Washington

WASHINGTON • UNIVERSITY • IN • ST • LOUIS

School of Engineering & Applied Science

An Algorithm for Message Delivery to Mobile Units

Amy L. Murphy
Gruia-Catalin Roman
George Varghese

WUCS-96-28

14 November 1997

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

An Algorithm for Message Delivery to Mobile Units

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

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 unit, *unicast*, or to a group of mobile units, *multicast*. Standard solutions used in Mobile IP and cellular phones for the unicast problem rely on *tracking* the mobile unit. Tracking solutions scale badly when mobile units move frequently, and do not generalize well to multicast delivery. Our paper proposes a new message delivery algorithm based on a modification of classical snapshot algorithms and includes a proof outline using the UNITY logic. Our algorithm requires no tracking, provides stronger guarantees than existing protocols, and generalizes easily to multicasting to mobile units. 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 units 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 from office to home and around the country with the computer at their side. Both location-transparent and context-dependent services are desired. Decoupled computing is becoming the norm [8]. Disconnection is no longer a network fault but a common event intentionally caused by the user in order to conserve power or a consequence of movement.

Nevertheless, the typical model of distributed computing treats a network as a graph in which vertices represent processing nodes and edges denote communication channels. Faults may render parts of the network inoperational either temporarily or permanently. Despite faults, the overall structure is considered to be static. One way to introduce mobility in this context is to treat some of the nodes of a network as mobile support centers (MSC) and to introduce a new breed of nodes, called mobile. The latter type are allowed only to connect to and disconnect from mobile support centers. The result is a fixed core of static nodes and a fluid fringe consisting of mobile units. This turns out to be the dominant model in mobile computing today.

In this paper we suggest yet another way of thinking about mobility in the context of the traditional fixed graph structure. The basic idea is to treat mobile units as roving messages which preserve their identity as they travel across the network. While operating inside one cell, a cellular phone may be viewed as residing at a node inside the support network; similarly, the handover protocol (triggered by the detection of signal degradation) may be modeled as the traversal of a channel between two nodes representing the individual cells. Voice transmissions among two phones are also modeled as messages.

Because of this dual role messages can play in this new model, we need to be careful with terminology. Henceforth, we will use <u>network</u> to denote a static organization of processing <u>nodes</u> and <u>channels</u>; we will reserve the term <u>mobile unit</u> for messages that model mobile units; and we will employ the term <u>announcement</u> for any messages which carry information from one mobile unit to another. Announcements, as just defined, originate with a mobile unit and are passed among nodes in order to achieve eventual delivery to some other mobile unit. We refer to all other kinds of messages simply as messages. Of course, both mobile units and announcements are <u>messages</u>, however, a mobile unit is a special message that retains an identity as it passes through the network.

Our interest in this model rests with *its ability to facilitate the application of established distributed algorithms to problems in mobile computing*. To illustrate this point, this paper shows how algorithms designed to

[1] The work of the first two authors is supported in part by the National Science Foundation under Grant No. CCR-9217751 and CCR-9624815. 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.

compute distributed global snapshots provide reasonable solutions for the problem of multicasting announcements among mobile units. Furthermore, these results are immediately applicable to the related problems of route discovery and maintenance among mobile units in the presence of mobile support centers.

The remainder of this paper is organized as follows. In Section 2, we outline the problem of message delivery in a mobile setting, propose a simple solution, and discuss its shortcomings. Section 3 introduces our approach to announcement delivery using snapshot algorithms and specifies the solution as a UNITY [3] program. Section 4 outlines the proof of correctness for our delivery algorithm. In Section 5, we discuss the broader implications of these results, show some potential shortcomings of our approach and ways to address them. Section 6 consists of some brief concluding remarks.

2. Announcement Delivery to Mobile Units: Problem Statement

The problem we are interested in is the delivery of announcements among pairs of mobile units. A mobile unit can send and receive announcements only when it is present at some node in the fixed network, a situation that models the existence of an established connection between a mobile unit and a support station. Whenever a mobile unit is on a channel, it may be viewed as being temporarily disconnected from the network and, therefore, unable to communicate with any of the nodes.

The *announcement delivery problem* can now be formulated as follows: Given a fully connected network with FIFO channels and guaranteed message delivery, an announcement located at one of its nodes, and a roving mobile unit for which the announcement is destined, develop a distributed algorithm that guarantees single delivery of the announcement, and leaves no trace of the announcement, at either a node or a mobile unit, within a bounded time after delivery. Minimizing storage requirements across the network is another concern.

Standard solutions to unicast announcement delivery (e.g., cellular phones [6] and Mobile IP [12]) rely on *tracking* a mobile unit as it moves around a fixed wired network. For example, in cellular systems as a phone P moves from cell C to adjoining cell C', the phone detects a stronger signal from the MSC at C' and requests a handover from C to C'. After the handover, C' is now responsible for forwarding voice calls to the phone, and the cellular system keeps track of this association between C' and P. Similarly in Mobile IP, a mobile unit M has a *home agent*. When M moves to another location, M contacts a *foreign agent*. The home agent is then informed that M is now reachable through the foreign agent. Once again, Mobile IP attempts to keep track of the current location of a mobile unit. In Mobile IP with forwarding optimizations, if the mobile unit moves too rapidly and the system is unable to stabilize, forwarded packets will chase the mobile unit around the system without ever being delivered.

While the *unicast* problem of delivering an announcement to a *single* recipient is important, in recent years the *multicast* problem of delivering an announcement to *multiple* recipients has also become crucial. Multicast support through the MBONE has become a standard part of the Internet [5, 13], and is finding wide use for conferencing (e.g., tools like VIC and VAT [7, 9]) and video distribution. Work has been done to provide reliable multicast to a mobile community [1], however the members of the community must be known in advance of the multicast. While multicasting and mobility have been treated as important extensions for the next generation Internet, the working groups in these areas have largely proceeded separately. An important open problem then is that of multicasting to mobile units.

Since tracking does not work well with rapidly moving nodes and we wish to support multicasting, our paper considers solutions to Announcement delivery based on *search*. The idea is to keep no tracking information but to search for the mobile unit(s) whenever they are needed. Clearly, searching the entire Internet for a mobile unit appears ludicrous. However, the Internet is divided into a number of hierarchical domains and subnets and the intent is to perform a form of broadcast search within a small local domain. For example, a cable office may wish to multicast announcements to a number of cable servicemen that are roaming in the metro St. Louis area.

Broadcast search is not quite trivial because the mobile unit may move from node to node, like the Artful Dodger, one step ahead of the broadcast. This seems unlikely in practice, however, because mobile units move quickly with respect to the propagation of a announcement. The problem returns when the actual propagation methods employed to perform message delivery are examined. One such mechanism is to create a spanning tree over all nodes in the graph, and propagate the announcement along this path. As long as the mobile unit remains at a single location, delivery is trivial. However, if the mobile unit is at the border between two cells and about to start a handover when delivery starts, it is possible for the mobile unit to move from a node farther down in the spanning tree to one higher in the spanning tree in between the times when the announcement had already propagated past the higher node and when it arrived at the lower node. This is due to the fact that a handover is accomplished by message passing, and therefore the movement of the mobile unit is now on the same order of magnitude as message passing. This problem can be solved if nodes keep copies of the Announcement for an indefinite period. However, Internet routers do not have the storage to keep application messages indefinitely. Announcements must be garbage collected quickly if the scheme is to have any chance of being practical. Our solution has the attractive property of guaranteeing delivery exactly once (in practice, with unreliable links, delivery would be at most once) while allowing rapid garbage collection in time proportional to a round trip delay on a single link.

Our solution is based on the classical notion of a snapshot. In particular, our preferred embodiment is based on the original Chandy-Lamport snapshot algorithm [2]. In doing so, we bring together the two concerns of the paper: applying techniques from distributed algorithms to mobile computing, and the problem of announcement delivery. The use of snapshots is based on the intuitive notion that a mobile unit cannot avoid a snapshot: the mobile unit(s) must show up in the snapshot at some node or link in some local snapshot. We can therefore modify the local snapshot algorithms to effect announcement delivery. A snapshot, thus, plays the role of a rather thorough posse.

3. An Algorithm Based on Snapshots

The solution we propose combines the ideas of tracking and broadcast by noting that only one node will actually deliver the announcement, but without restricting this node to a single home agent as in Mobile IP and addressing the memory concerns of broadcast. The problem now becomes uniquely identifying this processing node in a changing system, and allowing all other nodes to locally distinguish that they are not the delivery node and therefore do not need to retain a copy of the announcement for delivery.

To identify the single processing node that will deliver the announcement, we propose modifications to global snapshot algorithms. 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. In the mobile setting, a local snapshot contains information about the mobile units in current communication with the node, as well as the mobile units and announcements 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 announcement sent directly to that location may miss the mobile unit. For this reason, the computation of the snapshot itself must be modified to perform announcement delivery. Specifically, we note that because each mobile unit must appear only once in the global snapshot, and the announcement can only be delivered once, *the node that records the destination mobile unit is responsible for delivering the announcement*.

In general, snapshot algorithms proceed across a network by passing the knowledge that the snapshot is occurring to neighboring nodes. We assume throughout that all channels are FIFO. We append to this knowledge the actual announcement. When the announcement arrives at a node, it is stored and the local snapshot begins. As the snapshot proceeds, the node 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 processing node deletes its copy of the announcement, assured that the destination mobile unit will appear in some other processing node's local snapshot. However, if the destination node is part of the snapshot, either in communication or on an incoming channel, this node is responsible for delivery. Because the nodes watch for the arrival of the mobile unit, as soon as it arrives, delivery occurs. When the local snapshot completes, the message can be deleted because the node has already delivered the message. In this manner, as the local snapshots complete, the announcement copies are removed from the processing nodes, 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 1. UNITY [3] 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 announcements and mobile units. Actually, since we are interested in the behavior of the algorithm solely with respect to a particular mobile unit and one announcement 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 *announcement* and *mobile*. The state of the individual nodes is captured by three arrays: *MobileAtNode*

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

Figure 1. Snapshot Delivery Program for FIFO Channels

is indexed by node identifiers and holds a boolean value which is true whenever the mobile unit is present at that node and false otherwise; *AnnouncementAtNode* is defined in the same manner but denotes the fact that the node is holding a copy of the announcement; finally, *InChanFlush*, indexed by channels, allows a node 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 node identifiers, records the fact that a node has seen the announcement. The variable `found` becomes true when the announcement is delivered to the mobile unit, while `location` records the identity of the last channel or node 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 announcement at an arbitrary node, x_0. This node is marked as holding the announcement, i.e., *AnnouncementAtNode* is true for node x_0 and in no other place; `notified` is initialized accordingly. The contents of the channels are initialized with respect to the location of the mobile unit (reflected by the arbitrary initial value for `location`) and x_0 and the announcement All outgoing channels of x_0 receive a copy of the announcement and if the mobile unit is on the same channel with the announcement it must precede the announcement in the channel. If the mobile unit is at the node x_0, i.e., the mobile unit is at the location where the announcement was dropped off, the announcement 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** $x < y \parallel y := x$ **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 to, among many other purposes, 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

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

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 2. When the announcement arrives at a node, by definition, the channel it arrived on becomes flushed. If the node was already marked when the announcement 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 node was not previously marked, the arrival of the announcement triggers a delivery to the mobile unit when the latter is in communication with the node, and a dispersion of the announcement on all outgoing channels (Stmt 2). Through this fanning out of the announcement and the connectivity assumptions, every node eventually processes one announcement from every incoming channel. When a mobile unit arrives at a processing node, the node must determine whether or not to attempt to deliver the announcement. Trivially, if the node does not have a copy of the announcement, the decision not to deliver is made (Stmt 3). However, if the node 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 announcement, and we must not redeliver it (Stmt 3). If the node is marked and the mobile unit arrives on an unflushed channel, delivery is attempted (Stmt 4).

The processing of a mobile unit from the head of a channel models mobile unit movement from a channel to a node. We must also allow movement from a node to a channel. We add one statement to the program allowing a mobile unit which is located at a node 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

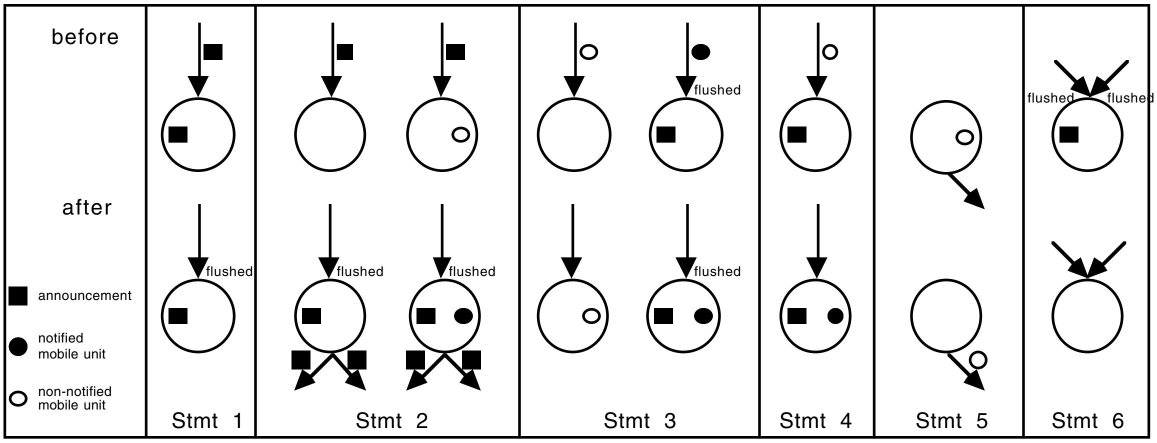

Figure 2. State transition associated with each program statement.

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 announcement remain in the system when the algorithm terminates (Stmt 6). To do this, the announcement must be deleted from every node, and the channels must return to their original, unflushed, state. We use the local snapshot property of global snapshots to detect this. The local snapshot terminates when all incoming channels are flushed, and the final statement is enabled for execution, and when it is selected, the cleanup is completed.

4. Proof Outline

In Section 3, we presented a global snapshot algorithm modified to perform message delivery in a mobile system. We now exploit the formal power of UNITY to present a proof outline of the correctness of the *SnapDeliver* algorithm. Our obligations include showing that (A) there is no residual storage in the system at some point after the algorithm begins execution, (B) the announcement is delivered to the intended recipient, and (C) the announcement is delivered only once. We approach these obligations one at a time.

4.1 No Residual Storage

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

$$\text{INIT} \rightarrow \langle \forall n,m :: \neg\text{AnnouncementAtNode}(n) \wedge \neg \textit{announcement} \in \text{ChanQ}(n,m) \wedge \neg\text{InChanFlush}(n,m)\rangle \qquad \text{(A)}$$

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 announcements are not generated within the program, once all copies of the announcement are gone, there will never be any more announcements in any channels or at any nodes.

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 nodes have received a copy of the announcement and have therefore been notified of its existence; hence the use of the auxiliary variable `notified` which becomes true when the announcement arrives at the node and is never set back to false. Once all nodes have been notified, we examine the channels. It is possible for a notified node to have multiple incoming channels and have only received the announcement on a subset of those channels. In order to clear the network, we must show that a copy of the announcement 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 nodes are `notified`, eventually all channels will be `flushed`. The stability of `notified` allows us to include all nodes notified in the right hand side. The final stage of the proof uses the `notified` status of all nodes and the flushed status of all channels to prove the copy of the announcement at all nodes is deleted, the channels do not have copies of the announcement, and all channel variables have been reset.

$$\text{INIT} \rightarrow \langle \forall n :: \texttt{notified}(n) \rangle \quad \text{(A.1)}$$
$$\langle \forall n :: \texttt{notified}(n) \rangle \rightarrow \langle \forall n,m :: \texttt{notified}(n) \wedge \texttt{flushed}(n,m) \rangle \quad \text{(A.2)}$$
$$\langle \forall n,m :: \texttt{notified}(n) \wedge \texttt{flushed}(n,m) \rangle \quad \text{(A.3)}$$
$$\rightarrow \langle \forall n,m :: \neg \text{AnnouncementAtNode}(n) \wedge \neg \text{InChanFlush}(n,m) \wedge \neg \textit{announcement} \in \text{ChanQ}(n,m) \rangle$$

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

To show that once all the nodes 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 nodes are notified, any channel *c* which has not been flushed must be between two notified nodes, and in this case there must be a copy of the announcement on *c*. The metric either decreases by some other channel *c'* being flushed, or the identified announcement 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 nodes remain notified.

Proving the final progress property involves examining each node and its incoming channels. First, we note the difference between the auxiliary variable flushed and the traditional variable InChanFlush. When the announcement 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 node is cleared, while flushed remains true. This allows us to reason about channels which have been traversed by the announcement, 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 node 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 throughout, then all other incoming channels must be in the same state, and the announcement must be at the node. These properties enable Stmt 6 of the program which, when executed, will clear the nodes and channels. This reasoning can be carried out at all nodes of the system and eventually the announcement copies will be removed from nodes and channels, and all InChanFlush variables will be reset to false.

4.2 Eventual Announcement Delivery

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

$$\text{INIT} \rightarrow \texttt{found} \quad \text{(B)}$$

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

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

The first (B.1) was proved earlier in A.1, the second (B.2) is a consequence of the implication, and the third (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 nodes are notified, it must be true that the unfound mobile unit is on an unflushed channel, and the mobile unit is ahead of the announcement, i.e., the following invariant holds:

$$\textbf{inv}\ \langle \forall n :: \texttt{notified}(n) \wedge \neg \texttt{found} \rangle \quad \text{(B.3.1)}$$

$\Rightarrow \langle \exists$n,m:: *mobile*$\in$ChanQ(n,m) $\wedge \neg$`flushed`(n,m) $\wedge$ mobile.preceeds.ann$\in$ChanQ(n,m)$\rangle$

where mobile.preceeds.ann$\in$ChanQ(n,m) denotes the fact that, if both the announcement 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 announcement, the mobile unit arrives at the node before the channel is flushed, and an announcement copy must still be at the node. Therefore, when the mobile unit moves onto the node, delivery must occur.

4.3 Single Delivery

Having shown delivery, it remains to be proven that the announcement 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.

inv `num_deliveries` ≤ 1 (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 announcement 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 announcement arrives at a node which has not yet received a copy of the announcement. If the mobile unit is present at this node, delivery occurs. In Stmt 4, the mobile unit arrives along an unflushed channel at a node with a copy of the announcement. 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:

inv `found` (C.1)
$\Rightarrow \neg$(MobileAtNode(n) $\wedge \neg$AnnouncementAtNode(n) $\wedge$ head.ChanQ(m,n)=*announcement*)
inv `found` (C.2)
$\Rightarrow \neg$(head.ChanQ(n,m)=*mobile* $\wedge \neg$InChanFlush(n,m) $\wedge$ AnnouncementAtNode(m))
inv `num_deliveries`>0 $\Rightarrow$ `found` (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 nodes which are $\neg$`notified`, the channels which are $\neg$`flushed` and do not have announcements on them, and the channel segments between an announcement and a node. 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 nodes 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 an announcement in all such paths. This announcement can either be on a channel or at a node.

inv `found` $\Rightarrow \langle \forall$p:`path`::*announcement*$\in$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 an announcement on it. This relates to C.1, where the mobile unit is at a node without the announcement, because a node without an announcement must be in *will-be-notified*. But if `found` were true, then all paths would have an announcement 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 an announcement 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 an announcement. 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.

5. Extensions and Reality Check

To deliver multiple announcements simultaneously using this approach, we augment the state information and storage requirements at each node. Specifically, every node indexes and stores the incoming announcements and maintains a separate flushed status for each announcement in the system. This information is maintained until the node locally determines they can be cleared. In the worst case, every node must have storage available for every potential announcement in the system, as well as maintain the flushed status of each channel with respect to each announcement. Although this appears excessive, we maintain that the nature of the algorithm in a real setting will not require maximum capacity. In other words, because the nodes are able to locally determine when to delete the announcements, the nature of the problem will determine how long an announcement is stored at a node.

An 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 current address to their home agent for forwarding purposes, remain at that foreign agent long enough for the forwarded messages to arrive. With forwarding enhancements added to the foreign agents, this issue is minimized because the former location of a mobile unit becomes a kind of packet forwarder. However, even with forwarding, if the mobile unit 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.

In more moderately changing environments, route discovery can increase efficiency and decrease overhead. In these situations, the snapshot delivery algorithm can be adapted to perform route discovery by having a message travel back to the original location containing information about the current location of the mobile unit. Because the snapshot algorithm explicitly cleans up when it is locally complete, we provide stronger guarantees than some route discovery mechanisms which rely on timeouts and cache swapping.

Another advantage of snapshot algorithms can be seen with broadcast. The IP backbone is based heavily on the principles of broadcast, and many applications make use of this feature. However, in existing approaches to mobility much of the work has focused on point to point delivery. The snapshot delivery algorithm can be trivially extended to support multiple mobile units or a multicast address as destinations. Without changing the algorithm, it can be shown that delivery of an announcement is attempted to all mobile units before the algorithm terminates. If the announcement's destination address was changed to contain a broadcast or multicast address, delivery could be carried out whenever a connection with a mobile unit accepting those addresses was encountered. Unlike [1], the group of mobile units interested in the multicast need not be known when the message is sent, but we still guarantee delivery to all mobile units present in the graph for the duration of the delivery process. Interestingly, even in broadcast or multicast, the restriction of single delivery of an announcement holds.

Our modified multicast snapshot algorithm has worst-case overhead of one announcement per link in each direction to multicast. By contrast, the algorithm used in IP DVMRP [4] 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.

Next we reexamine assumptions we made to show that they are reasonable. The issues we discuss here are non-FIFO channels, base station connectivity, reliable delivery on links, and storage requirements.

Non-FIFO channels: One major objection to using the Chandy-Lamport algorithm is its reliance on FIFO channel behavior. More specifically, we modeled both the mobile units and the announcement as traveling on the same channel. This seems like an unreasonable assumption given that mobile units move much more slowly through space than messages through a fixed network. To further explore this problem, we must adopt a more specific model of reality and show how the FIFO assumption can be integrated as a simple extension. In [11], we provide a detailed description of the American standard for cellular communication, AMPS, and how adding a simple message to the handover protocol can make the channel FIFO without sacrificing the mobility of the units or the guarantees of the algorithm. With this extension, we suggest that it is possible to reason about FIFO channels in reality as well as theory.

Base station connectivity: Another possible concern with the model we present is the necessity for physical connections between all nodes whose cells border one another. Although this is not practical due to the expense of such connectivity, it is possible to model full connectivity by adding virtual channels between nodes that do not have physical connections. In addition to the announcement following these virtual channels, the handover protocol must also be adapted to guarantee the FIFO channel assumption.

Storage requirements: In snapshot delivery, we assume that the nodes hold a copy of the announcement for delivery to the mobile units 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 announcement 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 these nodes to be maintaining copies of the 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.

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 Link State Routing in OSI [14] and OSPF [10]. Another solution is to only provide best-effort service. Since lost messages can lead to deadlock we need to delete announcements after a timeout even if an announcement is still expected on a link.

6. Conclusions

In this paper we presented an algorithm for multicast and unicast delivery of messages to mobile units [13], provided an assertional-style proof for the algorithm, and offered some evidence of its potential practical use. 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 form 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 model, i.e., the range of problems for which treating the mobile unit as a persistent message is an appropriate abstraction.

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. 292-9, 1993.

[2] 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.

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

[4] 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.

[5] H. Eriksson, "MBONE: The Multicast Backbone," *Communications of the ACM*, vol. 37, no. 8, pp. 54-60, 1994.

[6] J. Ioannidis and G. Q. Macguire, Jr., "The Design and Implementation of a Mobile Internetworking Architecture," *1992 Winter Usenix*, pp. 491-502, 1993.

[7] V. Jacobson and S. McCanne, "Visual Audio Tool," ftp://ftp.ee.lbl.gov/conferencing/vat, Lawrence Berkeley Laboratory, computer program 3.4, October 1995.

[8] 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.

[9] S. McCanne and V. Jacobson, "vic: A Flexible Framework for Packet Video," *ACM Multimedia '95,* San Francisco, CA, pp. 511-522, 1995.

[10] J. Moy, "OSPF Version 2," Internet Engineering Task Force, Internet draft March 1994.

[11] A. L. Murphy, G.-C. Roman, and G. Varghese, "An Algorithm for Message Delivery in a Micromobility Environment," Washington University in St. Louis, Technical Report WUCS-97-28, 1997.

[12] C. Perkins, "IP Mobility Support," Internet Engineering Task Force, ftp://ftp.ietf.cnri.reston.va.us/internet-drafts/draft-ietf-mobileip-protocol-16-txt, Internet draft draft-ietf-mobileip-16, April 22 1996.

[13] J. B. Postel, "Internet Protocol," Network Working Group, Technical Report RFC 791, September 1981.

[14] H. Zimmerman, "OSI Reference Model -- The ISO Model of Architecture for Open Systems Interconnection," *IEEE Transactions on Communication*, vol. 28, no. 4, pp. 425-432, 1980.