Washington
WASHINGTON • UNIVERSITY • IN • ST • LOUIS
School of Engineering & Applied Science

An Algorithm for Message Delivery in a Micromobility Environment

Amy L. Murphy
Gruia-Catalin Roman
George Varghese

WUCS-97-22

28 April 1997

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

An Algorithm for Message Delivery in A Micromobility Environment

Amy L. Murphy, Gruia-Catalin Roman, and George Varghese
Washington University, St. Louis

Abstract

With recent advances in wireless communication and the ubiquity of laptops, mobile computing has become an important research area. An essential problem in mobile computing is the delivery of a message from a source to either a single mobile node, *unicast*, or to a group of mobile nodes, *multicast*. Standard solutions proposed for macromobility (Mobile IP) and micromobility (cellular phones) for the unicast problem rely on *tracking* the mobile node. Tracking solutions scale badly when mobile nodes move frequently, and do not generalize well to multicast delivery. Our paper proposes a new message delivery algorithm for micromobility based on a modification of classical snapshot algorithms. Our algorithm requires no tracking, provides stronger guarantees than existing protocols in micromobility, and generalizes easily to multicasting. Besides a particular solution to the delivery problem, our approach offers a new strategy for transferring established results from distributed computing to mobile computing. The general idea is to treat mobile nodes as messages that roam across the fixed network structure and to leverage off existing distributed algorithms that compute information about messages.

1. Introduction

Mobile computing reflects a prevailing societal and technological trend towards ubiquitous access to computational and communication resources. Wireless technology and the decreasing size of computer components allow users to travel within the office building, from office to home, and around the country with the computer at their side. These types of movement lead to two approaches to the issues of mobility based on the spatial and time granularity of movement. The literature has referred to these styles as micromobility and macromobility.

In macromobility, the user moves slowly among large networks. This is analogous to a user taking their laptop computer from a local network at work to a local network at home. It is assumed that this movement is relatively infrequent, and often involves complete disconnection from network resources for a length of time. Micromobility models the movement of a user within a single network, or a tightly clustered group of networks. This is similar to the mobile telephone model in which a user participating in a phone conversation moves among support centers without losing the connection. In micromobility, the user is allowed to move any number of times, but the rate of movement must be slow enough to allow for the handover to complete. Otherwise, the telephone conversation will be lost.

Both disconnected operation and connected operation are goals for systems with mobile components. In disconnected operation [7], rules must be devised for handling disconnection, allowing changes while disconnected, and reintegration of changes upon reattachment. For connected operation [12], [8], [9], there must be a mechanism to get messages to the mobile components while they are connected. There are two main approaches to solving this problem: tracking and search.

In tracking, the location of the mobile unit is known at all times, and therefore a route to the unit exists. In Mobile IP, this is accomplished by the mobile unit registering its new location with a home agent, and having the home agent forward any messages for that mobile unit to the new location. Although Mobile IP is well accepted among the proponents of the Internet partially due to its incremental deployment, other approaches propose changing the routers to adapt to the movement of the mobile units and effectively intercept packets en route to the home agent and direct them toward the mobile unit itself [11]. Because these approaches involve fundamental changes to the routers, this is more popular as a solution within corporations developing the technology.

---

[1] The work of the first two authors is supported in part by the National Science Foundation under Grant No. CCR-9217751. The work of the third author was supported in part by an ONR Young Investigator Award. 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 or the Office of Naval Research.

In search solutions, the location of the mobile unit is not kept anywhere in the system. Therefore, in order to get a message to the mobile unit, the sender must either broadcast a search request to locate the mobile unit, then forward the message along the same path that the search message took to that location, or the sender can simply broadcast a copy of the message. The first mechanism has been suggested for ad hoc mobility [6] where there is no infrastructure to route packets along. This approach takes advantage of the natural broadcasting nature of mobile communication such as radio to execute a broadcasting protocol to any neighboring mobile units within range. This kind of route discovery is useful in moderately changing environments where a route to a mobile unit is viable long enough for both route discovery and message delivery.

Another problem being addressed in the mobility community is that of multicast. In the context of Mobile IP and macromobility, two possible approaches are available. The mobile unit can register to receive multicast packets through its home agent, in which case, the home agent must encapsulate every multicast message and unicast it to the foreign location of the mobile unit. Alternately, the mobile unit can join the multicast group at a local multicast router on the subnet it is visiting. This assumes the local subnet supports multicast. Because many multicast protocols rely on the address of the sender to properly forward messages along a multicast tree, corresponding changes must be made in order for the mobile unit to send to a multicast group address.

This approach works for macromobility, however our focus is micromobility. Work by Acharya and Badrinath [1] describes an algorithm for reliable multicast which takes a centralized control approach, keeps messages alive at all mobile support centers until termination, and requires the sender to know all the recipients of the multicast. In many cases throughout the Internet, reliability is not necessary, and the source of a multicast message either does not want to know or cannot know the recipients of the multicast. It is generally accepted that if the number of recipients is large, the sender does not want to sacrifice the resources to maintain such a large and dynamic list. This is one of the assumptions in DVMRP and other standard multicast protocols. After presenting our algorithm, we will return to the issue of multicast and show how our approach generalizes to multicast.

In general, our work focuses on message delivery in a micromobility environment using a form of search through broadcast. We assume a fixed structure similar to that of the mobile telephone system with fixed base stations broadcasting messages to mobile units, mobile support centers coordinating the base stations, and mobile units moving within a region of space within range of the base stations. Our approach adapts the Chandy and Lamport algorithm for global snapshots in a distributed system to message delivery in a mobile environment. In the next section, we show that simple broadcasting to mobile units can fail and we present our approach using the UNITY notation. In section 3, we present the advantages we gain from using the established work of the global snapshot and provide a proof outline based on the program from the previous section. Section 4 is a reality check in which we examine our approach in detail identifying some of its potential shortcomings and our solutions. In section 5 we show possible extensions to our approach including route discovery and multicast, and section 6 provides some conclusions and future directions.

2. Problem Motivation

The problem we are interested in is the delivery of messages to a mobile unit in a network of fixed mobile support centers (MSCs) and radio base stations (RBSs). For simplicity, we will assume each MSC controls only one RBS, and all MSCs whose radio base stations are neighbors have a fixed communication channel between them. For example, in Figure 1, each cell represents a single (MSC, RBS) pair, and the MSCs of all neighboring cells are connected by a fixed network. A mobile unit can send and receive messages from only one RBS at a time, and only when it is in the cell associated with that RBS.

A common broadcasting scheme in this setting is to construct a spanning tree over the MSCs and send the message along this tree. In Figure 1, a spanning tree is indicated by the solid lines. For this example, suppose a mobile node is located at cell 1, near the border of cell 2. Suppose cell 2 initiates the message to be delivered to the mobile unit. Following the proposed spanning tree broadcast, MSC_2 has RBS_2 broadcast the message locally, then forwards the message on the two outgoing links of the spanning tree. After sending the message successfully, MSC_2 deletes its copy of the message. The MSCs down stream behave in a similar manner, broadcasting locally, forwarding the message to their children, and finally deleting their copy of the message. If the mobile unit does not move out of cell 1, it will receive a copy of the message when it is broadcast by RBS_1. However, because the mobile unit is on the border between cell 1 and cell 2, it is possible for a handover to be initiated and for the mobile unit to lose contact with RBS_1 and pick up communication with RBS_2. If this handover occurs after RBS_2 deletes its copy of the message and before RBS_1 broadcasts its copy of the message, the mobile unit will not receive the message even though it was connected to the network during the entire broadcast lifetime of the message. Although in reality messages travel through the network much faster than a mobile unit can travel through space, because a basic handover requires very little time to complete, and the length of the path along the spanning tree for the two cells m is moving between could be longer to traverse than for the handover to complete, it is reasonable for a simple broadcast mechanism such as this to fail in this kind of circumstance.

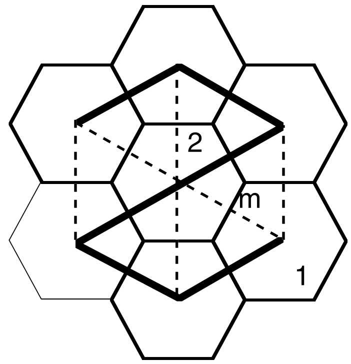

Figure 1. Cell based broadcast.

Therefore, we propose an alternative broadcast algorithm which guarantees the mobile agent in the above circumstance will receive a copy of the message. Our solution is based on the classical notion of a snapshot from distributed computing, in particular, our algorithm is based on the original Chandy-Lamport snapshot algorithm [3].

Before moving on to our solution to message delivery, we must digress into some of the details of the general snapshot algorithm. A snapshot algorithm is typically executed over a graph consisting of processors and communication channels connecting these processors. The goal of the snapshot is to record a consistent state which is possible in an execution of the running computation. Nodes communicate via message passing along the channels. The state of the system consists of the values of the registers and memory at the processors and any messages in transit between the nodes. For example, a simple snapshot algorithm can freeze computation at all processors, record the state of the processors and the location of all messages, then restart the computation. This is impractical in most distributed settings, but it gives the intuitive notion of the results of a snapshot algorithm.

Although snapshot algorithms were developed to detect stable properties such as termination or deadlock by creating and analyzing a consistent view of the distributed state, minor adjustments allow them to perform announcement delivery in the dynamic, mobile environment. A traditional global snapshot is a collection of the local snapshots for the individual processing nodes consisting of the local state plus all messages in transit to that node. In general, distributed snapshot algorithms proceed across a network by passing the knowledge that the snapshot is occurring to neighboring nodes via control messages. Every message appears exactly once in the global snapshot either on a channel or at a node.

To translate this into the mobile setting, the nodes in a distributed computation directly correspond to the mobile support centers and radio base stations of mobility. The messages recorded by the global snapshot correspond to the mobile units. Finally, the messages being delivered to the mobile units are carried on the snapshot control messages. Therefore, in the mobile setting, a local snapshot contains information about the mobile units in current communication with the base station, as well as the mobile units in transit to that node. Every mobile unit appears exactly once either at a node or on a channel. Because of the way various snapshot algorithms work, and the mobility inherent in the system, by the time the snapshot is analyzed to locate the destination mobile unit, any message sent directly to that location may miss the mobile unit. For this reason, the computation of the snapshot itself must be modified to perform message delivery. Specifically, we note that because each mobile unit must appear only once in the global snapshot, and the message can only be delivered once, the base station that records the destination mobile unit is responsible for delivering the message. More simply stated, recording a mobile unit is equivalent to message delivery.

Throughout this paper, we have chosen the Chandy-Lamport algorithm for global snapshots. In the environment for this algorithm, FIFO channels are assumed between all processing nodes. For the remainder of this section we will assume a single directional, FIFO channels along which both messages and mobile units move. In section 4, however, we will relax this model by proposing reasonable additions to the handover protocol to allow reality to model this FIFO channel. Further, mobile movement is characterized by a mobile unit moving into a channel. Also for the purposes of explanation we will eliminate the RBS from the model, but again, in section 4 we will bring it back with slight modifications.

```
Program    SnapDeliver
declare
    ChanQ : array [Channels] of Queue[mobile ∪ message];
    MobileAtMSC,  MessageAtMSC : array[MSCs] of Boolean;
    InChanFlush : array[Channels] of Boolean;
    notified : array [MSCs] of Boolean;
    found : Boolean;
    flushed : array[Channels] of Boolean;
    location : MSCs ∪ Channels;
initially
    ⟨[]  x,y : (x,y) ∈ Channels ::        ;x0 is the initial location of the message
         MessageAtMSC(x), notified(x), MobileAtMSC(x) = (x=x0), (x=x0), (location=x)
         []   InChanFlush(x,y), flushed(x,y) = false, false
         []   ;if the mobile unit is at the initial location of the message, delivery is complete
              found = (location=x)
         []   ;set up initial channel state based on mobile unit location and message location
              ChanQ(x,y) = enqueue(ε,mobile)                          if location=(x,y) ∧ x≠x0
              || ChanQ(x,y) = enqueue(enqueue(ε,mobile),message)      if location=(x,y) ∧ x=x0
              || ChanQ(x,y) = enqueue(ε,message)                      if location≠(x,y) ∧ x=x0
              || ChanQ(x,y) = ε                                       if location≠(x,y) ∧ x≠x0
    ⟩
assign
    ⟨[]  x,y : (x,y) ∈ Channels ::
         ;(Stmt 1) Duplicate message arrives at a MSC.
         InChanFlush(x,y), flushed(x,y), ChanQ(x,y) := true, true, tail.ChanQ(x,y)
              if head.ChanQ(x,y)=message ∧ MessageAtMSC(y)
         []
         ;(Stmt 2) New message propagates on all outgoing channels.
         ⟨ || : head.ChanQ(x,y)=message ∧ ¬MessageAtMSC(y) ::
                   found = true if MobileAtMSC(y)
                   ||   MessageAtMSC(y), InChanFlush(x,y), flushed(x,y) := true, true, true
                   ||   ⟨ || u : (y,u) ∈ Channels :: ChanQ(y,u) := enqueue(ChanQ(y,u),message) ⟩
                   ||   ChanQ(x,y), notified(y)  := tail.ChanQ(x,y), true⟩
         []
         ;(Stmt 3) Mobile unit arrives at a MSC with no delivery.
         location, ChanQ(x,y), MobileAtMSC(y) := y, tail.ChanQ(x,y), true
              if head.ChanQ(x,y)=mobile ∧ (InChanFlush(x,y) ∨ ¬MessageAtMSC(y))
         []
         ;(Stmt 4) Mobile unit arrives at a MSC with delivery
         found, location, ChanQ(x,y), MobileAtMSC(y) := true, y, tail.ChanQ(x,y), true
              if head.ChanQ(x,y)=mobile ∧ ¬InChanFlush(x,y) ∧ MessageAtMSC(y)
         []
         ;(Stmt 5) Mobile unit moves from a MSC to a channel
         location, ChanQ(x,y), MobileAtMSC(x) := (x,y), enqueue(ChanQ(x,y), mobile), false
              if MobileAtMSC(x)
         []
         ;(Stmt 6) Local snapshot complete at a MSC and all traces of message removed
         ⟨ || : ⟨∀ u : (u,y) ∈ Channels :: InChanFlush(u,y)⟩ ::
                   MessageAtMSC(y) := false
                   ||   ⟨ || u : (u,y) ∈ Channels :: InChanFlush(u,y) := false ⟩  ⟩
    ⟩
end
```

Figure 2. Snapshot Delivery Program for FIFO Channels

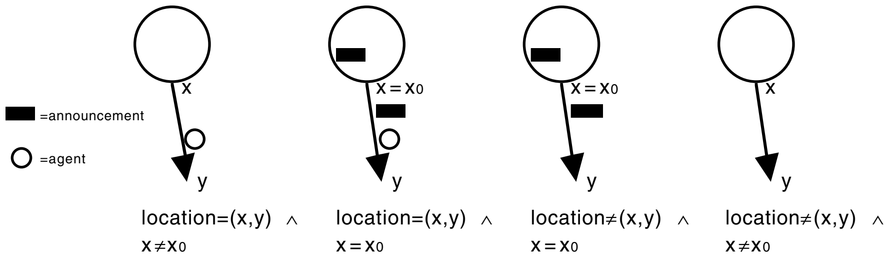

Figure 3. All possible initializations of a channel(x,y) with the message location x_0.

Because we are modeling message delivery, we assume the snapshot is initiated at the original location of the message, a single MSC. In general, snapshot algorithms proceed across a network by passing the knowledge that the snapshot is occurring to neighboring MSCs. We append to this knowledge the actual message. When the message arrives at a MSC, it is stored and the local snapshot begins. As the snapshot proceeds, the MSC watches for the destination mobile unit to arrive. If the local snapshot completes without recording the destination mobile unit as part of its state, the MSC deletes its copy of the message, assured that the destination mobile unit will appear in some other MSC's local snapshot. However, if the destination MSC is part of the snapshot, either in communication or on an incoming channel, this MSC is responsible for delivery. Once the message has been delivered, it can be deleted from the MSC. In this manner, as the local snapshots complete, the message copies are removed from the MSCs, meeting the requirement that no copies remain in the system.

We capture the essence of our solution in the form of a UNITY program shown in Figure 2. UNITY [4] provides a state transition model for concurrent programming in which progress and safety properties can be cleanly defined and proven from the program text. We utilize the UNITY proof logic in the next section where we formally verify the correctness of the algorithm. In the UNITY program, each channel is defined as an element of an array, *ChanQ*, of message queues. (See the **declare** section consisting of Pascal-like declarations.) The queues are restricted to holding only messages and mobile units. Actually, since we are interested in the behavior of the algorithm solely with respect to a particular mobile unit and one message assumed to be destined for it, these are the only two kinds of messages that can appear anywhere in the system, i.e., the channel queue can contain only the constants *message* and *mobile*. The state of the individual MSCs is captured by three arrays: *MobileAtMSC* is indexed by MSC identifiers and holds a boolean value which is true whenever the mobile unit is present at that MSC and false otherwise; *MessageAtMSC* is defined in the same manner but denotes the fact that the MSC is holding a copy of the message; finally, *InChanFlush*, indexed by channels, allows a MSC to keep track of which channels have been flushed. Basically, *InChanFlush* will be used to indicate that a local snapshot of the channel is complete.

In addition to these algorithm variables, we include in the **declare** section several auxiliary variables. For easy discrimination, they appear in the `courier` font and in lower case. These variables do not affect the execution of the program, but are convenient to have when constructing the proofs. The array `notified`, indexed by MSC identifiers, records the fact that a MSC has seen the message. The variable `found` becomes true when the message is delivered to the mobile unit, while `location` records the identity of the last channel or MSC to receive the mobile unit.

Constraints on the range of acceptable initial values for the program variables are stated by a set of equations appearing in the **initially** section of the program. Most initial values are straightforward. We place the message at an arbitrary MSC, x_0. This MSC is marked as holding the message, i.e., *MessageAtMSC* is true for MSC x_0 and in no other place; `notified` is initialized accordingly. The contents of the channels are initialized according to the location of the mobile unit (reflected by the arbitrary initial value for `location`) and x_0, as shown in Figure 3. If the mobile unit is at the MSC x_0, i.e., the mobile unit is at the location where the message was dropped off, the message is delivered to start with (`found` is initialized to true).

The algorithm appears in the **assign** section of the program. It consists of a set of multiple conditional assignment statements which are selected for execution one at a time subject to a weak fairness assumption. All executions are assumed to be infinite with finite executions being characterized by the fact that they reach a fixed point. Before continuing our discussions of the algorithm we need to take a brief detour and explain the notation we use. Individual statements are written either in terms of matching lists of variables and expressions subject to the same condition as in

x, y := y, x **if** x < y

which sorts the values of x and y or by using the parallel bar construction as in the equivalent formulation

x := y **if** x < y || y := x **if** x < y

The box symbol is used as a separator between assignment statements. The only unusual notation is a very general three-part construct used, among many other purposes, to build both composite statements and sets of statements. For instance,

$\langle [\!]\ i : 1 \le i \le N :: A[i] := 0 \rangle$

defines a set of statements that zero the elements of the array A, one for each value of *i* in the range *1* to *N*, while

$\langle \|\ i : 1 \le i \le N :: A[i] := 0 \rangle$

constructs one composite statement which performs the same task in a single atomic step.

Let us consider next the actions performed by the algorithm. Each of the statements can be seen in the diagrams of Figure 4. When the message arrives at a MSC, by definition, the channel it arrived on becomes flushed. If the MSC was already marked, i.e. holding a copy of the message, when the new message arrived, no further action is taken (Stmt 1). This is one of the cases where redelivery to a stationary mobile unit must be avoided. However, if the MSC was not previously marked, the arrival of the message triggers a delivery to the mobile unit when the latter is in communication with the MSC, and a dispersion of the message on all outgoing channels (Stmt 2). Through this fanning out of the message and the connectivity assumptions, every MSC eventually processes one message from every incoming channel. When a mobile unit arrives at a MSC, the MSC must determine whether or not to attempt to deliver the message. Trivially, if the MSC does not have a copy of the message, the decision not to deliver is made (Stmt 3a). However, if the MSC is marked and the mobile unit arrives on a flushed channel, it is known that the mobile unit has already received a copy of the message, and we must not redeliver it (Stmt 3b). This will be proven in the next section. If the MSC is marked and the mobile unit arrives on an unflushed channel, delivery is attempted (Stmt 4).

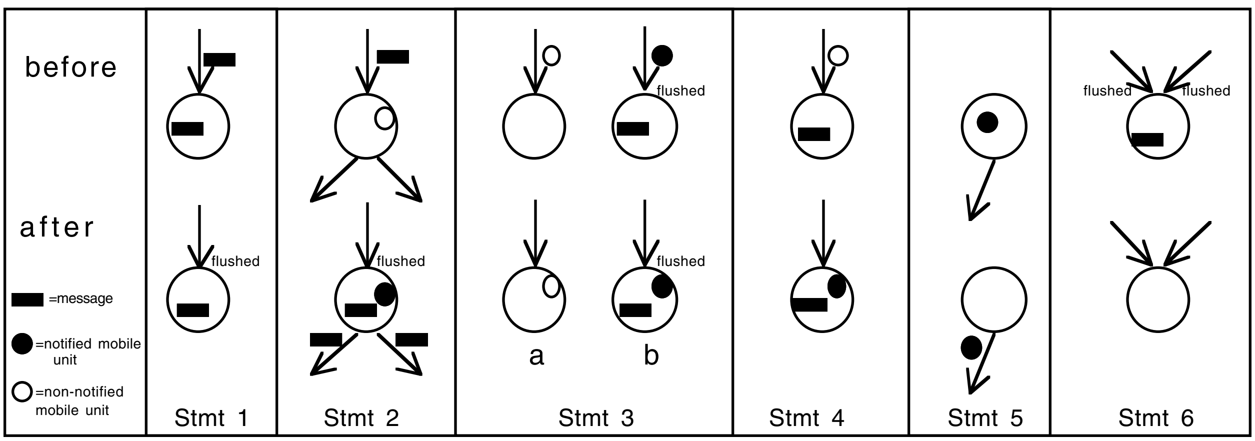

Figure 4. State transition associated with each program statement.

The processing of a mobile unit from the head of a channel models mobile unit movement from a channel to a MSC. We must also allow movement from a MSC to a channel. We add one statement to the program allowing a mobile unit which is located at a MSC to move onto any of its outgoing channels (Stmt 5). Because of the weak fairness provided by UNITY, the statement is eventually selected and the mobile unit moves on, thus modeling its random movement. In most real situations, the movement will be controlled by a user, however this could be coded into the program as a predefined path for the mobile unit to follow. Again, for simplicity, we focus on random movement.

The final statement of the program addresses the requirement that no trace of the message remain in the system when the algorithm terminates (Stmt 6). To do this, the message must be deleted from every MSC, and the channels must return to their original, unflushed, state. We use the local snapshot property of global snapshots to detect this. When the local snapshot terminates, all incoming channels are flushed, and the final statement is enabled for execution, and when it is selected, the cleanup is completed.

3. Proof Outline

In Section 3, we presented a global snapshot algorithm modified to perform message delivery in a mobile system. Because the approach is based on a well understood algorithm from distributed computing, we can adopt the proven properties from the distributed computing environment into the mobile environment. Three main properties are proven for the Chandy-Lamport distributed snapshot: there is no residual storage in the system at some point after the algorithm begins execution, every message is recorded once, and no message is recorded more than once. We translate these properties directly into the mobile environment stating that (A) eventually there is no residual storage in the system at some point after the algorithm begins execution, (B) the message is delivered to the intended recipient, and (C) the message is delivered only once. In this section, we will present a proof outline for these properties based on the UNITY program presented in the previous section.

3.1 No Residual Storage

To show that eventually all information concerning the message is removed from the system, we prove that from the initial conditions, the program will eventually reach a state where there are no messages at any MSCs, there are no messages in any channels, and the channel variable reverses to its original cleared value. This can be captured by the following **leads-to** property:

$$\text{INIT} \rightarrow \langle \forall n,m :: \neg \text{MessageAtMSC}(n) \wedge \neg \mathit{message} \in \text{ChanQ}(n,m) \wedge \neg \text{InChanFlush}(n,m) \rangle \qquad \text{(A)}$$

In addition, we must show that the right hand side is stable. In other words, once all the system variables have been cleared, the algorithm does not become active again. Stability is proven directly from the program text. Because messages are not generated within the program, once all copies of the message are gone, there will never be any more messages in any channels or at any MSCs.

Progress is proven using transitivity and several simpler properties. The first step in proving the property A involves showing that from the initial conditions, the program reaches a state where all MSCs have received a copy of the message and have therefore been notified of its existence; hence the use of the auxiliary variable `notified` which becomes true when the message arrives at the MSC and is never set back to false. Once all MSCs have been notified, we examine the channels. It is possible for a notified MSC to have multiple incoming channels and have only received the message on a subset of those channels. In order to clear the network, we must show that a copy of the message arrives on all incoming channels. This property is tracked on a per channel basis using the boolean variable `flushed`. Therefore, the second step in the proof shows that once all MSCs are `notified`, eventually all channels will be `flushed`. The stability of `notified` allows us to include all MSCs notified in the right hand side. The final stage of the proof uses the `notified` status of all MSCs and the `flushed` status of all channels to prove that the copy of the message at all MSCs is deleted, the channels do not have copies of the message, and all channel variables have been reset.

$$\text{INIT} \rightarrow \langle \forall n :: \mathtt{notified}(n) \rangle \qquad \text{(A.1)}$$

$$\langle \forall n :: \mathtt{notified}(n) \rangle \rightarrow \langle \forall n,m :: \mathtt{notified}(n) \wedge \mathtt{flushed}(n,m) \rangle \qquad \text{(A.2)}$$

$$\langle \forall n,m :: \mathtt{notified}(n) \wedge \mathtt{flushed}(n,m) \rangle \qquad \text{(A.3)}$$
$$\rightarrow \langle \forall n,m :: \neg \text{MessageAtMSC}(n) \wedge \neg \text{InChanFlush}(n,m) \wedge \neg \mathit{message} \in \text{ChanQ}(n,m) \rangle$$

To show that eventually all MSCs are notified (A.1), we introduce a metric to count the number of MSCs which have not yet been notified. In the initial configuration, only the MSC originating the message has a copy, a copy is enqueued on all outgoing channels of that MSC, and the metric is initialized to the total number of MSCs minus one. We can show that from this point, the metric decreases to zero. If the metric is greater than zero, by the connectivity of the graph, there must be a channel from a notified to an unnotified MSC and there must be a message on that channel. We then show that the metric decreases either by some message in another region of the graph being processed by an unnotified MSC, or the identified message moving to the head of the channel and being processed by the previously unnotified MSC, implicitly decreasing the number of unnotified MSCs. We iteratively apply this reasoning to show that eventually all MSCs are notified and the metric has decreased to zero.

To show that once all the MSCs are notified, all channels will eventually be flushed (A.2), we use a similar mechanism as above. We introduce a metric to count the number of channels which have not been flushed and show it decreases to zero. As before, we note that because all MSCs are notified, any channel *c* which has not been flushed must be between two notified MSCs, and in this case there must be a copy of the message on *c*. The metric either decreases by some other channel *c'* being flushed, or the identified message arriving at the head of *c* and being processed. Because both auxiliary variables `flushed` and `notified` are stable, we have shown that eventually all channels are flushed and all MSCs remain notified.

Proving the final progress property involves examining each MSC and its incoming channels. First, we note the difference between the auxiliary variable `flushed` and the concrete variable InChanFlush. When the message arrives from a channel, both variables are set to true. The primary difference is that InChanFlush is set back to false when the channel's destination MSC is cleared, while `flushed` remains true. This allows us to reason about channels which have been traversed by the message, but have been reset.

This is valuable because the left hand side of A.3 only states that all channels have, in the past, had InChanFlush set to true but nothing about the current value of InChanFlush. We know that if InChanFlush is false but `flushed` is true, then the destination MSC has been cleared, and all other incoming channels have been cleared. This can be shown to be stable. However, if InChanFlush is true and `flushed` is true, then all other incoming channels must be in the same state, and the message must be at the MSC. These properties enable Stmt 6 of the program which, when executed, will clear the MSCs and channels. This reasoning can be carried out at all MSCs of the system and eventually the message copies will be removed from MSCs and channels, and all InChanFlush variables will be reset to false.

3.2 Eventual Message Delivery

The next property we must prove is that the message is eventually delivered, i.e., the auxiliary variable `found` is eventually set to true:

$$\text{INIT} \rightarrow \texttt{found} \tag{B}$$

Property B can be proven by using transitivity and conjunction from the following properties:

$$\text{INIT} \rightarrow \langle \forall \text{n}::\texttt{notified}(\text{n})\rangle \tag{B.1}$$
$$\langle \forall \text{n}::\texttt{notified}(\text{n}) \wedge \texttt{found}\rangle \rightarrow \texttt{found} \tag{B.2}$$
$$\langle \forall \text{n}::\texttt{notified}(\text{n}) \wedge \neg\texttt{found}\rangle \rightarrow \texttt{found} \tag{B.3}$$

B.1 was proved earlier in A.1, property B.2 is a consequence of the implication, and B.3 remains to be proven. To prove this property, we must consider the possible locations for a mobile unit which has not been found. If all MSCs are notified, it must be true that the unfound mobile unit is on an unflushed channel, and the mobile unit is ahead of the message, i.e., the following invariant holds:

$$\textbf{inv}\ \langle \forall \text{n}::\texttt{notified}(\text{n}) \wedge \neg\texttt{found}\rangle \tag{B.3.1}$$
$$\Rightarrow \langle \exists \text{n,m}::\ \textit{mobile} \in \text{ChanQ(n,m)} \wedge \neg\texttt{flushed}(\text{n,m}) \wedge \text{mobile.preceeds.ann} \in \text{ChanQ(n,m)}\rangle$$

where mobile.preceeds.ann$\in$ChanQ(n,m) denotes the fact that, if both the message and the mobile unit are in the channel, the mobile unit is closer to the head of channel (n,m). If only the mobile unit is in the channel, this predicate is true by definition. Invariant B.3.1 can be combined with B.3 to give an equivalent property that the right hand side of B.3.1 leads to the mobile unit being found. Because the mobile unit is in front of the message, the mobile unit arrives at the MSC before the channel is flushed, and a message copy must still be at the MSC. Therefore, when the mobile unit moves onto the MSC, delivery must occur.

3.3 Single Delivery

Having shown delivery, it remains to be proven that the message is only delivered one time. To do this, we count the number of times the `found` variable is set to true. This is accomplished by incrementing an auxiliary variable `num_deliveries` each time delivery occurs. To show multiple deliveries do not occur, we must prove that the number of deliveries never exceeds one.

$$\textbf{inv}\ \texttt{num\_deliveries} \leq 1 \tag{C}$$

Initially this property holds. If `num_deliveries` is zero, no statement can violate C because a single statement can only increase `num_deliveries` by one. Once the message has been delivered and `num_deliveries` is one, we focus on the two statements which can increment `num_deliveries` again: Stmt 2 and Stmt 4. In Stmt 2, the message arrives at a MSC which has not yet received a copy of the message. If the mobile unit is present at this MSC, delivery occurs. In Stmt 4, the mobile unit arrives along an unflushed channel at a MSC with a copy of the message. Therefore, delivery occurs. To show that multiple delivery is not possible, it is sufficient to show that once `found` is set to true, neither of the two conditions described above occurs. This is related to C because the definition of `num_deliveries` gives us the fact that when the number of deliveries exceeds zero, the `found` variable must be true. The following properties capture these statements:

$$\textbf{inv}\ \texttt{found} \tag{C.1}$$
$$\Rightarrow \neg(\text{MobileAtMSC(n)} \wedge \neg\text{MessageAtMSC(n)} \wedge \text{head.ChanQ(m,n)}=\textit{message})$$
$$\textbf{inv}\ \texttt{found} \tag{C.2}$$
$$\Rightarrow \neg(\text{head.ChanQ(n,m)}=\textit{mobile} \wedge \neg\text{InChanFlush(n,m)} \wedge \text{MessageAtMSC(m)})$$
$$\textbf{inv}\ \texttt{num\_deliveries}>0 \Rightarrow \texttt{found} \tag{C.3}$$

To prove properties C.1 and C.2, we characterize a region of the graph called *will-be-notified*, and define it as the set of all MSCs which are $\neg$`notified`, the channels which are $\neg$`flushed` and do not have messages on them, and the channel segments between a message and a MSC. Because C.1 and C.2 describe situations where a mobile unit is in this region, a complementary property to C.1 and C.2 is that a found mobile unit must not be in *will-be-notified*.

inv `found` $\Rightarrow$ *mobile*$\notin$ *will-be-notified* (C.1.1)

To use this knowledge to prove properties C.1 and C.2, we define a `path` to be a sequence of MSCs and channels between a mobile unit and the *will-be-notified* region. We state an invariant that once the mobile unit is found, there must be a message in all such paths. This message can either be on a channel or at a MSC.

inv `found` $\Rightarrow \langle \forall$`p:path`::*message*$\in$ p$\rangle$ (C.1.2)

This property can be proven from the initial conditions and over each program statement. Using this definition of a path, it is clear that if the mobile unit is in *will-be-notified*, then there exists at least one path between the mobile unit and *will-be-notified* without a message on it. This relates to C.1, where the mobile unit is at a MSC without the message, because a MSC without a message must be in *will-be-notified*. But if `found` were true, then all paths would have a message on them, and there is a trivial path which does not. Therefore, `found` cannot be true. In C.2, the mobile unit is on an unflushed channel which is also a part of *will-be-notified*. If the mobile unit had not been found, then all paths between the mobile unit and *will-be-notified* would have a message on them, but in C.2, the channel the mobile unit is on is a part of *will-be-notified*, and the `path` between the mobile unit and the channel it is on does not contain a message. Therefore, C.1 and C.2 are true.

By combining C.1 and C.2 with C.3, we have the desired result that once the number of deliveries is set to one (i.e., delivery has been accomplished), the mobile unit is never in a location where it will be delivered to again. Therefore, `found` cannot be set to true again, and `num_deliveries` cannot increase beyond one.

4. Reality Check

Here we reexamine assumptions we made to show that they are reasonable. The issues we discuss here are non-FIFO channels, multiple RBSs per MSC, base station connectivity, reliable delivery on links, the involvement level of MSCs, and storage requirements.

Non-FIFO Channels: One major objection to using the Chandy-Lamport algorithm is its reliance on FIFO behavior of channels. More specifically, in section 2, we modeled both the mobile units and the messages as traveling on the same channel. This seems like an unreasonable assumption given that mobile units move much more slowly through space than messages move through a fixed network. To further explore this problem, we most adopt a more specific model of reality and how the FIFO assumption is broken. Next we will propose a simple fix and show why the channel models FIFO again. Finally, we will relate this back to the handover protocol.

One of the American standards for analog cellular communication is AMPS. In AMPS, cellular telephone tune to only one frequency at a time. This shapes the handover protocol. We adapt this protocol for use with mobile units and the snapshot algorithm. When the signal from one mobile unit begins to degrade, the MSC communicating with it on frequency f_1 sends a message to all bordering cells requesting a signal strength of 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 5a shows the stages of the handover of mobile unit m moving from MSC_1 to MSC_2. First, MSC_1 sends a handover 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 switches to the new frequency and sends a hello to MSC_2. Finally, MSC_2 sends a handover complete message to MSC_1.

By using this 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.

The primary problem we face with this handover is the possibility of double delivery of a message to a mobile unit. This situation arises because it is possible for a message to overtake the mobile unit while it is involved in a handover. An example can be seen in Figure 5b.

When MSC_1 receives a message it broadcasts it to all mobile units in its cell, including m. After this broadcast, the switch message is sent to m and the message is forwarded to MSC_2. When MSC_2 receives the message and sees the hello from m, it does not know if the message was sent to m before or after the switch. Therefore, MSC_2 must send the message to m. In the example presented, this causes a double delivery to m.

The problem arises because instead of having one channel on which both messages and mobile units travel, we have, in effect, two channels which are not synchronized. One way to synchronize them is to simulate the switching of the mobile unit on the physical link by sending a packet representing a virtual mobile unit. The arrival of this virtual mobile unit on the channel indicates to the MSC_2 whether the mobile unit switched before or after the broadcasting of the message at MSC_1. If MSC_2 receives the virtual mobile unit before the hello from the actual mobile unit, MSC_2 delays the broadcast of the announcement, if any are pending, until the arrival of m. If the actual mobile unit arrives before the mobile unit, all communication with m must be delayed until the virtual mobile unit arrives. This is analogous to saying that while the virtual mobile unit is in the channel, it has not arrived at MSC_2. Therefore, the channel has been made FIFO by simply sending one additional packet on the physical link between the MSCs. Figure 5c shows how the virtual mobile unit affects the delivery. Just as the switch message was sent after the broadcast of the message, the vmu is sent after the message is put onto the channel.

One possible disadvantage to this solution is that if the mobile unit arrives before the virtual mobile unit, there is a time when the mobile unit is in communication range of the system, however it is prevented from communicating because the system views it as being on a channel. Although this is probably unlikely because as noted previously, messages move along channels faster than even the switch messages that cause the mobile unit to change location, there is a solution. It is possible to treat the entire handover as being an atomic transaction. This is equivalent to putting the virtual mobile unit into the handover request. In order to make the transaction complete, MSC_1 must not broadcast any messages between the handover request and the handover complete. This too had its disadvantages, namely the delay in processing of messages at the MSCs. Therefore, although the first approach adds messages to the system and requires additional processing at MSC_2, it does not prevent broadcasting at either MSC.

At this point, we should note that this adaptation of the AMPS model of handover does not allow the mobile unit to jump past the message in the channel. Therefore, it is not possible for the mobile unit to miss the message, but only for the message to be delivered twice. Duplicate message processing by the mobile unit can be eliminated by using sequence numbers on each message and requiring the mobile unit to check this value before processing, however this requires storage at the mobile units while the techniques presented above confine the additional processing and storage to the fixed MSCs.

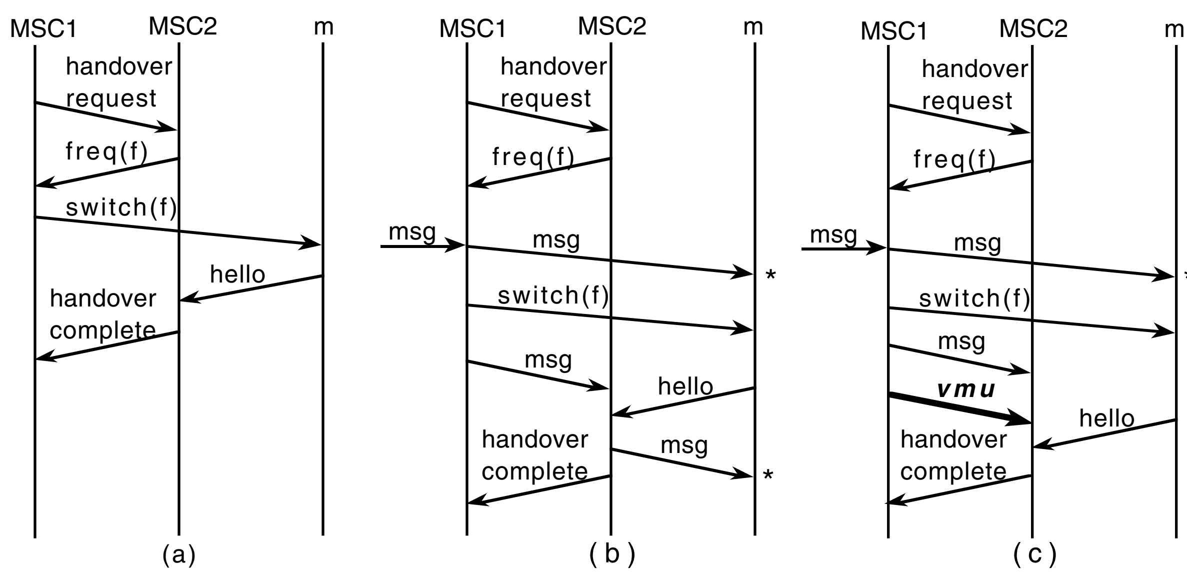

Figure 5 a) AMPS-like handover. b) Double delivery example (* indicates message delivery) c) Virtual mobile unit, FIFO

Multiple RBSs per MSC: Until this point we have only allowed one radio broadcast station for each support center, however, in today's cellular telephone system, MSCs manage sets of RBSs. For example, in Figure 1, it is feasible for one MSC to manage all cells shown. Because the algorithm we presented is intended to be run over the fixed network formed by the MSCs, the handover mechanisms apply only to the movement of a mobile unit from a RBS supported by one MSC to another RBS supported by a different MSC. The question remains about how to broadcast the message within the cells supported by a single MSC and maintain the constraints of single delivery and guarantee the single delivery. Because the MSC acts as a coordinator of the mobile units present at each of the RBSs, it is feasible to run a similar snapshot among the RBSs, allowing it to terminate before any handovers to other MSCs are permitted. This is a simple solution, however it shows how this algorithm can be implemented in a layered fashion.

Base station connectivity: Another possible concern with the model we present is the necessity for physical connections between all MSCs whose cells border one another. This structure is analogous to the structure in place to route cellular telephones. Therefore, we believe that it is reasonable for physically adjacent support centers to be connected by physical links.

Reliable delivery on links: The snapshot delivery algorithm assumes that link delivery is reliable. Most of the Internet uses unreliable links like Ethernets, frame relay, and ATM. The probability of error on such links may be small but packets are indeed dropped. A possible solution is to add acks for multicast messages as is done, for example, in the intelligent flooding algorithm used in Links State Routing in OSI [13] and OSPF [10]. Another solution is to only provide best-effort service. Since lost messages can lead to deadlock we need to delete messages after a timeout even if it is still expected along a channel.

Involvement level of MSCs: In every snapshot, every MSC must be involved to guarantee delivery and termination. In a paper on running distributed computations in a mobile setting [2], the authors warn against required involvement of all mobile nodes in a computation, especially due to the voluntary disconnection often associated with mobile computing. Such disconnection is often done to conserve power, or in some cases, to allow disconnected operation. In either case, the mobile unit is not available for participation in the distributed algorithm. These arguments are important for creating distributed algorithms for mobile computing environments, however our goal is not to create a global snapshot of the mobile units, but instead to employ the snapshot technique to a different end, namely message delivery. Additionally, the control messages of the snapshot are not being run over the mobile units, but rather over the fixed mobile support centers. In order to guarantee delivery, the mobile unit must be present in the system, however, this is a reasonable assumption because there is no means to reach a disconnected mobile unit. At this point, it is interesting to note that if the mobile unit is not present in the system during a delivery attempt, the algorithm will terminate normally, removing all trace of the message from the system, but without delivery.

Storage requirements: In snapshot delivery, we assume that the MSCs hold a copy of the message for delivery to the mobile agents for a bounded period of time limited to the duration of the local snapshot. In a system with bi-directional channels, because the local snapshot terminates when the message arrives on all incoming 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 messages when their primary purpose is routing. However, in this case, because no routing information is being kept about the mobile units, the system will be required to keep additional state in order to provide delivery guarantees. Therefore, keeping a copy for a short duration is a reasonable assumption.

5. Extensions

The snapshot delivery algorithm is extendable to perform delivery of multiple messages simultaneously, delivery to rapidly moving mobile units, route discovery, and multicast. Each of these will be examined in turn.

Multiple message deliveries: To deliver multiple messages simultaneously using snapshots, we run several copies of the algorithm in parallel. This is analogous to having each MSC index and store the incoming messages and maintain separate channel status for each message in the system. This information is kept until the MSC locally determines it can be cleared. In the worst case, every node must have storage available for every potential message in the system, as well as maintain channel status with respect to each message. Although this appears excessive, we maintain that the nature of the snapshot algorithm in a real setting will not require maximum capacity. In other words, because the MSCs are able to locally determine when to delete the messages, the nature of the network will determine how long a message is stored at the MSC.

Rapidly moving mobile units: Another advantage to this algorithm is the ability to operate in rapidly changing environments with the same delivery guarantees. In Mobile IP, mobile units must remain in one place long enough to send a message with their new address to their home agent for forwarding purposes, and remain at that foreign agent long enough for the forwarded messages to arrive. With forwarding enhancements added to the foreign agents in Mobile IP, the issue is minimized because the former location of a mobile unit becomes a kind of packet forwarder. However, even with forwarding, if the agent moves too rapidly and the system is unable to stabilize, forwarded packets will chase the mobile unit around the system without ever being delivered. Because snapshots do not maintain a notion of home or route, movements are immediately accounted for by the delivery scheme. Similarly, using snapshots for delivery has advantages over ad hoc methods of route discovery which create a path to the mobile unit with one pass over the system. In rapidly changing environments, the mobile unit can change location between route discovery and message delivery causing lost messages. The route discovery method is used because the discovery packets are small and inexpensive to transmit over wireless links while data packets are comparatively large. This is important because the nodes use a timeout or cache swapping to delete the routing packets.

Route discovery: In more moderately changing environments, route discovery can increase efficiency and decrease overhead. In these situations, the snapshot delivery algorithm can be used to perform route discovery. When the discovery message from source S located at MSC_S arrives at the destination mobile unit D located at MSC_D, the mobile unit responds with a packet directed toward MSC_S with MSC_D and a particular RBS in its data to identify its location to the source. This assumes that source is also moving relatively slowly, however a route response message could be sent to S from D in a similar manner as the route discovery. All packets sent to D after the discovery should contain MSC_D and RBS, and routing along the fixed network is now possible. Alternately, the snapshot algorithm can be adapted to perform route discovery in an ad hoc environment. Because there is no fixed network to route along, the discovery packet builds a sequence of hops in its payload as it traverses the network. When the packet is delivered, the mobile unit sends a packet to the sender along the reverse of the path shown in the data. This route is then used for future messages. Because the mobile unit is delivered to only once in the snapshot, only one message will be returned to the sender, unlike in a more standard broadcast where the sender could receive multiple routes to the destination due to branching during the broadcast of the discovery packet. The snapshot algorithm explicitly cleans up when it is locally complete, we provide stronger guarantees than traditional route discovery which relies on timeouts and cache swapping to complete routes.

Multicast: As stated previously, another concern in the mobility community is multicast. While Mobile IP addresses the issues of macromobility, and some work has been done on reliable multicast for micromobility [1], our algorithm easily extends to perform multicast to all mobile units in the system during the execution of the snapshot. Without changing the snapshot algorithm, it can be shown that delivery of a message is attempted to all mobile units in the system before the algorithm terminates. If the message contains a broadcast or multicast destination address, delivery can be carried out whenever a connection with the mobile unit accepting those addresses is established. Interestingly, even in broadcast or multicast, the restriction of single delivery of a message holds. Although this description is concise, the importance of it should not be lost in its simplicity.

Our modified snapshot algorithm has worst-case overhead of one message per link in each direction to multicast. By contrast, the algorithm used in IP DVMRP [5] effectively computes a tree. Its overhead is the number of links in the tree plus the number of links that have endnodes that participate in this multicast.

6. Conclusions

In this paper we presented an algorithm for multicast and unicast delivery of messages to mobile nodes in a micromobility environment, proved some essential properties, offered evidence of its potential practical use, and provided additional extensions to make it applicable in multiple settings. In doing so, we introduced a new kind of approach to the study of mobility, one based on a model whose mechanics are borrowed directly from the established literature on distributed computing. Treating mobile units as messages provides an effective means for transferring results from classical distributed algorithms literature to the emerging field of mobile computing. The next challenge we face is to evaluate the viability of the new mobile, i.e., the range of problems for which the message-as-node abstraction is appropriate.

7. Bibliography

[1] A. Acharya and B. R. Badrinath, "Delivering Multicast Messages in Networks with Mobile Hosts," *13th International Conference on Distributed Computing Systems,* New York, pp. , 1993.

[2] B. R. Badrinath, A. Acharya, and T. Imielinski, "Structuring Distributed Algorithms for Mobile Hosts," *Proceedings of the 14th Intl. Conf. on Distributed Computing Systems,* Poznan, Poland, pp. 21-8, 1994.

[3] K. M. Chandy and L. Lamport, "Distributed Snapshots: Determining Global States of Distributed Systems," *ACM Transactions on Computing Systems*, vol. 3, no. 1, pp. 63-75, 1985.

[4] K. M. Chandy and J. Misra, *Parallel Program Design: A Foundation*. Addison-Wesley, 1988.

[5] S. E. Deering and D. R. Cheriton, "Multicast Routing in Datagram Internetworks and Extended LANs," *ACM Trans. on Computer Systems*, vol. 8, no. 2, pp. 85-110, 1990.

[6] D. B. Johnson and D. A. Maltz, “Truly seamless wireless and mobile host networking. Protocols for adaptive wireless and mobile networking,” *IEEE Personal Communications*, vol. 3, no. 1, pp. 34-42, 1996.
[7] J. J. Kistler and M. Satyanarayanan, “Disconnected Operation in the Coda File System,” *ACM Trans. Computer Systems*, vol. 10, no. 1, pp. 3-25, 1992.
[8] P. Krishna, M. Chatterjee, N. H. Vaidya, and D. K. Pradhan, “A cluster-based approach for routing in ad-hoc networks,” *USENIX Symposium on Mobile and Location-Independent Computing*, pp. 136-146, 1995.
[9] S. Kryukova, B. Massingill, and B. Sanders, “Specification and Proof of an Algorithm for Location Management for Mobile Communication Devices,” *International Workshop on Formal Methods for Parallel Programming: Theory and Applications,* Geneva, pp. , 1997.
[10] J. Moy, “OSPF Version 2,” Internet Engineering Task Force, Internet draft March 1994 1994.
[11] A. Myles and D. Skellern, “Comparing Four IP Based Mobile Host Protocols,” *Computer Networks and ISDN Systems*, vol. 26, no. , pp. 349-355, 1993.
[12] C. Perkins, “IP Mobility Support,” IETF Network Working Group, Technical Report RFC 2002, October 1996.
[13] H. Zimmerman, “OSI Reference Model -- The ISO Model of Architecture for Open Systems Interconnection,” *IEEE Transactions on Communication*, vol. 28, no. , pp. 425-432, 1980.