Washington University in St. Louis

Washington University Open Scholarship

All Computer Science and Engineering Research

Computer Science and Engineering

Report Number: WUCS-99-30

1999-01-01

Tracking Mobile Units for Dependable Message Delivery

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

As computing components get smaller and people become accustomed to having computational power at their disposal at any time, mobile computing is developing as an important research area. One of the fundamental problems in mobility is maintaining connectivity through message passing as the user moves through the network. An approach to this is to have a single home node constantly track the current location of the mobile unit and forward messages to this location. One problem with this approach is that during the update to the home agent after movement, messages are often dropped, especially in the case of... **Read complete abstract on page 2.**

Follow this and additional works at: https://openscholarship.wustl.edu/cse_research

Part of the Computer Engineering Commons, and the Computer Sciences Commons

Department of Computer Science & Engineering - Washington University in St. Louis
Campus Box 1045 - St. Louis, MO - 63130 - ph: (314) 935-6160.

Tracking Mobile Units for Dependable Message Delivery

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

Complete Abstract:

As computing components get smaller and people become accustomed to having computational power at their disposal at any time, mobile computing is developing as an important research area. One of the fundamental problems in mobility is maintaining connectivity through message passing as the user moves through the network. An approach to this is to have a single home node constantly track the current location of the mobile unit and forward messages to this location. One problem with this approach is that during the update to the home agent after movement, messages are often dropped, especially in the case of frequent movement. In this paper, we present a new algorithm which uses a home agent, but maintains information regarding a subnet within which the mobile unit must be present. We also present a reliable message delivery algorithm which is superimposed on the region maintenance algorithm. Our strategy is based on ideas from diffusing computations as first proposed by Dijkstra and Scholten. Finally, we present a second algorithm which limits the size of the subnet by keeping only a path from the home node to the mobile unit.

Washington

WASHINGTON·UNIVERSITY·IN·ST·LOUIS

School of Engineering & Applied Science

Tracking Mobile Units for
Dependable Message Delivery

Amy L. Murphy
Gruia-Catalin Roman
George Varghese

WUCS-99-30

December 16, 1999

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

Tracking Mobile Units for Dependable Message Delivery

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

December 16, 1999

Abstract

As computing components get smaller and people become accustomed to having computational power at their disposal at any time, mobile computing is developing as an important research area. One of the fundamental problems in mobility is maintaining connectivity through message passing as the user moves through the network. An approach to this is to have a single home node constantly track the current location of the mobile unit and forward messages to this location. One problem with this approach is that during the update to the home agent after movement, messages are often dropped, especially in the case of frequent movement. In this paper, we present a new algorithm which uses a home agent, but maintains information regarding a subnet within which the mobile unit must be present. We also present a reliable message delivery algorithm which is superimposed on the region maintenance algorithm. Our strategy is based on ideas from diffusing computations as first proposed by Dijkstra and Scholten. Finally, we present a second algorithm which limits the size of the subnet by keeping only a path from the home node to the mobile unit.

Keywords: Mobile computing, dependable message delivery, tracking, diffusing computations, distributed computing

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. Both location-transparent and context-dependent services are desired. Decoupled computing is becoming the norm. 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. Tethered connectivity is making way to opportunistic transient connections via radio or infrared transmitters.

The focus of this paper is message delivery to mobile units. In a fixed network, message delivery relies on established routes through the network. Although faults can render parts of the network inoperational or even inaccessible, it is assumed that these faults are infrequent and the system is able to stabilize despite the changes. In mobility, the changing connectivity of the mobile components is not a fault, but rather a feature. As a mobile unit moves through the network, its accessibility point changes. In the base station model of mobility, similar to the cellular telephone system, each mobile unit is connected to the network at a single point, or base station. This base station can either be wired or wireless, but connectivity changes with greater frequency than typical of network faults. Our goal is to be able to get a message to a mobile unit as it is moving among cells. In this environment mobile units can have multiple points of connection in a short period of time, mobile units can disconnect completely from the network, movement is not necessarily predictable, and tracking the current location of the mobile unit at multiple places in the network is too expensive.

One proposed solution to delivering a message to a mobile unit is Mobile IP [6]. Every mobile unit is assigned a single home agent which is responsible for forwarding messages (packets) to the mobile user. Each time the mobile unit moves, it must provide the home agent with a new location. This solution is simple in that it does not require any infrastructure changes in the network. Encapsulation and endpoint specific software are used to accomplish

location transparency. More distributed approaches update the routers themselves with forwarding information. By keeping information closer to the mobile unit [5] the potentially long path for a message originating near the current location of the mobile unit can be short circuited by not sending it all the way to the home agent. However, neither solution provides reliable delivery. It is possible for a packet to be sent from the home agent toward the mobile agent, and for the mobile agent to move before the packet is delivered. In Mobile IP a higher layer in the network protocol stack is responsible for reliability and retransmission when necessary. Eventually, the packets are delivered with reasonable probability.

In cellular telephones, a system similar to Mobile IP is employed when users roam outside their home region [9]. When the telephone is activated, the user registers with the home, indicating essentially a new area code for redirecting calls. The registration process occurs infrequently because the most common case is for the user to remain within a single region. Within that region, another approach must be taken to locate the mobile each time a call arrives, and handovers are used to maintain connectivity when a user crosses a cell boundary during a session.

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

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

These technological trends pose the following interesting question: *Can we devise efficient protocols that guarantee delivery to a node (or set of nodes) that move at arbitrary speeds across a fixed network?* Clearly, trivial solutions exist, they broadcast the message to all nodes and store the message at all nodes until the mobile arrives. By contrast, more efficient solutions should limit the broadcast range and/or the amount of storage required.

In this paper, we start with the idea of employing diffusing computations proposed by Dijkstra and Scholten [3] and adapt it to message delivery. By equating the root node of the computation to the concept of a home agent from Mobile IP, and by replacing the messages of the computation with mobile units, the result is an algorithm which, instead of tracking a computation as messages are passed through a system of processing nodes, tracks the movement of a mobile unit as it visits various base stations in the system. Essentially, the graph of the Dijkstra-Scholten algorithm defines a region within which the mobile unit is always located. Although this is not directly a message delivery algorithm, by propagating a message throughout this region, we can achieve message delivery. The algorithm can be readily adapted for this purpose and can be optimized for message delivery, e.g., our solution prunes unnecessary portions of the graph reducing the area to which a message must be propagated.

Our approach to algorithm development involves the application of a new paradigm which adapts algorithms from traditional distributed computing to the mobile environment. This paradigm treats mobile units as messages that travel across the network, and examines adaptations of standard distributed algorithms to mobility; in this case, Dijkstra-Scholten as a tracking algorithm. One of the assumptions often made in distributed computing is FIFO channel behavior. The algorithms we developed rely on the ability to ensure this property in the mobile environment. In particular, when mobile units and messages move along the same channel. This may seem unrealistic considering that mobile units move slowly in comparison to the transmission of messages along wires.

2

The remainder of the paper is organized as follows: Section 2 presents our model of moblity, offers a precise formulation of the problem and presents our algorithm for ensuring the FIFO behavior of mobile units and messages. Section 3 explores the details of a message delivery algorithm derived directly from the Dijkstra-Scholten model for diffusing computations. Section 4 presents another algorithm inspired by the first, but reduces the message delivery overhead. For this algorithm, we provide a formal verification of its properties. Finally, Section 6 contains analysis and conclusions.

2 Problem Definition: Message Delivery

The problem we are interested in is the delivery of messages to rapidly moving mobile units. An acceptable solution should guarantee at at least once delivery of the message, minimize storage requirements across the network, and leave no trace of the message in the system within a bounded time after delivery. Because mobile units do not communicate directly with one another, the network must provide the support to deliver messages.

The cellular telephone design provides the foundation for the model of mobility we adopt in this paper. Figure 1a shows a typical cellular telephone model with a single mobile support center (MSC) in each cell. The MSC is responsible for communication with the mobile units within its region and serves as a manager for handover requests when a mobile moves between MSCs. Figure 1b shows how the cellular telephone model is transformed into a graph of nodes and channels where the nodes represent the individual cells and the channels represent the ability of a mobile unit to move from one cell to another. We assume that the resulting network is connected, in other words, a path exists between every pair of nodes.

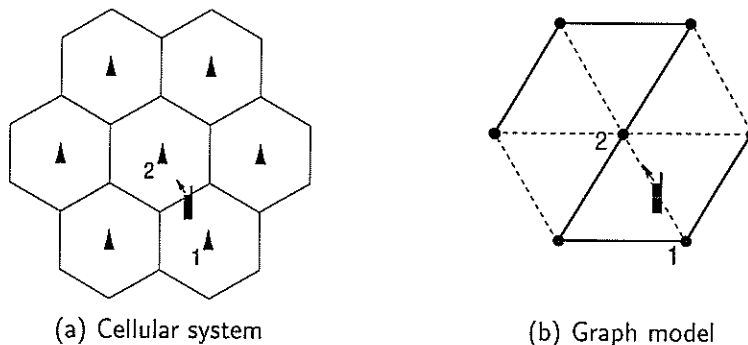(a) Cellular system   (b) Graph model

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

We also assume that a mobile unit moving between two MSCs can be modeled as being on a channel identical to messages in transit. In this manner, we no longer differentiate between physical movement and wired communication. It is reasonable to ask what happens when messages and mobile units are found on the same channel. We make the assumption that all channels preserve message ordering, i.e., they are FIFO channels. This appears to require that mobile units travel through space and reconnect to the next support center as fast or faster than messages can be transmitted across the network. Although flush primitives [2] can be used to make traditional non-FIFO channels FIFO, the separate channels used for mobile units and messages makes such flush primitives inapplicable to mobility. The FIFO behavior, however, can be realized by integrating the handover protocol with message passing.

For instance, the AMPS standard for cellular communication [9] describes a handover protocol which defines a sequence of wired messages between source and destination MSCs, as well as wireless communications with the
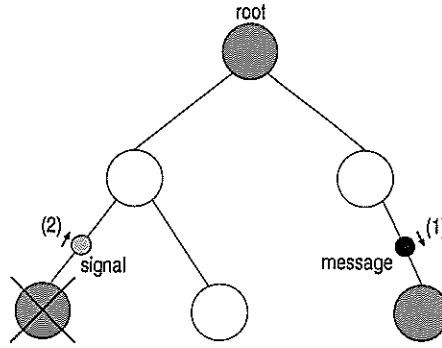
3

Figure 2: Dijkstra-Scholten for detecting termination of a diffusing computation. Shaded nodes are idle, white nodes are active.

mobile unit. By introducing a single additional wired message to this protocol, we can coordinate the wireless transfer of the mobile unit between the MSCs with the wired transfer of any messages, including data messages. Minor adjustments, must be made to both the sender and receiver to achieve this result, e.g., buffering messages until the mobile unit announces its arrival at the destination. No changes are forced on the behavior of the mobile unit. The details of this approach are available in [4]. Here, we emphasize that achieving FIFO behavior between mobile units and messages requires only trivial changes to existing handover protocols and therefore can be assumed as a network property in the remainder of this paper.

3 Applying diffusing computations to mobile unit tracking

Diffusing computations have the property that the computation initiates at a single root node while all other nodes are *idle*. The computation spreads as messages are sent from *active* nodes. Dijkstra and Scholten [3] describe an algorithm for detecting termination of such computations. The basic idea is that of maintaining a spanning tree that includes all active nodes, as shown in Figure 2. A message sent from an active node to an idle node (message (1) in Figure 2) adds the latter to the tree as a child of the former. Messages sent among tree nodes have no effect on the structure but may activate idle nodes still in the tree. An idle leaf node can leave the tree at any time by notifying its parent (signal (2) in Figure 2). Termination is detected when an idle root is all that remains in the tree.

We adapt this tree maintenance algorithm to the mobile environment. A node is seen as active when the mobile unit is present. The resulting algorithm maintains a tree identical to Figure 2 with the mobile unit at an active node or on a channel leaving a tree node. This enables us to guarantee the continued existence of a path from the root to the mobile unit along tree edges. We use this property to develop a guaranteed message delivery algorithm. The latter is superimposed on top of the graph maintenance algorithm. To maintain the distinction between the data messages being delivered and any control messages used to effect the delivery, we will refer to the data message as an *announcement*. In this section, we first describe the details of the graph maintenance algorithm, then present the guaranteed data message, i.e., announcement, delivery algorithm. A short discussion and possible extensions follow.

3.1 Mobile tracking

Although the Dijkstra-Scholten algorithm can be easily described and understood, the distributed message passing nature of the algorithm leads to subtle complexities. The details of the algorithm can be found in Figure 3. Each action is one atomic step and we assume weak fairness in action selection. For the purposes of discussion, we

4

```
┌─────────────────────────────────────────────────────────────────────────────────────────┐
│ State                                                                                     │
│ MobileAt_A            boolean, TRUE if mobile unit at A, initially FALSE except at root    │
│ Parent(A)             the parent of node A, initially NULL                                 │
│ Children(A)           multiset of children of node A, initially ∅                          │
│                                                                                           │
│ Actions                                                                                   │
│ MOBILEARRIVES_A(B) ;arrival at A from B      SENDMOBILE_A(B) ;mobile moves from A to B      │
│     Effect:                                      Preconditions:                            │
│         MobileAt_A := TRUE                           MobileAt_A and channel (A,B) exists    │
│         if Parent(A) ≠ NULL then                 Effect:                                   │
│             send signal(A) to B                      MobileAt_A := FALSE                    │
│         else                                         Children(A) := Children(A) ∪ {B}       │
│             Parent(A) := B                                                                 │
│                                              CLEANUP_A(B) ;remove node A from tree           │
│ SIGNALARRIVES_A(B) ;arrival at A from B          Preconditions:                            │
│     Effect:                                          Children(A) = ∅ ∧ ¬MobileAt_A          │
│         Children(A) := Children(A) − {B}             Parent(A) = B                          │
│                                                  Effect:                                   │
│                                                      send signal(A) to B                   │
│                                                      Parent(A) := NULL                      │
└─────────────────────────────────────────────────────────────────────────────────────────┘
```

Figure 3: Diffusing computations adapted for tracking a mobile unit

assume that the mobile unit is initially located at the root and moves nondeterministically throughout the graph (Figure 3, operation $\text{SENDMOBILE}_A(B)$).

In the introduction of this section, we described an algorithm which maintains a tree structure with edges from parent to child. By the distributed nature of the environment, the sender of a message cannot know whether or not the destination node is already in the tree, and cannot know whether or not to add the destination as a child. Therefore, the tree structure is maintained with edges from child to parent (recorded in *Parent(A)* in Figure 3).

For detecting termination and removing nodes from the tree, a node must be able to detect when it is an idle leaf node. This is done by tracking each message sent by each node. The Dijkstra-Scholten algorithm requires that every message be acknowledged by the destination with a *signal*. If the message arrives and the destination node is already part of the tree, the spanning tree topology does not change and the *signal* is sent immediately. Otherwise the *signal* is delayed and sent when the destination node removes itself from the tree. The source node tracks all messages by destination in a multiset or bag. Nodes in this bag indicate children nodes of the spanning tree, nodes to which the message has not arrived, or nodes from which the signal has been sent but not yet received. When the bag is empty, the node has no children and can remove itself from the tree by signaling its parent. For detecting termination of a diffusing computation, it is only necessary to keep a count of the number of successors. Because we intend to use this information during announcement delivery, we must maintain the bag of children.

Similar processing must occur in the mobile setting. Each movement of the mobile unit is tracked in a multiset (e.g., *Children(A)*). An element is removed from this multiset when the node receives a signal (Figure 3, operation SIGNALARRIVES). A signal is sent immediately when the mobile unit arrives and the node is already part of the tree (Figure 3, MOBILEARRIVES) and is delayed otherwise. A delayed signal is released when the node becomes a leaf to be removed from the tree (Figure 3, CLEANUP).

3.2   Superimposing announcement delivery

Having described the graph maintenance algorithm, we now present an algorithm to guarantee at-least-once delivery of an announcement. The details of this are shown in Figure 4 as actions superimposed on the graph maintenance actions of Figure 3. Actions with the same label execute in parallel while new actions are fairly interleaved with

State
⟨same as before⟩
AnnouncementAt$_A$ boolean, true if announcement stored at A, initially false everywhere
started   boolean, true if delivery has started, initially false
Actions

MOBILEARRIVES$_A$(B) *;arrival at A from B*
    Effect:
        ⟨same as before⟩
        if AnnouncementAt$_A$ **then**
            deliver announcement
            send *ack* to Parent(A) and children(A)

SIGNALARRIVES$_A$(B) *;arrival at A from B*
    ⟨same as before⟩

SENDMOBILE$_A$(B) *;moves from A to B*
    ⟨same as before⟩

ANNOUNCEMENTARRIVES$_A$(B) *;arrival at A from B*
    Effect:
        if Parent(A) = B **then**
            if MobileAt$_A$ **then**
                deliver announcement
                send *ack* to B
            else
                AnnouncementAt$_A$:= TRUE *;save ann.*
            send *announcements* to children(A)

CLEANUP$_A$(B) *;remove node A from tree*
    Preconditions:
        ⟨same as before⟩
    Effect:
        ⟨same as before⟩
        AnnouncementAt$_A$:= FALSE *;delete ann.*

ACKARRIVES$_A$(B) *;arrival at A from B*
    Effect:
        if Parent(A)=B ∨ B ∈ Children(A)
        if AnnouncementAt$_A$ **then**
            AnnouncementAt$_A$:= FALSE *;delete ann.*
            send *acks* to children(A) except B

ANNOUNCEMENTSTART *;root sends announcement*
    Preconditions:
        started = FALSE
    Effect:
        started := TRUE
        if MoibleAt$_{root}$ = TRUE **then**
            deliver announcement
        else
            AnnouncementAt$_{root}$:= TRUE
            send *announcement* to children(root)

Figure 4: Announcement delivery on top of diffusing computations.

the existing actions.

For announcement delivery we assume that the announcement originates at the root and we rely on the property that there is always a path from the root to the mobile unit alongedges in the tree. We note that the reverse edges of the tree (from parent to child) are a subset of the edges from parent to child maintained as successors of the parent (e.g., *Children(A)*). It is only necessary to send the announcement along edges in the spanning tree. But, because this tree is maintained with pointers from child to parent, the announcement must be propagated along the successor edges, from parent to child. When an announcement arrives from a source other than the parent, the announcement is rejected (Figure 4, ANNOUNCEMENTARRIVES). In this manner, the announcement is only processed along the tree paths. Effectively, a frontier of announcements sweeps through the spanning tree. When the announcement and the mobile unit are co-located at a node, the announcement is delivered (Figure 4, ANNOUNCEMENTARRIVES, MOBILEARRIVES).

In a stable environment where the mobile unit does not move, this announcement passing is sufficient to guarantee delivery. However if the mobile unit moves from a node in the tree below the frontier to a node above the frontier, delivery may fail. Therefore, each node stores a copy of the announcement until delivery is complete or the node is removed from the tree (Figure 4, ANNOUNCEMENTARRIVES). Storing the announcement in this manner ensures that the mobile unit cannot move to a region above the frontier without receiving a copy of the announcement. Because there is always a path from the root to the mobile unit, there must be an announcement on the frontier traversing this path and the announcement will eventually reach the mobile unit thus leading to delivery (Figure 4, MOBILEARRIVES). This path may change as the mobile unit moves from one region of the tree to another, however, the existence of a path is guaranteed by the graph maintenance algorithm presented in the previous section and the existence of the announcement on this path is guaranteed by the delivery algorithm of this section.

In the worst case, it is possible for the mobile unit to continuously travel with the announcement on the channel exactly one step behind. Eventually the mobile unit must either stop moving when the maximum length path is reached (equal to the number of nodes in the system), or the mobile unit will return to a previously visited tree node. When the mobile unit returns to a tree node, which, by the assumptions, must be above the frontier of announcements, it will receive the announcement stored there.

Storing the announcement requires an additional cleanup phase to remove all copies. When the mobile unit receives the announcement, an acknowledgement is generated and sent along the successor and parent edges (Figure 4, ANNOUNCEMENTARRIVES, MOBILEARRIVES). As before, the acknowledgment is rejected along paths which are not part of the tree (Figure 4, ACKARRIVES). The connectivity of the tree ensures that the acknowledgement will propagate to all nodes holding copies of the announcement. Leaf nodes being removed from the tree must also delete their copy of the announcement (Figure 4, CLEANUP).

This algorithm ensures at-least-once delivery of the announcement. Because the announcement copies remain in the graph until an acknowledgment is received, it is possible for the mobile unit to move from a region where the acknowledgments have propagated to a region where they have not. When this occurs, the mobile unit will receive an additional copy of the announcement, which it can reject based on sequence numbers. It is important to note that each time delivery occurs, a new set of acknowledgments will be generated. It can be shown that these acknowledgments do not inhibit the clean up process, but rather lead to a faster clean up. Each set of acknowledgments spreads independently through the tree removing announcement copies, but terminates when a region without announcement copies is reached.

3.3 Discussion

By superimposing the delivery actions on top of the graph maintenance, the result is an algorithm which guarantees at least once delivery of an announcement while actively maintaining a graph of the system nodes where the mobile

unit has recently traveled.

It is not necessary for the spanning tree be pruned as soon as an idle leaf node exists. Instead this processing can be delayed until a period of low bandwidth utilization. An application may benefit by allowing the construction of a wide spanning tree within which the mobile units travels. Tradeoffs include shorter paths from the root to the mobile unit versus an increase in the number of nodes involved in each announcement delivery.

By constructing the graph based on the movement of the mobile unit, the path from the root to the mobile unit may not be optimal. Therefore, a possible extension is to run an optimization protocol to reduce the length of this path. Such an optimization must take into consideration the continued movement of the mobile unit as well as any announcement deliveries in progress. The tradeoff with this approach is between the benefit of a shorter route from the root to the mobile unit and the additional bandwidth and complexity required to run the optimization.

Although in our algorithm only one mobile unit is present, the graph maintenance algorithm requires no extensions to track a group of mobile units. The resulting spanning tree can be used for unicast announcement delivery without any modifications and for multicast announcement delivery by changing only the announcement clean up mechanism. As presented, the delivery of the announcement triggers the propagation of acknowledgments. In the multicast case, it is possible for the announcement not to reach all mobile units before the cleanup starts. One practical solution is to eliminate the cleanup rules entirely, and assign a timeout to the announcement. This timeout should be proportional to the time it takes for the announcement to traverse the diameter of the network.

4   Backbone

We now introduce a new tracking and delivery algorithm inspired by the previous investigation with diffusing computations. Our goal is to reduce the number of nodes to which the announcement propagates, and to accomplish this we note that only the path between the root and mobile unit is necessary for delivery. In the previous approach, although the parts of the graph not on the path from the root to the mobile unit can be eliminated with *remove* messages, announcements still propagate unnecessarily down these subtrees before the node deletion occurs. To avoid this, the algorithm presented in this section maintains a graph with only one path leading away from the root and terminating at the mobile unit. This path is referred to as the *backbone*. The nodes in the remainder of the graph form structures referred to as *tails* and are actively removed from the graph, rather than relying on idle leaf nodes to remove themselves. Maintenance of this new structure requires additional information to be carried by the mobile unit regarding the path from the root, as well as the addition of a *delete* message to remove tail nodes. The announcement delivery mechanism remains essentially the same as before, but the simpler graph reduces the number of announcement copies stored during delivery.

To understand how the backbone is kept independent of the tails, we examine how the graph changes as the mobile unit moves. It is important to note that by the definition of the backbone, the mobile unit is always either at the last node of the backbone, or on a channel leading away from it. In Figure 5a, the backbone is composed of nodes A, B, and C and the dashed arrow shows the movement of the mobile unit from node C to D where D is not part of the graph. This is the most straightforwared case in which the backbone is extended to include D by adding both the child pointer from C to D (not shown) and the parent pointer in the reverse direction (solid arrow in Figure 5b).

In Figure 5b, the mobile moves to a node B, a node already in the backbone and with a non-null parent pointer. It is clear from the figure that the backbone should be shortened to only include A and B without changing any parent pointers, and that C and D should be deleted. To explicitly remove the tail created by C and D, a delete message is sent to the child of B. When C receives the delete from its parent, it will nullify its parent pointer, propagate the delete to its child, and nullify its child pointer.

If at this point the mobile moves from B onto D before the arrival of the delete (See Figure 5c), D still has a

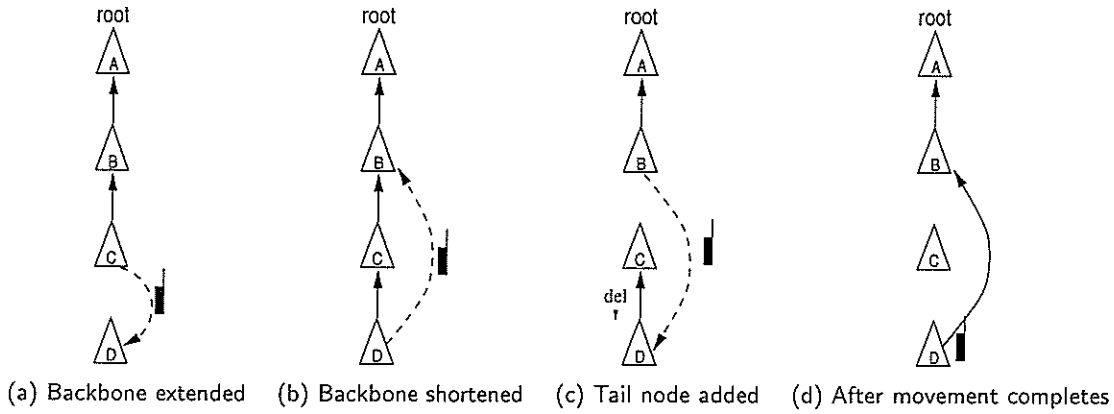(a) Backbone extended   (b) Backbone shortened   (c) Tail node added   (d) After movement completes

Figure 5: The parent pointers of the backbone change as the mobile moves to (a) a node not in the backbone, (b) a node higher in the backbone, and (c) a tail node. (d) shows the state after all channels have been cleared.
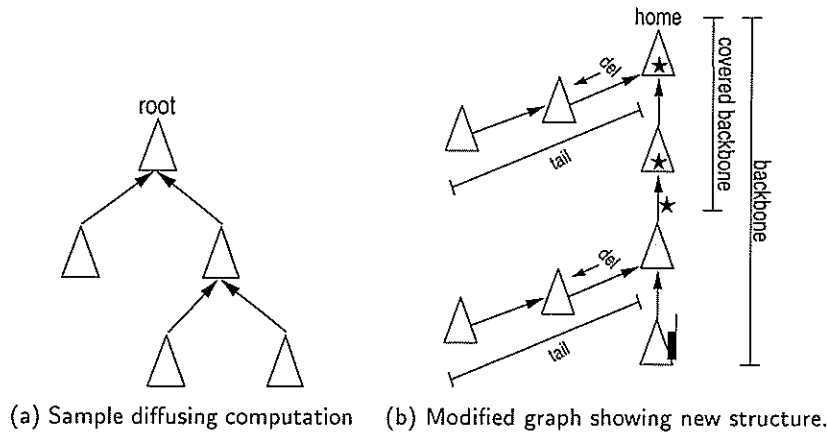

(a) Sample diffusing computation   (b) Modified graph showing new structure.

Figure 6: By adapting diffusing computations to mobility, we construct a graph reflecting the movement of the mobile. In order to deliver an announcement, the only part of the graph we need is the path from the root to the mobile, the *backbone*. Therefore we adapt the Dijkstra-Scholten algorithm to maintain only this graph segment and delete all the others.

parent pointer (C) and we cannot distinguish this case from the previous case (where B also had a non null parent pointer). In the previous case the parent of the node the mobile unit arrived at did not change, but in this case, we wish to have D's parent set to B (the node the mobile unit is arriving from) so that the backbone is maintained. To distinguish these two cases, we require the mobile unit to carry a sequence of the identities of the nodes in the backbone. In the first case where the mobile unit arrives at B, B is in the list of backbone nodes maintained by the mobile unit, therefore B keeps its parent pointer unchanged, but prunes the backbone list to remove C and D. However, when the mobile arrives at D, only A and B are in the backbone list, therefore the parent pointer of D is changed to point to B. But, what happens to the delete message moving from C to D? Because C is no longer D's parent when the delete arrives, it is simply dropped and the backbone is not affected.

The delivery algorithm is then superimposed on top of the generated graph. It is not sufficient to send the announcement down the spanning tree created by the backbone without keeping copies at all nodes along the path because the mobile is free to move from a region below the announcement to one above it (as in Figure 5b, assuming the announcement had propagated to C but not to D). Therefore, to guarantee delivery, as the announcement propagates down the backbone, a copy is stored at each node until delivery is complete. We refer to the portion of the backbone with an announcement as the *covered backbone*, see Figure 6b. Delivery can occur by the mobile unit moving to a location in the covered backbone, or the announcement catching up with the mobile unit at a

9

node. In either case, an acknowledgment is generated and sent via the parent pointers toward the root. If the announcement is delivered by the mobile unit moving on to the covered backbone, a delete is generated toward the child and an acknowledgment is generated toward the parent. Therefore any extra copies of the announcement on the newly created tail will be deleted with the nodes.

4.1 Details

The details for the tracking algorithm are shown in Figure 7. As before, we model arbitrary movement of the mobile by an action, called $\text{SENDMOBILE}_A(B)$, that allows a mobile at a node to move non-deterministically onto any outgoing channel.

MOBILEARRIVES shows the bulk of the processing and relates closely to the actions described in Figure 5. When the mobile unit arrives at a node, the changes to be backbone must be determined. If the mobile is doubling back onto the backbone, the parent pointers remain unchanged and the path carried by the mobile is shortened to reflect the new backbone (as in Figure 5b). If the node is not in the backbone (Figure 5a) or is part of a tail (Figure 5c), then the parent pointers must change to add this node to the backbone, and the node must be appended to the backbone list carried by the mobile. In any case, the children of this node (if any) are no longer necessary for announcement delivery, therefore a delete message is sent to the child, and the child pointer is cleared.

In addition to maintaining the graph, we must also address announcement delivery. As in the previous algorithms, when the mobile unit arrives at a node where the announcement is stored, delivery occurs, yielding *at-least-once* semantics for delivery. In this algorithm, we introduce a sequence number to ensure *exactly-once* delivery semantics. Therefore, when the mobile arrives at a node with the announcement, delivery is attempted if the sequence number of the last announcement received by the mobile is less than the sequence number of the waiting announcement. In all cases (whether or not delivery was just accomplished), at this point the announcement has been delivered to the mobile unit and an acknowledgment is generated along the path toward the root to clean up the announcement copies. No acknowledgment needs to be generated toward the tails because any announcement copies on tails will be removed at the same time the tail node is removed from the graph.

When the propagating announcement arrives at a node, ANNOUNCEMENTARRIVES, it is either arriving from a parent or some other node. If the announcement arrives from a node other than the parent, it should be discarded because to guarantee delivery the announcement need only propagate along the backbone. However, when an announcement arrives from the parent it must be processed. If the mobile is present, delivery is attempted with the same restrictions as before with respect to the sequence number and the acknowledgment is started toward the root. If the mobile is not present, the node stores a copy of the announcement in case the mobile arrives at a later time. Additionally, the announcement is propagated to the child link.

ACKARRIVES enables the cleanup of the announcements by propagating acks along the backbone toward the root via the parent pointers. Acks can also be present on tail links, but these are essentially redundant to the delete messages and do not affect the correctness of the algorithm.

The purpose of the delete messages is to remove the tail segments of the graph. Recall that a tail is created by a backbone node sending a delete to its child. Therefore, a delete should only arrive from a parent node. If we were to accept a delete from a non-parent node, as in the delete from C to D in Figure 5c, we could destroy the backbone. However if the delete arrives from the parent, we are assured that the node no longer resides on the backbone and should be deleted. Therefore, the arrival of a delete from a parent triggers the deletion of the stored announcement, the propagation of the delete to the child, and the clearing of both child and parent pointers.

<u>State</u>

AnnouncementAt_A	boolean, true if announcement stored at A, initially false everywhere
MobileAt_A	boolean, true if mobile unit at A, initially false except at root
Parent(A)	the parent of node A, initially NULL
Child(A)	the child of node A, initially NULL
started	boolean, true if delivery has started, initially false
MList	list of nodes carried by the mobile, initially contains only the root

<u>Actions</u>

ANNOUNCEMENTARRIVES$_A$(B) *;arrival at A from B*
    Effect:
        if Parent(A)=B **then**
            if MobileAt$_A$ **then**
                deliver announcement
                send *ack* to B
            else
                AnnouncementAt$_A$:=TRUE *;save ann.*
                send *announcement* to Child(A)

MOBILEARRIVES$_A$(B) *;arrival at A from B*
    Effect:
        MobileAt$_A$:=TRUE
        if A ∈ MList **then**
            A keeps old parent
            MList truncated after A to the end
            if AnnouncementAt$_A$ **then**
                deliver announcement
                Send *ack* to Parent(A)
                AnnouncementAt$_A$:=FALSE *;delete ann.*
        else
            Parent(A):=B
            MList := MList ∘ A
            AnnouncementAt$_A$:=FALSE *;delete ann.*
        send *delete* to Child(A)
        Child(A):=NULL

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

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

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

ANNOUNCEMENTSTART *;root initiates ann.*
    Preconditions:
        started = FALSE
    Effect:
        started:=TRUE
        if MoibleAt$_{root}$=TRUE **then**
            deliver announcement
        else
            AnnouncementAt$_{root}$:=TRUE
            send *announcement* to Child(root)

Figure 7: Tracking and delivery algorithm derived using some initial ideas from termination detection

4.2 Discussion and Generalizations

Keeping the backbone sequence is a similar methodology to routing protocols passing complete paths to the destination as in BGP [7] to avoid loops. It has been argued that keeping such information in the packet greatly increases its size. However, in our case, the information is being kept by the mobile unit and we assume there is sufficient storage on such a device for this additional information.

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

4.3 Correctness

Because this algorithm deviates significantly from the original Dijkstra-Scholten model of diffusing computations, essential properties necessary for announcement delivery are proven in this section: 1) announcement delivery is guaranteed, 2) after delivery announcement copies are eventually removed from the system, and 3) any tail node is eventually cleared. Although the third property is not essential to announcement delivery, it is necessary to show announcement cleanup.

Before approaching the proof, we formalize several useful definitions in Figure 8. The most important of these are the backbone, covered backbone, and tails. Intuitively, the backbone is the sequence of nodes starting at the root and terminating at either the node holding the mobile unit or the node the mobile unit just left if it is on a channel. The covered backbone is the sequence of backbone nodes with announcement copies. Tails are any path segments not on the backbone.

4.3.1 Announcement Delivery Guarantee

Our overall goal is to show at-least-once delivery of an announcement to a mobile unit. Therefore, the first property that we prove is (A) that from the state where no announcement exists in the system (*predelivery*), eventually a state is reached where the mobile unit has a copy of the announcement(*postdelivery*)[2]:

$$\text{predelivery} \mapsto \text{postdelivery} \tag{A}$$

$$\tag{A}$$

Although it is possible to make this transition in a single step (by executing ANNOUNCEMENTSTART while the mobile unit is at the root) it is more common for the system to move into an intermediate state where delivery is in progress (A.1). We must show that from this state (*delivery*), either the announcement will be delivered, or, in the worst case, the covered backbone will increase in length to include every node of the system (A.2). Once this occurs, delivery is guaranteed to take place when the mobile unit arrives at any node (A.3).

$$\text{predelivery } \textbf{ensures } \text{delivery} \vee \text{postdelivery} \tag{A.1}$$

$$\text{delivery} \mapsto \text{postdelivery} \vee (\text{delivery} \wedge \langle \exists \alpha :: \text{coveredBone}(\alpha) \wedge \langle \forall n : n \in N :: n \in \alpha \rangle \rangle) \tag{A.2}$$

$$\text{coveredBone}(\alpha) \wedge \text{delivery} \wedge \langle \forall n : n \in N :: n \in \alpha \rangle \textbf{ ensures } \text{postdelivery} \tag{A.3}$$

---

[2]Progress properties are expressed using the UNITY relations $\mapsto$ (read *leads-to*) and ensures. Predicate relation $p \mapsto q$ expresses progress by requiring that if, at any point during execution, the predicate p is satisfied, then there is some later state where q is satisfied. Similarly, p **ensures** q states that if the program is in a state satisfying p, it remains in that state unless q is established, and, in addition, it does not remain forever in a state satisfying p but not q.

D.1	$\text{reachable}(m,n) \equiv m = n$ $\lor\ (\dots)$ $\lor\ (\exists m' :: \text{reachable}(m,m') \land \text{reachable}(m',n))$[1]	Node n is reachable from node m if there is a path from m to n where every channel path has parent and child pointers of the channel endpoints pointing toward one another.
D.2	$\text{path}(p,n) \equiv n \in p$ $\land\ \langle \forall m : m \in p :: \text{reachable}(m,n) \lor \text{reachable}(n,m)\rangle$ $\land\ \langle \forall i,j : 1 \le i,j \le |p| \land i \ne j :: p_i \ne p_j\rangle$	Path p includes node n and is an acyclic sequence of reachable nodes.
D.3	$\text{maxpath}(p,n,R) \equiv \text{path}(p,n)$ $\land\ \langle \forall \dots \rangle$ $\land\ \langle \forall \dots \rangle$ $\land\ R(p)$	Path p is the maximal length path including node n that satisfies the predicate R. Extending p in either direction through concatenation ($\circ$) either violates the path relationships or the condition R.
D.4	$\text{backbone}(p) \equiv \text{maxpath}(p, \text{root}, \langle \dots m \dots \ne \text{MOB} \dots \rangle)$	Path p is the backbone, i.e. the path of maximal length which includes the root and does not include the mobile unit or any channel. The constant MOB is used to identify the mobile unit.
D.5	$\text{coveredBone}(p) \equiv \text{maxpath}(p, \text{root}, \langle \dots \rangle)$	Path p is the covered backbone, i.e. the maximal length path including the root (the backbone) where all nodes contain announcement copies.
D.6	$\text{tail}(p,n) \equiv \text{maxpath}(p, n, \langle \forall m : m \in q :: m \ne \text{root}\rangle)$	The tail is the maximal length path of any node n where no node on the path is part of the backbone.

Figure 8: Useful definitions.

We approach each of these properties in turn, first showing that from *predelivery*, either *delivery* or *postdelivery* must follow (A.1). Until the action ANNOUNCEMENTSTART fires, the system remains in *predelivery* and ANNOUNCEMENTSTART remains enabled. Trivially, when it fires, either the announcement will be delivered (if the mobile unit is present at the root) or the announcement will begin to propagate through the system.

Once the delivery state is reached, we must show that the covered backbone will increase in length to include all nodes or the announcement will be delivered (A.2). To do this, we strengthen the progress property A.2 to state that the covered backbone cannot decrease in length.

$$\text{delivery} \land \text{coveredBone}(\alpha) \land k = |\alpha| < N$$
$$\mapsto (\text{delivery} \land \text{coveredBone}(\alpha) \land |\alpha| > k) \lor \text{postdelivery} \qquad \text{(A.2.1)}$$

In order to formally make this assertion, we must first show that during delivery the covered backbone exists. Showing the existence of the covered backbone independent from other system attributes is not possible. Therefore we prove a stronger invariant that not only establishes the existence of the covered backbone, but also the existence of the backbone and the relationship between the two. By definition, the covered backbone is a subset of the backbone. We further assert that if the covered backbone is shorter than the backbone, there is an announcement leaving the last node of the covered backbone (where last(α) returns the final element of the path α). Alternately, if the covered backbone and backbone are equivalent, the mobile unit must precede the announcement (indicated by the constant ANN) in the channel leaving the last node.

13

$$\text{delivery} \Rightarrow \langle \exists \alpha, \beta, f : \text{backbone}(\beta) \wedge \text{coveredBone}(\alpha) \wedge f = \text{last}(\alpha) ::$$
$$(\alpha \subset \beta \wedge \text{ANN} \in \text{Chan}(f, \text{Child}(f)))$$
$$\vee \ (\alpha = \beta \wedge \text{mobile.preceeds.ann}(f, \text{Child}(f)))\rangle \tag{I.1}$$
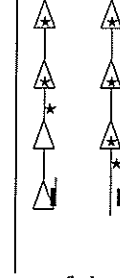
This invariant is proven by showing that it holds initially as well as over all statements of the program. Throughout this proof, we use several supporting properties which appear in Appendix A. Specifically: Inv I.1.1 the integrity of the backbone, Inv I.1.2 that the backbone always exists, Inv I.1.3 that there is at most one announcement in a channel, Inv I.1.4 that there are no announcements during predelivery, and Inv I.1.5 that there are no acknowledgments during delivery. We now show the proof of the top level property concerning the existence of the covered backbone during delivery (I.1):

- It is trivial to show the initial conditions of I.1 because initially, *delivery* is false.

- MOBILEARRIVES$_A(B)$: We assume the integrity of the backbone (Inv I.1.1). First we consider the case where the system is in delivery and the right hand side of this invariant (I.1) holds. The covered backbone is not affected if the mobile unit arrives at a non-backbone node or a backbone node below the covered backbone. If the mobile unit arrives at a covered backbone node, the announcement is delivered and the invariant is trivially true by falsifying the left hand side.

  Next we consider when the system is not in delivery. If the system is in predelivery and we assume there are no announcements during predelivery (Inv I.1.4), the movement of the mobile unit cannot affect the delivery status. Once the system is in postdelivery, it cannot return to delivery, so the invariant remains true.

- ANNOUNCEMENTARRIVES$_A(B)$: We assume there is at most one announcement on a channel in the system (Inv I.1.3). Therefore, if the system is in delivery and we assume this invariant is true before the announcement arrives, the announcement must be leaving the covered backbone. Further, since the announcement is at the head of the channel, it cannot be the case that the mobile unit and announcement are in the same channel, so the covered backbone must be a proper subsequence of the backbone. Therefore, by the definitions of the covered backbone and backbone, the node the announcement arrives at is on the backbone, and either the announcement is delivered or is propagated.

  If delivery occurs, this invariant is trivially satisfied by falsifying the delivery condition.

  If the announcement is propagated into the next channel, then the covered backbone is extended by one node which has already been shown to be part of the backbone. The announcement is put onto the child link of this node, which by the backbone definition must be a channel on the backbone or backbone extension. The announcement must follow the mobile unit if the mobile unit is on the same channel.

  As before, if the system is not in delivery, then the delivery status of the system cannot change with the execution execution of this statement.

- SENDMOBILE$_A(B)$: Before the statement executes, the mobile unit must be at a node, otherwise the statement is a skip. Since we assume this invariant to be true, it must be the case that the covered backbone is a proper subsequence of the backbone. Therefore, when the mobile unit leaves the node, the backbone is only changed to include the new backbone extension, the covered backbone is not affected, and the invariant remains true.

- ANNOUNCEMENTSTART: We assume that if the system is in predelivery, there are no announcements in the system (Inv I.1.4). Therefore after this statement executes, either delivery occurs and Inv I.1 invariant is

trivially true, or the announcement is placed at the root and on the outgoing link, establishing the right hand side of the invariant. If the system is in delivery or postdelivery, this statement is a skip.

- AckArrives$_A(B)$: We assume there are no acknowledgments in the system during delivery (Inv I.1.5), and therefore this statement is essentially a skip during delivery. This statement cannot change the delivery status, therefore, if the system is in predelivery or postdelivery, the invariant is trivially true.

- DeleteArrives$_A(B)$: We assume that delete messages do not affect the backbone (Inv I.1.1), therefore they will not affect the covered backbone, and this invariant will remain true. As before, this statement cannot change the delivery status.

This concludes the proof that during delivery, the covered backbone exists. We now show that the covered backbone must grow, as defined by property A.2.1. We note two specific cases that the system can be in with respect to the mobile unit and the announcement and show how either the covered backbone must increase or delivery will occur. The first case is where *the mobile unit and announcement are not on the same channel*. Since the system is in delivery, there cannot be an acknowledgment on the channel (Inv I.1.5). Since the announcement is on a backbone channel, there cannot be a delete on the channel (Inv I.1.1). The assumption is that the mobile unit is not on the same channel. This covers all possible message types that could precede the announcement on the channel, therefore the announcement must be at the head of the channel. So, in this case, the progress property A.2.1 which concerns the growth of the covered backbone becomes an ensures because the announcement will remain at the head of the channel until processed, lengthening the covered backbone, or the mobile unit will arrive at a node causing delivery. In either case, the condition on right hand side becomes true.

In the second case, *the mobile unit and announcement are on the same channel*. By Invariant I.1, the mobile unit precedes the announcement in this channel. We state a trivial progress property that if the mobile unit is at the head of a channel, it is ensured to arrive at the destination node:

$$\text{mobile.at.head}(m, n) \text{ ensures MobileAt}(n) \tag{A.2.1.1}$$

Because there is only one mobile unit, after the mobile unit is removed from the channel, either the system is taken out of delivery by the mobile unit receiving the announcement, or the system has been reduced to the first case where the mobile unit and announcement are not on the same channel.

The previous discussion effectively shows property A.2.1, namely that the covered backbone must grow until all nodes in the system are part of the covered backbone or delivery has occurred. To complete the proof that delivery is guaranteed, we need to show that when all nodes are part of the covered backbone, delivery *must* occur. By the definitions of the covered backbone and backbone, when all nodes are part of the covered backbone, the two are equivalent. The mobile unit must be on a channel because all nodes have announcement copies and if the mobile unit is at a node, it must have received the announcement copy (either when the mobile unit arrived or when the announcement arrived). The destination of the channel the mobile unit is on must be part of the backbone because all nodes are part of the backbone. If there is a delete in front of the mobile unit, it will not have any effect on the backbone (Inv I.1.1). There cannot be an acknowledgment in the channel (Inv I.1.5). The announcement must be behind the mobile unit (Inv I.1.1). Therefore, after the delete (if any) is processed, the mobile unit is at the head of the channel. The MobileArrives$_A(B)$ action will cause delivery. Therefore, announcement delivery is guaranteed from the initial state of the system.

4.3.2 Backbone announcements cleaned up

Once the announcement has been delivered, we show that eventually all stored announcement copies are removed. There are two cases to address: the announcements on nodes on the backbone and those not on the backbone.

In the next section, we will show how all nodes which are not part of the backbone will be cleaned up, while this section focuses on the cleanup of the backbone nodes. In particular, we wish to show that after the announcement has been delivered, eventually all announcement copies on the backbone will be deleted.

$$\text{postdelivery} \mapsto \langle \forall m, \beta : \text{backbone}(\beta) \wedge m \in \beta :: \neg \text{AnnouncementAt}(m) \rangle \tag{B}$$

We introduce a safety property describing the state of the backbone in postdelivery. Namely, (a.) the backbone and covered backbone exist, (b.) there is an acknowledgment in the channel heading toward the last node of the covered backbone, (c.) all nodes in the backbone not in the covered backbone do not have announcement copies, (d.) there are no announcement copies on any backbone channels or the backbone extension, and (e.) there are no acknowledgments on the channels of the covered backbone. Intuitively, this invariant shows that there is only one segment of the backbone with announcement copies and the nodes on this segment are poised to receive an acknowledgment.

$$\text{postdelivery} \Rightarrow \langle \exists \alpha, \beta : \text{backbone}(\beta) \wedge \text{coveredBone}(\alpha) \wedge f_\alpha = \text{last}(\alpha) \wedge f_\beta = \text{last}(\beta) ::$$
$$\alpha \subset \beta \wedge \text{ACK} \in \text{Chan}(\text{Child}(f_\alpha), f_\alpha)$$
$$\wedge \langle \forall m : m \in (\beta - \alpha) :: \neg \text{AnnouncementAt}(m) \rangle$$
$$\wedge \langle \forall m, n : m, n \in \beta \wedge m = \text{Parent}(n) :: \text{ANN} \notin \text{Chan}(m, n) \rangle$$
$$\wedge \text{MOB} \in \text{Chan}(f_\beta, \text{Child}(f_\beta)) \Rightarrow \neg \text{mobile.preceeds.ann}(f_\beta, \text{Child}(f_\beta))$$
$$\wedge \langle \forall m, n : m, n \in \alpha \wedge n = \text{Child}(m) :: \neg \text{ACK} \in \text{Chan}(n, m) \rangle \tag{I.2}$$

We now show the proof of this statement by showing that if it holds before the execution of each statement, it must hold after the execution of the statement:

- MOBILEARRIVES$_A(B)$: When the mobile unit arrives at a non-backbone node, the backbone is extended to include this node. The channel just traversed will become part of the backbone, but will not have an announcement on it by the last part of this invariant. The covered backbone will not change. If there is an announcement at this node, it will be removed so that there are still no announcement copies at nodes other than the covered backbone.

  If the mobile unit arrives at a backbone node that is not part of the covered backbone, the covered backbone does not change. There are still no announcements at backbone nodes other than the covered backbone, and because no new channels are added to the backbone, there are no announcement copies on the channels of the covered backbone.

  If the mobile unit arrives at a backbone node that is part of the covered backbone, the covered backbone is shortened to be all nodes above this new location of the mobile unit. Each of these nodes must have an announcement copy because they were part of the covered backbone before the mobile unit arrived. Also, an acknowledgment is generated in the channel heading toward the new covered backbone and this invariant is established. As before, there are no new channels in the backbone, so there are still no announcements on any backbone channels.

  We must also consider the case where the postdelivery condition is established by the arrival of the mobile unit. The components of this invariant are established because the only announcement in the system was at the end of the covered backbone which must be downstream from where the mobile unit arrived, so there are no announcements in backbone channels. The remainder of the invariant is established in a manner similar to the case where the system is in postdelivery and the mobile unit arrives at a covered backbone node.

16

- ANNOUNCEMENTARRIVES$_A(B)$: If the system is in delivery, the arrival of the announcement could establish postdelivery. In this case, the components of this invariant are established because all nodes above the mobile unit are part of the covered backbone. The acknowledgment is put into the channel above the mobile unit, which is the end of the covered backbone. The announcement copy at the node the mobile unit is at is deleted.

  If the system is in postdelivery and the announcement arrives, the announcement could arrive at a backbone node only from a non-backbone channel and will be dropped. Therefore, the invariant will not be affected because neither the backbone nodes nor links are affected.

- SENDMOBILE$_A(B)$: If the mobile unit leaves a node, the covered backbone does not change. Also, the mobile unit is at the end of the channel, so any announcements in the same channel must be before the mobile unit.

- DELETEARRIVES$_A(B)$: The arrival of a delete at a backbone node will not affect the backbone or the covered backbone. The arrival of a delete elsewhere in the system will not affect this invariant.

- ACKARRIVES$_A(B)$: If an acknowledgment arrives at a covered backbone node, it must be at the end of the covered backbone. Therefore, the processing of this acknowledgment will shrink the covered backbone by one node and put the acknowledgment farther up in the backbone. Alternately, the root could receive the acknowledgment and there would no longer be a covered backbone.

  If an acknowledgment arrives at a non-backbone node and is accepted, it will not be put onto the backbone. If it is not accepted, nothing changes in the covered backbone or backbone, therefore the invariant is maintained.

- ANNOUNCEMENTSTART: This statement has no effect during postdelivery. During predelivery, this statement could establish this invariant by delivering the announcement to a node at the root. In this case, the covered backbone does not exist, and the invariant is true.

Our goal is to show progress in the cleanup of announcements on the covered backbone. To do this, we use a progress metric that measures the reduction in length of the covered backbone. Because the only nodes on the backbone with announcement copies must be on the covered backbone by Invariant I.2, once the covered backbone has length zero, all announcements on the backbone have been deleted.

$$\text{postdelivery} \land \text{coveredBone}(\alpha) \land |\alpha| = k \geq 1 \mapsto |\alpha| < k \tag{B.1}$$

To prove this statement, we note that by the previous invariants, it has been established that there is an acknowledgment in the channel heading toward the covered backbone. If the acknowledgment is not at the head of the channel, then there must be something else in front of it. There cannot be a delete on the channel, because that would mean there is a delete on the backbone which is not allowed by invariant I.1.1. An announcement would have no effect because it is not arriving on a parent link. If the mobile unit were on the channel, then the arrival of the mobile unit would cause delivery because the announcement must be at the last node of the covered backbone, and the covered backbone would change.

So, either the mobile unit will arrive from the same channel as the acknowledgment or on another channel and will cause the covered backbone to shrink, or the acknowledgment will be processed and cause the covered backbone to shrink. Since there are only a finite number of messages in the channel in front of the acknowledgment, these will be processed and eventually either the acknowledgment will reach the head of the channel or the covered backbone will shrink in another way (through the arrival of the mobile unit at a covered backbone node).

When the covered backbone shrinks to zero length, there will be no more announcement copies on any backbone nodes, accomplishing backbone cleanup.

4.3.3 Tail Cleanup

In addition to backbone cleanup, we must also ensure that any announcement copies not on the backbone will eventually be deleted. More precisely, any node which is on a tail will eventually be cleared or put on the backbone (C), where clear(n) indicates that n's parent and child pointers are NULL (which will be shown to imply the announcement is no longer stored there). Since a node can only accept an announcement from a parent, this implies that only nodes with non-null parent pointers could have announcement copies. Since a node can only clear its pointers at the same time as it clears its storage, there is no way for a node (other than the root) to have a copy of the announcement and non-null pointers.

We cannot guarantee that the mobile unit will eventually arrive at the node thereby adding that node to the backbone, we must prove that there is a delete message that will eventually arrive at the node if it remains on the tail. We show that every tail has a delete message on the channel heading toward the first node of the tail (I.3), where the *first* node of the tail is defined to be the node whose parent pointer points toward a node that does not point toward it as the child. This delete message will eventually be processed, shrinking the length of the tail (C.1). When the tail contains only node, the tail is guaranteed to be cleared (C.2).

$$\text{tail}(\tau, n) \mapsto \text{clear}(n) \vee \langle \exists \beta : \text{backbone}(\beta) :: n \in \beta \rangle \qquad (\text{C})$$

$$\langle \exists \tau : \text{tail}(\tau, n) \wedge (n \neq \text{first}(\tau)) \wedge (|\tau| = k) \wedge (|\tau| > 1) \rangle \mapsto \langle \exists \tau : \text{tail}(\tau, n) \wedge |\tau| < k \rangle$$
$$\vee \langle \exists \beta : \text{backbone}(\beta) :: n \in \beta \rangle \qquad (\text{C.1})$$

$$\langle \exists \tau : \text{tail}(\tau, n) \wedge |\tau| = 1 \rangle \textbf{ ensures } \text{clear}(n) \vee \langle \exists \beta : \text{backbone}(\beta) :: n \in \beta \rangle$$
$$(\text{C.2})$$

To show that the tail can shrink, we must guarantee the existence of the delete message at the end of the tail. We do this by assuming the invariant before each statement execution and showing it holds after statement execution.

$$\langle \forall n, \tau : \text{tail}(\tau, n) \wedge n = \text{first}(\tau) :: \text{DEL} \in \text{Chan}(\text{Parent}(n), n) \rangle \qquad (\text{I.3})$$

- MobileArrives$_A$(B): If the mobile unit arrives at a backbone node, one or zero tails are created. If no tails are created, the invariant trivially holds. If a tail is created, it consists of the nodes that are removed from the backbone. These nodes by definition point toward one another as parent and child, making them a tail. No other tails are affected. The new tail by definition has a first node. The first node of the new tail is the node formerly pointed to as the child of the node where the mobile unit is currently at. A delete is put onto this channel, establishing this invariant.

  If the mobile unit arrives at a node that is not on the backbone and not on a tail, no new tails are created, no deletes are sent, and no old tails are affected.

  If the mobile unit arrives at a tail node, the tail is cut into two segments around the mobile unit. The nodes above the mobile unit are not affected because the first node of the tail is still the same and the delete is not affected. The nodes below the mobile unit are similar to the first case, and the delete generated down the old child pointer establishes the invariant.

- DeleteArrives$_A$(B): If a delete arrives at any node on a link other than from the parent, this delete could not be critical to any tail, and therefore dropping it has no effect on the invariant.

  If a delete arrives at the first node of a tail along the parent link, the delete is propagated to the new first node of the tail and a node is removed from the tail.

- AnnouncementArrives$_A(B)$: SendMobile$_A(B)$: AckArrives$_A(B)$: AnnouncementStart do not affect the invariant

With this invariant, it is clear that when the delete is processed, a node is removed from the tail, and the tail shrinks (property C.1). If the delete is not at the head of the channel, the messages ahead of it must be processed. Neither an acknowledgment nor an announcement will affect progress. If a mobile unit arrives, the node is added to the backbone, satisfying the progress condition.

Finally, we formally define clear (D.7), then show that if a node is clear, it has no announcement copies (I.3.1):

$$\text{clear}(n) \equiv \text{Parent}(n) = \text{Child}(n) = \text{NULL} \tag{D.7}$$

$$\text{clear}(n) \Rightarrow \neg\text{AnnouncementAt}(n) \tag{I.3.1}$$

This invariant is easily shown over every statement. Intuitively, when a node sets both its parent and child pointers to NULL, as in DeleteArrives$_A(B)$, the announcement copies at the node are deleted. Since it is not possible to set the child and parent pointers to NULL any other way, and an announcement is only accepted from a non-NULL parent link, there cannot be an announcement at a node that has both NULL pointers.

Therefore, once a node is either clear or put back on the backbone, it will not have an announcement copy. As the tails shrink, we are guaranteed that the announcements not on the backbone will be removed from the system.

5   Discussion

We have described two algorithms to guarantee the delivery of an announcement to a mobile unit with no assumptions regarding the speed of movement. In this section, we compare our approach with other tracking based delivery schemes designed for the mobile environment including Mobile IP, a scheme by Sony, another by Sanders et. al., and finally a multicast scheme by Badrinath et. al.

Each of these algorithms uses the notion of a home node toward which the announcement is initially sent. In Mobile IP [6], the home node tracks as closely as possible the current location of the mobile unit and all data is sent from the home directly to this location. This information is updated each time the mobile unit moves, introducing a discrepancy between the actual location and the stored location during movement. Any data sent to the mobile unit during this update will be dropped. Mobile IP has no mechanism to recover this data, but rather assumes that higher layer protocols such as TCP will handle buffering and retransmitting lost data. One proposal within Mobile IP is to allow the previous location of the mobile unit to cache the new location and forward data rather than dropping it. While this can reduce the number of dropped announcements, it still does not guarantee delivery as the mobile unit can continue to move, always one step ahead of the forwarded announcement.

One proposal is to increase the amount of correct location information in the system by distributing this information to multiple routers, as in our tree and backbone maintenance algorithms. The Sony [5] approach keeps the home node as up to date as possible, but also makes the other system routers active components, caching mobile unit locations. As the packet is forwarded, each router uses its own information to determine the next hop for the packet. During movement and updating of routing information, the routers closer to the mobile unit will have more up to date information, and fewer packets will be lost than in the Mobile IP approach. This approach still does not provide delivery guarantees, and few details are given concerning updates to router caches. One benefit is that announcements need not be sent all the way to the home node before being forwarded toward the mobile unit. Instead any intermediate router caching a location for the mobile unit can reroute the packet toward the mobile unit.

The Sanders approach [8] has this same advantage, allowing intermediate routers to forward the announcement. Sanders describes precisely how intermediate routers are updated. A hop by hop path is kept from the home node

to the last known location of the mobile unit. When the mobile unit moves, the path is shortened one node at a time until the common node between the shortest path to the old location and the shortest path to the new location is reached, then the path is extended one node at a time. Any announcements encountering the hop by hop path are forwarded toward the last node of this path. During the updating of this path, announcements move with the update message which are changing the path and will eventually reach the next known location of the mobile unit. However, the mobile unit may have moved during the update, in which case, the data messages will continue to travel with the next update message. Although no messages are dropped, the slow update time and ability of the mobile unit to keep moving could prevent delivery.

In each of these approaches, a single copy of the announcement is kept in the system, while our approach stores multiple copies throughout the system until delivery is complete. We believe that sacrificing this storage for the limited times that our algorithms require is worthwhile to provide guaranteed delivery of the announcement. If we weaken these requirements, our approaches can be modified to reduce storage. In the first algorithm, the announcement can be sent down the spanning tree in a wave. When the announcement arrives at a node, if the mobile unit is present, it is delivered, otherwise it is sent on all outgoing spanning tree channels. Although multiple copies are generated, they will not be stored, but simply passed to the next hop. When the announcement reaches a leaf node it will be dropped. If the mobile unit remains in a region of the graph below the announcement propagation, it will receive the message, however if it is in transit or moves to a region above the message propagation, the announcement will not be delivered. In our second approach, a single announcement copy can be sent down the backbone path. Even if the announcement ends up on a tail, it will continue toward the mobile unit. Because the path we define to the mobile unit is based on movement pattern rather than shortest path as in the Sanders algorithm, there is only one pathological movement pattern (a figure eight crossing the backbone) where the mobile unit can continue to avoid delivery.

Another approach which keeps multiple announcement copies is Badrinath's guaranteed multicast algorithm [1] which stores announcement copies at all system nodes until all mobile units in the multicast group have received the announcement. This information is gathered by the announcement initiator from the nodes that actually delivered the announcement. A disadvantage of this algorithm is that all recipients must be known in advance, a property not always known in multicast. Our first algorithm can trivially be extended to track the movement of multiple mobile units, and because it is based on the actual movement of the mobile units can reduce the number of nodes involved in the multicast delivery with respect to the Badrinath approach.

6   Conclusions

Our primary contribution in this work is the introduction of a new approach to the study of mobility, one based on a model whose mechanics are borrowed directly from the established literature of distributed computing. Treating mobile units as messages provides an effective means for transferring results from classical distributed algorithm literature to the emerging field of mobile computing. A secondary contribution is the development of two algorithms for message delivery to mobile units, the first a direct derivative of the diffusing computations distributed algorithm, and the second an optimizing refinement of the first based on careful study of the problem and the solution. Each algorithm is applicable in a variety of settings where mobile computing components are used and reliable communication is essential.
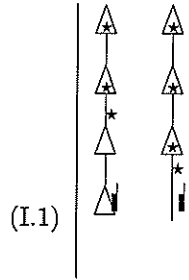
References

[1] A. Acharya, A. Bakre, and B.R. Badrinath. IP multicast extensions for mobile internetworking. Technical Report LCSR-TR-243, Rutgers University, 1995.

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

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

[4] A.L. Murphy, G.-C. Roman, and G. Varghese. A distributed snapshot algorithm adapted for message delivery to mobile units. Submitted to JPDC, January 1999.

[5] A. Myles and D. Skellern. Comparing four IP based mobile host protocols. *Computer Networks and ISDN Systems*, 26(3):349–355, 1993.

[6] C.E. Perkins. IP mobility support. Technical Report RFC 2002, IETF Network Working Group, October 1996.

[7] Y. Rekhter and T. Li. A Border Gateway Protocol 4 (BGP-4). *RFC 1771*, March 1995.

[8] B. Sanders, B. Massingill, and S. Kryukova. Derivation of an algorithm for location management for mobile communication devices. *Parallel Processing Letters*, 8(4):473–488, December 1998.

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

A Supporting Invariants

This appendix proves several supporting invariants needed to prove the existence of the covered backbone (Inv I.1):

$$
\begin{aligned}
\text{delivery} \Rightarrow \langle \exists \alpha, \beta, f : \text{backbone}(\beta) \wedge \text{coveredBone}(\alpha) \wedge f = \text{last}(\alpha) :: \\
(\alpha \subset \beta \wedge \text{ANN} \in \text{Chan}(f, \text{Child}(f))) \\
\vee \ (\alpha = \beta \wedge \text{mobile.preceeds.ann}(f, \text{Child}(f))) \rangle
\end{aligned}
\tag{I.1}
$$

A.1 Integrity of the Backbone

The first supporting invariant addresses the **integrity of the backbone**, stating that if the backbone exists, it must have the following properties: (a.) the sequence of nodes in the backbone must be the same as the list of nodes carried by the mobile unit (program variable MList), (b.) if the mobile unit is on a channel, that channel must be the backbone extension (defined as the channel segment between the last node of the backbone and the mobile unit), (c.) if the mobile unit is at a node (program variable MobileAt), that node must be the last node of the backbone sequence, (d.) there are no delete messages (indicated with the constant DEL) in any backbone channels, and (e.) there are no delete messages on the backbone extension.

$$
\begin{aligned}
\text{backbone}(\beta) \wedge \text{last}(\beta) = f \Rightarrow &\text{MList} = \beta \\
&\wedge \text{MOB} \in \text{Chan}(m, n) \Rightarrow m = f \wedge \text{Child}(m) = n \\
&\wedge \text{MobileAt}(m) \Rightarrow m = f \\
&\wedge \langle \forall m, n : m, n \in \beta \wedge n = \text{Child}(m) :: \text{DEL} \notin \text{Chan}(m, n) \rangle \\
&\wedge \neg \text{mobile.preceeds.delete}(f, \text{Child}(f))
\end{aligned}
\tag{I.1.1}
$$

Again, this invariant is proven over each of the program statements.

- MOBILEARRIVES$_A(B)$: If A is on the backbone, the MList is truncated after this node, none of the pointers change, and the backbone is maintained and matches the MList.

  If A is not on the backbone, A is appended to MList. The last node of the backbone is pointing toward A as its child by this invariant, therefore when A sets its parent pointer, the backbone is extended because the maximal length path now includes the link just traversed by the mobile unit.

  Since there were no delete messages on the backbone and the mobile unit was processed from the head of the channel, after this statement executes, there are still no delete messages on any channels. However, a delete is generated leaving A. Since the child of A is NULL, this channel cannot be part of the backbone because there is no path that includes it. Therefore the generated delete does not violate the invariant.

- DELETEARRIVES$_A(B)$: If the delete is accepted, A is not on the backbone because by this invariant, there are no delete messages that will affect backbone nodes. Therefore, changing the pointers of A does not affect the backbone or backbone extension. After the statement executes, A is still not part of the backbone, therefore the outgoing channel of A that now has a delete message is not part of the backbone.

  If the delete is not accepted, the statement is essentially a skip, which trivially maintains the invariant.

- SENDMOBILE$_A(B)$: The nodes of the backbone do not change, however the mobile unit is no longer on the node. Therefore the backbone extension is created and the last node of the backbone does point toward the mobile unit. Also, it is not possible for the mobile unit to precede a delete because the mobile unit is put onto the end of the channel, and by the FIFO assumption with the channels, the mobile unit follows everything that was in the channel.

- ANNOUNCEMENTARRIVES$_A(B)$, ACKARRIVES$_A(B)$, and ANNOUNCEMENTSTART do not affect any of the variables of this invariant.

A.2 Backbone always exists

To be able to assert the previous invariant at any point, we must show that the **backbone always exists**.

$$\langle \exists \beta :: \text{backbone}(\beta) \rangle \tag{I.1.2}$$

Because the root is a constant, the backbone can always be constructed. Note, this invariant does not say anything about the structure of the backbone, only that one can always be constructed. The structure invariant is found in I.1.1.

A.3 A most one announcement

At any time during program execution, there is **at most one announcement** in the system which is on a channel. There might be multiple copies of the announcement in the system, but these copies are at nodes.

$$\langle \sum m, n : \text{ANN} \in \text{Chan}(m, n) :: 1 \rangle \leq 1 \tag{I.1.3}$$

The intuition behind proof of this property is that during predelivery, there are no announcements in the system (on either nodes or channels) (Inv I.1.4). Once ANNOUNCEMENTSTART fires, either one announcement is put on a channel, or none (in the case where delivery occurs immediately). Since ANNOUNCEMENTSTART acts as a skip from this point forward, and no other statement can generate an announcement without consuming an announcement from a channel, there will never be more than one announcement on any channel.

A.4 No announcements during predelivery

To support the previous argument, we need the fact that there are **no announcements in the system during predelivery**.

$$\text{predelivery} \Rightarrow \langle \forall m, n :: \text{ANN} \notin \text{Chan}(m,n) \wedge \neg \text{AnnouncementAt}(m) \rangle \qquad (\text{I.1.4})$$

By the initial conditions, there are no announcements in the system. The only statement which can generate an announcement is ANNOUNCEMENTSTART and after this statement fires, the system is no longer in predelivery. The system cannot enter predelivery again because there is no statement to set predelivery to true.

A.5 No acknowledgements during delivery

We must also show that there are **no acknowledgments during delivery** (where the constant ACK indicates an acknowledgment).

$$\text{delivery} \Rightarrow \langle \forall m, n :: \neg \text{ACK} \in \text{Chan}(m,n) \rangle \qquad (\text{I.1.5})$$

It is trivial to show that there are no acknowledgments initially, and none are generated during predelivery. the first acknowledgment is generated when the mobile unit receives the announcement, which by definition takes the system out of into postdelivery, making this invariant trivially true.