its copy of the announcement.

An announcement can arrive at a node either from its parent or from another node that is part of a tail. If the announcement arrives from a node other than the parent, it should be discarded. If a delete from a node other than the parent is processed, it could propagate through the network endlessly. However, when an announcement arrives from the parent it is always processed. If the mobile is present, the announcement is delivered and the ack propagation started. If the mobile is not present, the node stores a copy of the announcement in case the mobile arrives at a later time and immediately sends the announcement to its child.

ACKARRIVES enables the cleanup of the announcements by propagating acks toward the root via the parent pointers. The purpose of the delete messages is to remove the tail segments of the graph. Recall that a tail is created by a backbone node sending a delete to its child. Therefore, a delete should only arrive from a parent node. If we were to accept a delete from a non-parent node, as in the delete from C to D in Figure 4c, we could destroy the backbone. However if the delete arrives from the parent, we are assured that the node no longer resides on the backbone and should be deleted. Therefore, the arrival of a delete from a parent triggers the deletion of the stored announcement, the propagation of the delete to the child, and the clearing of both child and parent pointers.

```
State
AnnouncementAt$_A$    boolean, true if announcement stored at A; initially false everywhere
MobileAt$_A$          boolean, true if mobile unit at A; initially false except at root
Parent(A)             the parent of node A is node B; initially NULL
Child(A)              the child of node A is node B; initially NULL
started               boolean, true if delivery has started; initially false

Actions
ANNOUNCEMENTARRIVES$_A$(B)    ;arrival at A from B
  Effect:
    if Parent(A)=B
      if MobileAt$_A$
        if sequence number in mobile less than
             sequence number of this ann.
          deliver announcement
          mobile records sequence number
        send ack to B
      else
        AnnouncementAt$_A$:=TRUE  ;save ann.
        send announcement to Child(A)

ACKARRIVES$_A$(B)    ;arrival at A from B
  Effect:
    if AnnouncementAt$_A$
      AnnouncementAt$_A$:=FALSE  ;delete ann.
      send ack to Parent(A)

DELETEARRIVES$_A$(B)    ;arrival at A from B
  Effect:
    if Parent(A)=B
      if AnnouncementAt$_A$
        AnnouncementAt$_A$:=FALSE  ;delete announcement
      send delete to Child(A)
      Parent(A):=NULL
      Child(A):=NULL

MOBILEARRIVES$_A$(B)    ;arrival at A from B
  Effect:
    if A in mobile's list of nodes
      A keeps old parent
      mobile's list of nodes is truncated
           after A to the end
    else
      Parent(A):=B
      A added to mobile's list of nodes
    A sends delete to Child(A)
    Child(A):=NULL
    if AnnouncementAt$_A$
      if sequence number in mobile less than
           sequence number of this ann.
        deliver announcement
        mobile records sequence number
      Send ack to Parent(A)
      AnnouncementAt$_A$:=FALSE  ;delete ann

MOBILELEAVES$_A$(B)    ;leaves from A to B
  Preconditions:
    MobileAt$_A$ and channel (A,B) exists
  Effect:
    MobileAt$_A$:=FALSE
    Child(A):=B

ANNOUNCEMENTSTART    ;root sends announcement
  Preconditions:
    started = FALSE
  Effect:
    started:=TRUE
    if MobileAt$_{root}$=TRUE
      deliver announcement
    else
      AnnouncementAt$_{root}$:=TRUE
      root sends announcement to Child(root)
```

Figure 5: Tracking algorithm obtained using some initial ideas from termination detection

3.2 Correctness

The first node in the backbone is either the node on which the mobile is currently residing (if it is at a node) or the node it just left from (if it is in a channel). The remaining nodes are found by tracing parent pointers from the first node until we reach the root. We use the term *predelivery* to denote the execution interval during which delivery has been started but not accomplished. We present a proof sketch; a complete formal proof is more complex and has many more invariants. The following major invariants are useful:

I1: The backbone is well defined (i.e., it is acyclic and terminates at the root).

I2: During predelivery, there is exactly one announcement (that we call the *bellwether*)[1] on a link in the network. This link is either between two nodes on the backbone, or follows the mobile unit on the same link. (Intuitively, this is the leading edge of the announcement moving towards the mobile.)

I3: During predelivery, the backbone is divided into two sections. The nodes above the bellwether and the part of the link above the bellwether are called the covered backbone. The part of the link below and the nodes below form the rest of the backbone. During predelivery, nodes in the covered backbone have a copy of the announcement, and no other nodes in the graph have a copy.

I4: If A and B are backbone nodes and A is the parent of B, there is no delete message in transit from A to B. For all nodes A and B, if A is the parent of B, then either A records B as its child or there is a delete in transit from A to B.

I5: In predelivery, there are no acks in the system.

I6: After delivery, if a backbone node A has a copy of the announcement, then there must be an acyclic path from an ack to A by following child pointers in which each node has a copy of the announcement.

Eventual Announcement Delivery: We present a proof sketch that delivery must eventually occur. During predelivery, we know know that in any state, there is a bellwether announcement copy on the link from say X to Y, and a set of say n nodes in the covered backbone. We know that the bellwether must eventually arrive at node Y by link liveness.[2] We claim that after bellwether arrival either the announcement is delivered or the length of the covered backbone increases to $n + 1$. Clearly the length of the covered backbone cannot increase beyond the number of nodes in the graph (because the backbone is acyclic); thus delivery must eventually occur.

So how do we prove that after bellwether delivery, either the announcement is delivered, or the length of the covered backbone increases? Essentially, this follows because when the bellwether reaches Y, Y must become part of the backbone (thereby increasing the length of the covered backbone) unless the mobile has moved to some node say Z in the backbone above the bellwether (but in this case, delivery must have occurred, because by the invariants Z has a copy of the announcement).

[1]Webster's dictionary defines the bellwether as the leading sheep of a flock.

[2]We must assume that a mobile eventually arrives at the next MSC or delivery can be indefinitely postponed.

Tail Deletion and Garbage Collection: Invariant **I4** ensures that any tail has a delete message in transit to the node at the head of the tail (where the head of the tail is defined as the node whose parent no longer points to it as a child). Thus all tails will either be deleted or the nodes in the tail will be added back to the backbone by mobile movement. Thus nodes that the mobile once visited but are no longer part of the backbone will eventually clean up their parent pointers.

It is also important to show that after delivery, all announcement copies are either deleted by a node receiving a delete message or by an ack. Because acks follow parent pointers toward the root, the only way for the ack not to reach one of these nodes is for the ack to stop propagating toward an announcement copy. The only way for an ack to stop propagating is for it to reach a node without an announcement copy or for the mobile's arrival to change the backbone and destroy the acks. In the case of the mobile arriving on the backbone, a new ack is generated up the backbone, taking the place of the original ack farther down in the old backbone. The former case can occur if the ack arrives from a tail node. By using Invariant **I6**, however, we argue that there is always an ack en route to each announcement copy in the backbone even if an extra ack arrives from a tail. Because of the tail deletion property, announcements on tail nodes will eventually be deleted.

Generalizations: A simple extension of this algorithm is to allow for multiple concurrent announcement deliveries as in sliding window protocols. These announcements and all associated acks would have to be marked by sequence numbers so that they do not interfere, but the delivery mechanism uses the same graph. Therefore the rules governing the expansion and shrinking of the graph are not affected but the proofs of garbage collection and ack delivery are more delicate.

4 Message Delivery through Search

In a search approach, the general idea is to send the announcement to all nodes, and have one of these nodes actually deliver the announcement. A simple scheme to disseminate the announcement might be to construct a spanning tree over the nodes and pass the announcement along these links. For example, in Figure 1b, an announcement can be initiated at MSC_2 and sent in both directions along the spanning tree shown in bold.

If the mobile unit does not move during the announcement dissemination, it will receive the announcement. However, if it crosses the border between MSC_1 and MSC_2 after the broadcast at MSC_2 but before the broadcast at MSC_1, it will not receive the announcement even though it was in the system for the duration of the broadcast and not moving particularly fast. On the other hand, in a snapshot the location of every message and the state of every processor is recorded. If a snapshot behaved in a manner similar to the naive broadcast just described, instead of recording all messages in the system, it could fail to record a message on a channel.

We know, however, that it is possible to construct a distributed snapshot such that all messages are recorded. For this paper, we use the Chandy-Lamport [4] algorithm; however, many other snapshot algorithms (including non-FIFO algorithms) can be used as well. When running this snapshot in a system with a uniquely identified mobile unit, there is never more than one copy of this mobile in the system at a given time. Thus any system state (and hence any consistent snapshot state) will record the mobile exactly once. Intuitively, the point at which a snapshot records the mobile must involve some contact between the snapshot control message and the mobile. By modifying the snapshot marker messages to piggyback the announcement, we obtain *exactly once* delivery. In [9], this search algorithm is presented and its properties reasoned about directly from the program text. In this paper, however, we directly explore the relationship between the Chandy-Lamport algorithm and the resulting message delivery algorithm. Our new proof provides a simpler explanation and proof using a reduction to a snapshot and its properties.

In the Chandy-Lamport algorithm, a process begins its local snapshot when it receives the first marker. When this occurs, the marker is sent on all other outgoing channels and the state of the processor is recorded. If there are any messages at the node, they are recorded as part of the processor state (Figure 6a). If the node has already started its local snapshot when a message arrives along a channel (that the node has not seen the marker on), the message is recorded as being on the channel (Figure 6b). Recording continues until a marker is received on all incident links.

We translate these actions to the mobile environment. The announcement corresponds to the marker, and the mobile node corresponds to a message in the Chandy-Lamport algorithm. When an MSC receives the announcement for the first time, it sends copies on all outgoing channels and attempts delivery to any mobile unit present. If the mobile unit is at the MSC, it will receive the announcement (Figure 6c).

Just as a node continues recording until it has received the marker on all links, the delivery algorithm *will keep a copy of the announcement until it receives a copy of the announcement from all neighbors.* Intuitively, this is what prevents the mobile from hopping from node to node (like the Artful Dodger) eluding the announcement. Thus if the mobile arrives prior to the announcement on a channel, the MSC delivers the data as soon as the handover is complete (Figure 6d) because it has a stored local copy.

Note that each node keeps a copy of the announcement for only a small *local propagation delay* (i.e., the time to send and receive a message to all neighbors). This is more efficient than another proposal [1] which broadcasts the announcement to all nodes. Each node then stores the announcement until it either delivers the announcement or receives notice that another node has delivered it. Such a scheme requires storing copies of an announcement for *a network propagation delay*, and requires at least two phases (phase 1 for broadcasting the announcement, and phase 2 for the node that accomplishes delivery to send delivery notification to all other nodes and clean up the distributed announcement copies).

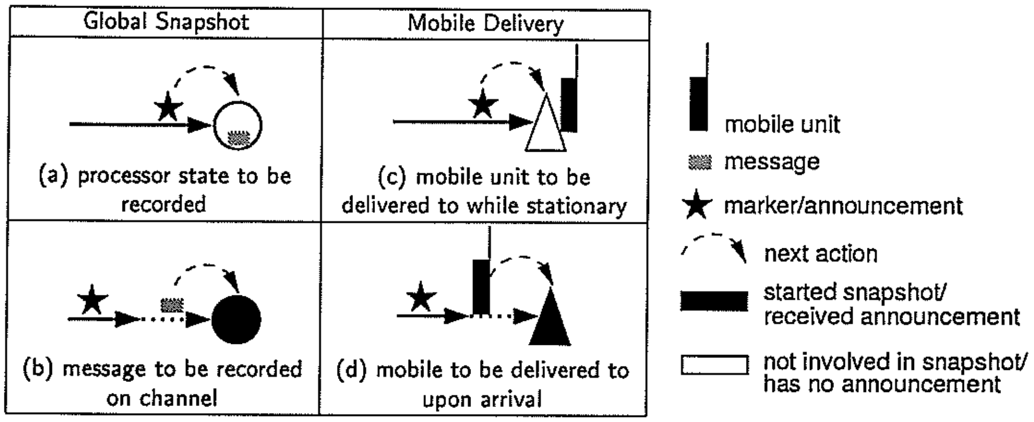

Global Snapshot
Mobile Delivery
(a) processor state to be recorded
(c) mobile unit to be delivered to while stationary
(b) message to be recorded on channel
(d) mobile to be delivered to upon arrival
mobile unit
message
marker/announcement
next action
started snapshot/ received announcement
not involved in snapshot/ has no announcement

Figure 6: Translation of concepts from global snapshots into mobile delivery. The curved arrow shows the processing of an element from a channel while the text describes the action triggered by such movement.

4.1 Details

For simplicity, the code considers only a single announcement being delivered and a single mobile unit in the system.

In order to determine whether or not to deliver the data to a mobile when it arrives from channel c, we must know if the announcement has arrived from c. Therefore, we define a flag, *flushed*, that indicates whether the data has been received via c. Initially, we set all *flushed* flags to false. To start the algorithm, we give an announcement to a single node, causing it to start its local snapshot and send the announcement on all outgoing channels. Because we are artificially placing the announcement at the node, we do not change the state of any of the flushed variables.

```
State
flushed_{A,B}          boolean, true if announcement traversed the link from A to B; initially false everywhere
AnnouncementAt_A       boolean, true if announcement stored at A; initially true only where ann. starts
MobileAt_A             boolean, true if mobile unit at A; initially true only where mobile located
Actions
ANNOUNCEMENTARRIVES_A(B)   ;arrival at A from B
    Effect:
        flushed_{B,A} := TRUE
        if ¬AnnouncementAt_A
            send announcement on all outgoing channels
            AnnouncementAt_A := TRUE   ;save ann.
            if MobileAt_A
                deliver announcement

MOBILEARRIVES_A(B)   ;arrival at A from B
    Effect:
        MobileAt_A := TRUE
        if ¬flushed_{B,A} and AnnouncementAt_A
            deliver announcement

MOBILELEAVES_A(B)   ;leaves from A to B
    Preconditions:
        MobileAt_A and channel (A,B) exists
    Effect:
        MobileAt_A := FALSE
        mobile unit moves onto (A,B)

CLEANUP_A   ;A finishes local snapshot
    Preconditions:
        Forall neighbors X, flushed_{X,A} = TRUE
    Effect:
        AnnouncementAt_A := FALSE   ;delete ann.
        Forall neighbors X, flushed_{X,A} := FALSE
```

Figure 7: Snapshot Delivery Code

In Figure 7, ANNOUNCEMENTARRIVES, we indicate that the announcement has arrived by setting the *flushed* variable for that channel. If the announcement had already arrived, we do not need to do any further processing. If we attempted to deliver the announcement again, we might deliver it twice to the same mobile. Additionally, if the announcement has already been propagated on all outgoing channels, we do not want to send it again because this could restart the delivery process unintentionally. Therefore, if the MSC was not already holding a copy of the announcement, we attempt delivery (as in Figure 6c) and store a copy of the announcement. We also propagate the announcement on all outgoing channels.

When the mobile unit arrives from a channel, MOBILEARRIVES Figure 7, we must either attempt to deliver the data or do nothing. If the MSC does not have a copy of the announcement, then trivially it cannot deliver it. However, if it does have the announcement and the channel has already been flushed, attempted delivery would be a duplication. Therefore, if the channel has not been flushed and the announcement is present we deliver the data (as in Figure 6d).

CLEANUP models the local completion of the delivery. Once the announcement has arrived on all channels, the local snapshot is complete. The node knows that either it already delivered the announcement or it never will. Therefore, the announcement can be deleted and the channel states reset to not flushed.

4.2 Correctness

To formally argue correctness, we start with an automaton that implements the Chandy-Lamport (CL) algorithm on a graph on which the mobile node is the only "message" that can travel from node to node. We then transform CL into our message delivery (MD) algorithm using the rules above. We then prove that for any execution of MD there is a corresponding execution of CL with the following correspondence invariants that can be established inductively.

I1: message arrival in CL corresponds to mobile arrival in MD.

I2: marker arrival in CL corresponds to announcement arrival in MD.

I3: recording the message in CL corresponds to delivering the announcement to the mobile in MD.

I4: being in the state of recording a local snapshot in CL corresponds to having a copy of the announcement in MD. Completing the local snapshot in CL corresponds to deleting the announcement copy in MD.

Next, we know that any execution sequence in CL will record the mobile exactly once. If the mobile is recorded at node X when the marker arrives at node X, then the correspondences show that delivery occurs at the corresponding point in the execution of MD. Similarly, if the mobile is recorded as being on incident link L of node X, we know from the correspondence that in the corresponding execution of MD, the mobile will also arrive on link L and the flushed flag for link L will be false in both automata. From the correspondence, we know that node X will still have a local copy of the announcement in MD. Thus, from the code of MD we know that delivery occurs. We also know that it cannot occur again; if so, CL would record the mobile twice. Finally, since CL eventually completes all of its local snapshots, MD eventually deletes all announcement copies.

4.3 Reality Check

In moving from the cellular domain into the mobile domain, we made five assumptions. We show how to remove or enforce these assumptions.

First, we assumed that the channels are FIFO and that both announcements and mobile units move along these channels. As shown in Section 2, it is possible to extend the handover protocol to enforce FIFO channels. Second, we assumed that all neighboring MSCs are directly connected by a wired channel. For cost and feasibility purposes, this is often not the case; however we can simply add virtual channels between any two MSCs (that a mobile can transit between and are yet not directly connected by a wire), and imposing the same FIFO constraints on these channels.

Third, snapshot delivery assumes that link delivery is reliable. However most of the Internet uses unreliable links like Ethernets, frame relay, and ATM. Although the probability of error is low, packets are indeed dropped. A possible solution is to add acks for announcement messages (as is done, for example, in the intelligent flooding algorithm used by Link State Routing in OSI [12] and OSPF [8]). Another solution is only to provide best-effort service. Since lost announcements can lead to deadlock we need to delete an announcement after a timeout even if it is still expected along a channel.

Fourth, we assumed that all MSCs in the network are involved in each delivery. It is impractical to assume that all nodes in the Internet will participate in a snapshot to locate a single mobile. However it is not unreasonable to assume that all nodes in a clearly defined region such as a city, campus, or building would participate. Movement between these regions would be achieved by an additional algorithm.

Finally, we assume that the MSCs are willing to hold the announcement for the duration of their local snapshot. In a system with bi-directional channels, a local snapshot can be as short as a single round trip delay between the MSCs. One can argue that it is not the place of the MSCs to be maintaining copies of the announcement when their primary purpose is routing; however, in this case, because no routing information is being kept about the mobile units, the additional burden is offset.

4.4 Extensions

It is trivial to generalize our scheme to multicast an announcement to a set of rapidly moving mobiles, even if the actual addresses of the nodes in this set are unknown. This allows mobiles to join and leave the multicast group at will, as in the IP multicast model. Prior work in Mobile IP requires re-registration on each mobile movement, while the results in [1] require the set of recipients to be known. Intuitively, multicast follows because the snapshot will record every mobile in the set exactly once. A mobile contains a list of multicast addresses that it subscribes to, and an announcement carries a multicast address it is sent to. When a mobile arrives at a MSC, its multicast set is checked against the set of addresses of stored announcements; if there is a match, delivery is done. Similarly, when an announcement to address G arrives at a MSC, it is delivered to all mobile nodes currently being handled by the MSC that subscribe to G. The algorithm can provide *exactly once* (if the announcement is sent reliably between MSCs) or *at most once* delivery semantics.

In some systems mobile movement is not rapid, and it is not unusual for two mobile units to remain fixed while exchanging a series of messages. In this case, we might want to derive a route from one mobile unit to the other and have all announcements travel this route. One way to discover such a route is to use the snapshot algorithm, and have the initial control message maintain the route that it takes to get to the mobile unit. When the unit is located, it responds along this route to the other mobile unit. Now both units have a fixed route for all communication and we do not require all nodes to be involved in every message delivery. If at any point this route fails, another can be established by running the snapshot for route discovery again.

5 Previous Work

[3] describes issues (e.g., disconnection, low bandwidth, and low computational power) and techniques that arise in converting distributed algorithms to the mobility domain while retaining their original properties (e.g., running a distributed snapshot to get the state of a group of mobile units). However, our work is fundamentally different because we use the distributed algorithms as a tool to achieve a different result (namely tracking and search) than that of the original distributed algorithm.

[1] describes a centralized, reliable multicast protocol for mobile networks. This approach requires the sender to specify all mobiles in the recipient list, and for all nodes in the system to hold a copy of the announcement until delivery is complete. Although this model makes guarantees even in the presence of disconnection, in most Internet multicast protocols, such as DVMRP and PIM [7], the participant list is not known by the senders. This makes such a scheme hard to implement. However, in our snapshot delivery multicast, recipient lists are not necessary. Further, our algorithm is more efficient because it stores an announcement for at most a link round trip delay, and all decisions are taken locally.

[6] provides a UNITY-style specification for a tracking algorithm that eventually converges to the shortest path from home to mobile. The convergence can be slow, and no guarantees are made that delivery will occur in a system with rapidly moving mobiles.

Similarly, Mobile IP [10] is a practical protocol being developed and tested. It has extensions to work in a cellular-like setting. However, in the case of rapid movement, the registration mechanism will not be able to complete and announcements will be dropped. There are also extensions to multicast but these are subject to the same restrictions.

Most related work relates primarily to the physical movement of mobile components and does not take into consideration mobile code. Each of our algorithms is directly applicable to the mobile code environment with proper network support.

A UNITY-style algorithm and proof of the second message delivery algorithm has been presented in [9]. This is a proof that does not exploit the relation to snapshots at all, and argues directly about message delivery properties. The I/O automata proof using reductions presented in this paper is new, and provides a clearer and simpler intuition of the problem approach. Additionally, the reality check is entirely new to this paper. Finally, the first tracking algorithm (the major result of this paper) and the FIFO channel extensions have never been published before.

6 Conclusions

We have presented two simple, practical, and efficient algorithms for message delivery to rapidly moving mobiles. While solutions like Mobile IP or cellular phone homing (which work only for slow movement) may be adequate today, it is recognized that message delivery to rapidly moving mobiles is a problem that will become increasingly important in the future, not only for physical mobility but also for logical mobility, as exemplified by mobile code. Besides, the problem of delivering a message to a mobile that moves at arbitrary speeds seems a fundamental problem, worthy of theoretical study.

Our first algorithm has good locality properties when used on a large network and requires only a few pointers per mobile unit. On the other hand, it assumes a home node and takes longer to garbage collect announcement copies stored on backbone nodes. Our second algorithm is simple, requires no overhead when no announcements have to be delivered, rapidly garbage collects stored announcements, and generalizes to multicasting. On the other hand, it does send a copy of the announcement on every link in the graph. Thus it seems more applicable for announcements within a small region of MSCs. Within limited domains, both algorithms or combinations thereof, appear practical and useful because of their simplicity and their guarantees of message delivery. A major open problem in to identify an algorithm that does not require broadcast to all nodes *and* provides rapid garbage collection (or to prove that there is no such algorithm).

Besides the specific delivery algorithms, perhaps the most important contribution of this paper is a new approach for developing algorithms for the mobile environment by treating mobile units as messages. To make such a model work in practice may require modifying basic mobility mechanisms (e.g., modifying handover to enforce FIFO behavior) and may require adding further ideas (as in our tracking algorithm which uses Dijkstra-Scholten as a point of departure, but has a very different final form). However, the benefits are considerable: we can leverage off scores of existing results in distributed computing, and gain valuable insight into algorithm development.

References

[1] A. Acharya and B.R. Badrinath. A framework for delivering multicast messages in networks with mobile hosts. *Journal of Special Topics in Mobile Networks and Applications (MONET)*, 1(2):199–219, October

1996.

[2] M. Ahuja. *Flush* primitives for asynchronous distributed systems. *Information Processing Letters*, 34(1):5–12, 22 February 1990.

[3] B.R. Badrinath, A. Acharya, and T. Imielinski. Impact of mobility on distributed computations. *ACM Operating Systems Review*, 27(2):15–20, 1993.

[4] K.M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computing Systems*, 3(1):63–75, 1985.

[5] E.W. Dijkstra and C. Scholten. Termination detection for diffusing computations. *Information Processing Letters*, 11(1), 1980.

[6] S. Kryukova, B. Massingill, and B. Sanders. Specification and proof of an algorithm for location management for mobile communication devices. In *Proceedings of the International Workshop on Formal Methods for Parallel Programming: Theory and Applications, IIPS '97*, Geneva, April 1997.

[7] T. Maufer and C. Semeria. Introduction to IP multicast routing. Internet Draft, July 1997. ftp://ds.internic.net/internet-drafts/draft-ietf-mboned-intro-multicast-03.txt.

[8] J. Moy. *OSPF Version 2*. Internet Engineering Task Force, 1994.

[9] A.L. Murphy, G.-C. Roman, and G. Varghese. An exercise in formal reasoning about mobile communications. In *Proceedings of the Ninth International Workshop on Software Specification and Design*, pages 25–33, Ise-Shima, Japan, April 1998. IEEE Computer Socitey Technical Council on Software Engineering, IEEE Computer Society.

[10] C. Perkins. IP mobility support. RFC 2002, IETF Network Working Group, 1996.

[11] Y. Rekhter and T. LiC. Perkins. A Border Gateway Protocol 4 (BGP-4). RFC 1771, IETF Network Working Group, March 1995.

[12] M. Steenstrup. *Routing in Communication Networks*, chapter 5. Prentice-Hall, 1995.

[13] M. Steenstrup. *Routing in Communication Networks*, chapter 10. Prentice-Hall, 1995.