Term Paper on
Dependency Sequences and Hierarchical Clocks:
Efficient alternatives to Vector Clocks for Mobile
Computing Systems

Mukesh Singh Rawat (05329020)
M. Tech I Year
Under the Guidance of Prof. Umesh Bellur
KReSIT, Indian Institute of Technology Bombay
April 5, 2006

Abstract

Vector clocks have been used to capture causal dependencies between processes in distributed computing systems. Vector clocks are not suitable for mobile computing systems due to (i) lack of scalability: its size is equal to the number of nodes, and (ii) its inability to cope with fluctuations in the number of nodes. In this paper I discuss two efficient alternatives to vector clock, namely, sets of dependency sequences, and hierarchical clock. Both the alternatives are scalable and are immune to fluctuations in the number of nodes in the system. After discussing these mechanisms, I will discuss the new protocol known as Mobi_Causal: A Protocol for Causal Message Ordering in Mobile Computing Systems. The implementation of causal ordering proposed in this protocol uses the new timestamping mechanisms: dependency sequences and hierarchical clocks. This protocol is characterized by the elimination of unnecessary inhibition delay in delivering messages while maintaining low message overhead. This protocol requires minimal resources on mobile hosts and wireless links. The protocol is also scalable and can easily handle dynamic change in the number of participating mobile hosts in the system.

1 Introduction

Several mobile computing applications will be distributed in nature, consisting of multiple processes executing concurrently on different nodes. In order to track concurrency, and the resultant nondeterminism, in the application, it is important to determine the causal and temporal relationship between events that occur in the distributed computation. Concurrency and nondeterminism tracking are important for analyzing, monitoring, debugging, and visualizing the behavior of a distributed system.

Applications running on distributed systems have employed vector clocks, to track causal dependency relations between events on different processes. The vector clock is an integer vector with as many components as the number of processes in the application. It has been proved that a vector of this length is necessary to track causal dependency relationships. For large distributed applications with many constituent processes the size of the vector clock is large. A message has to carry a vector timestamp: the value of the sending process’s vector clock at the time the message is sent. The communication overheads imposed by vector timestamps are acceptable for high data rate fixed wireline networks. However, in a mobile computing system, sending vector timestamps with each message is not a feasible solution due to the low data rate of wireless channels. This is especially so for large distributed applications with several constituent processes.

The communication overheads of vector clocks can be moved away from the wireless channels, and onto the wired network if the MSSs collectively take the responsibility for maintaining causal dependency information on behalf of all the MHs. In such a scenario, the MSSs are said to act as proxies for the MHs. Still, the size of each vector is likely to become prohibitively large to send it even over the higher bandwidth wireline network. Besides, the mobile host population fluctuates from time to time. Mobile hosts join the network, stay connected for a period of time, and then disconnect, either temporarily or permanently. Such fluctuations cannot be handled by vector clock implementations that employ static arrays of integers. This is because such implementations require that the number of nodes in the system does not change.
To overcome the above problems, there is a need for alternative means of tracking causal dependencies that impose as low an overhead as possible and are immune to fluctuations in the number of nodes in the system. In this paper I discuss two dependency tracking schemes that meet these requirements. The fact that all communication involving MHs goes through the MSSs is exploited.

Some contributions are proposed to implement causal message ordering protocols to mobile computing systems. To reduce computation and communication loads on mobile hosts, most of these algorithms store data structures relevant to causal ordering in mobile support stations (MSSs), and the algorithm is executed by the MSSs on behalf of the MHs. This yields to different vision between MSSs and MHs on order of events, which create an inhibition delay in the delivery of messages. This inhibition is due to the fact that an MSS is unable to maintain mutual concurrency information about events occurring at different MHs in its cell when no causal dependency exists between them. In this paper I discuss a new protocol (Mobi_causal) in which inhibition delay is eliminated. This protocol is based upon timestamping mechanisms known as dependency sequences and hierarchical clock as discussed above.

2 System Model

The events at the mobile host can be classified into three categories: message send events, message receive events, and internal events. The send and receive events signify the flow of information between processes and establish causal dependency from the sender process to the receiver process.

There is no global physical clock in the system. Hence, the order in which two events occur at two different processes cannot be always determined solely on the basis of the time of occurrence. However, information about the partial order of occurrence of events belonging to the same application can be gathered based on the causal dependencies between them. Such dependencies can be expressed using the Lamport’s happened before relation (→) between events. The Lamport’s happened before relationship can be defined as:

- \( a \rightarrow b \), if \( a \) and \( b \) are events in the same process and \( a \) occurred before \( b \).
- \( a \rightarrow b \), if \( a \) is the event of sending a message \( M \) in a process and \( b \) is the event of delivery of the same message to another process.
- If \( a \rightarrow b \), and \( b \rightarrow c \), then \( a \rightarrow c \) (i.e., "→" relation is transitive).

Let Past(\( a \)) denote all events in the causal past of event \( a \). That is,

\[
\text{Past}(a) = \{ b \mid b \rightarrow a \}.
\]

Past(\( a \)), along with \( a \), constitutes an envelope of causal predecessor events across all processes in the distributed computation corresponding to the application program. This envelope is referred to as the past cone of \( a \). If \( a \) not \( \rightarrow b \) and \( b \) not \( \rightarrow a \), then \( a \) and \( b \) are said to be concurrent and are represented as \( a \parallel b \).

In such a situation, \( a \) is not present in the past cone of \( b \), and vice-versa.

Lamport’s happened before relation is employed for representing the partial order between events. Let \( C(a) \) and \( C(b) \) be the scalar clock values associated with two events \( a \) and \( b \). Then, \( a \rightarrow b \ not \Rightarrow C(a) < C(b) \). But, the vice-versa is not true.

In case vector clock if \( V(a) \) and \( V(b) \) are the vector timestamps (values of the vector clocks when events occur) associated with events \( a \) and \( b \), respectively, then \( a \rightarrow b \Rightarrow V(a) < V(b) \). If \( V(a) < V(b) \), then event \( a \) is in the past cone of event \( b \).

3 Why n-component vector clocks is insufficient?

Causal dependencies between nodes are created solely by messages. So, an MSS is aware of all intra-cell causal dependencies between MHs in its cell, as well as intercell causal dependencies involving its MHs. This leads to an obvious question: Can all causal dependencies between nodes in the system (MHs and MSSs) be tracked using vector clocks of size \( n \), where \( n \) is the number of MSSs in the system?

**Lemma 1:** Vector clocks with as many components as the number of MSSs in the system cannot accurately capture all the causal dependencies in a mobile computing system.
Proof: Here the proof is given by counter example as shown in Figure 1.

Consider a two cell system, served by mobile service stations MSSp and MSSq. The mobile hosts MHa and MHb are present in the cell served by MSSp. Mobile hosts MHc and MHd are served by MSSq. As there are two MSSs in the system, the vector clock has two components. Initially, the vector clocks at both the MSSs are set to [0, 0]. MHb has to send a message m1 to MHc. So, MHb sends the message to MSSp. MSSp associates a vector timestamp of [1, 0] with the event Send(m1) (as it is the first message sent or received by MSSp), and forwards the message to MSSq. When m1 arrives at MSSq, MSSq sets its vector clock to [1, 1], associates this timestamp with the event Recv(m1) (i.e., V(Recv(m1)) = [1, 1]), and forwards m1 to MHc.

MHd has to send message m2 to an arbitrary node in the network, say MHc. So, MHd sends m2 to MSSq. MSSq receives m2 after receiving m1. Hence, MSSq updates its vector clock to [1, 2] and associates this timestamp with the event Send(m2), i.e., V(Send(m2)) = [1, 2].

As there is no causal relationship between Recv(m1) and Send(m2), the two events are concurrent. However, V(Recv(m1)) < V(Send(m2)) implying that Send(m2) is causally dependent on Recv(m1): which break-down of the isomorphism property of vector clocks.

As if one integer is used per cell to represent causality, the MSS of a cell increments its component of the vector clock for every send and receive event it observes even if these events are mutually concurrent. So, at the cell level, there is an implicit serialization of events. Even if no causal dependency exists between two events occurring at two different MHs in the same cell, the corresponding MSS is unable to maintain mutual concurrency information about those events.

3.1 A Geometric Interpretation

The above proof by contradiction can be represented geometrically as shown in Figure 2. Events e1 and e2 occur at mobile hosts MHc and MHd (present in the same cell), respectively. If the vector clock has only one component per cell, the MSS of the cell can erroneously interpret the past cone of an event, occurring at one MH, to be a union of the past cones of all events that have occurred at all the MHs present in the cell and have been observed by the MSS thus far. For example, event e1, in Figure 2, corresponds to Recv(m1) in the example described above, and e2 corresponds to Send(m2). As e2 is observed by MSSq after e1, MSSq erroneously interprets e1 to be in the past cone of e2. There is no way to determine if two or more events occurred concurrently in a cell.

4 Alternative dependency representations approaches

There are basically two different approaches for representing the causal predecessor of an event e are as follows:

- Sufficient dependency information is sent with each message so that each MH’s set of causal predecessors can be constructed quickly, based entirely on information available at the MSS of the cell in which it is present.
- A small amount of dependency information is sent with each message. When the set of causal predecessors of an event is to be constructed, the dependency information available locally is used as a set of pointers to information that is used for the construction of the causal predecessor set.
The above described both of the approaches have their tradeoffs. The first (known as Dependency Sequences) approach imposes higher communication overheads. However, time to construct the set of causal predecessors of an event is small. The second (known as Hierarchical Clocks) approach has low communication overheads. However, it takes longer to construct the set of causal predecessors of an event. The choice between the two approaches should be influenced by the frequency with which sets of causal predecessors need to be constructed and available communication bandwidth.

5 Dependency Sequence Approach

In this approach the Mobile service station (MSS) of a cell acts as the proxy for all MHs in its cell with regard to maintaining dependency information. All dependency causing events (message sends and receives) in a cell, observed by the corresponding MSS, are assigned sequence numbers in a monotonically increasing fashion. Initially, the sequence counter is set to 0. If \( e_i \) and \( e_j \) are two successive dependency causing events observed by an MSS, then the sequence number assigned to event \( e_j \) is one greater than the sequence number assigned to event \( e_i \).

The causal predecessors of an event \( e \) are represented as a set of dependency sequences. Each dependency sequence in the set corresponds to a cell in the system, and consists of a sequence of non-negative integers. Pairs of these integers represent contiguous sequences of dependency causing events in a cell that are causal predecessors of \( e \). As internal events of mobile hosts do not create causal dependencies, they are not considered. The cardinality of the set is equal to the number of cells in the system. Initially, each sequence in the set is empty. There is no upper bound on the length of a sequence. However, the length of each sequence is always even, and the integers in the sequence are arranged in non-descending order.

The first integer in the dependency sequence of \( e \), with respect to a cell, denotes the first dependency causing event observed by that cell’s MSS that is in the past cone of \( e \). All the dependency causing events of the cell whose sequence numbers are greater than or equal to the first integer and less than or equal to the second integer in the sequence are in the past cone of \( e \). All dependency causing events in the cell whose sequence numbers are greater than the second integer and less than the third integer are not in the past cone of \( e \) and constitute a dependency gap. Similarly, all dependency causing events with sequence numbers between the third and fourth integers in the sequence are in the past cone of \( e \), and so on.

This approach graphically represented in Figure 3. Message send and receive events 0 – 4, 6 – 12, 14 – 17 and 19 – 20 in cell \( C_1 \), represented by rectangular blocks, are causal predecessors of event \( e \) occurring at the mobile host shown in the figure. Events 5, 13 and 18 in cell \( C_1 \) are not causal predecessors of \( e \), and correspond to dependency gaps in the sequence. Similarly, the rectangular blocks on time-lines for cells \( C_2, C_3 \) and \( C_4 \) also represent the causal predecessors of \( e \). Thus, the dependency sequences of \( e \), for the four-cell system shown in Figure 3, are \{\{0, 4, 6, 12, 14, 17, 19, 20\}, \{0, 6, 9, 11, 14, 16\}, \{0, 3, 7, 15, 21, 33\}, \{0, 3, 5, 7, 9, 11, 17\}\}.

It is to be noted that the number of sequences in the set is equal to the number of cells in the system regardless of the number of \( MHs \) in the system. Thus, \( MHs \) may join and leave the system without having an impact on the number of sequences. Hence, the dependency sequence approach is immune to fluctuations in number of \( MHs \) in the system.
6 Hierarchical Clock Approach

6.1 Process Abstraction

The two cell system with four $MH_s$ and two $MSS_s$ shown in Figure 4(a) can be abstracted by a two process system shown in Figure 4(b). The horizontal dotted lines in Figure 4(b) represent the two abstracted processes. The solid directed lines indicate causal dependencies. The lines labeled $i$ indicate dependencies created by *intra-process* events, while the lines labeled $m$ indicate dependencies created by *inter-process* message communication. There is no dependency between the event corresponding to the send of a message from $MH_b$ to $MH_d$ ($e_{p,3}$) and the event corresponding to the reception of a message by $MH_a$ from $MH_c$ ($e_{q,4}$). Hence, in Figure 4(b), even though they occur at the same abstracted process, there is no causal dependency between them. Similarly, there is no causal dependency between events $e_{q,1}$ and $e_{q,2}$, or between $e_{q,2}$ and $e_{q,3}$. The above abstraction of a cell’s behavior as a process with partial ordering between its events, as described above, is a deviation from the standard definition of a process where all its events are totally ordered. Thus, Lamport’s *happened before* relation ($\rightarrow$) cannot completely capture the causal dependency relation. Instead, a new relation $\rightsquigarrow$, is defined over a set of events $e_1$, $e_2$ and $e_3$ as follows:

1. $e_1 \rightsquigarrow e_2$ if $e_1 \leftarrow e_2$, i.e., events $e_1$ and $e_2$ occur on the same abstracted process (cell) and there is a causal dependency from $e_1$ to $e_2$.
2. $e_1 \rightsquigarrow e_2$ if $e_1 \overset{m}{\rightarrow} e_2$, i.e., $e_1$ is the send event of a message in one abstracted process (cell) and $e_2$ is the corresponding receive event in another abstracted process.
3. $e_1 \rightsquigarrow e_2$ if $\exists e_3$: $e_1 \rightsquigarrow e_3$ and $e_3 \rightsquigarrow e_2$.

6.2 Hierarchical Clock Description

Each event of the abstracted process is assigned a hierarchical clock value to capture the $\rightsquigarrow$ relation between events. The hierarchical clock $\phi$ is maintained at $MSS_s$ and has following two components:

1. $\phi^i$ is a local clock representing the $\leftrightarrow$ relation. It is a variable length bit-vector with one bit for every event that has occurred in the process thus far. Each event (message send or message receive) observed
by the abstracted process is assigned a unique sequence number in a monotonically increasing fashion. Let \( e_{i,x} \) and \( e_{i,y} \) be the \( x^{th} \) and \( y^{th} \) events of the abstracted process \( P_i \), respectively. \( \phi^i(e_{i,x}) \) is \( x \) bits long and its \( x^{th} \) bit is set to 1. Also, \( \phi^i(e_{i,x}) \) is generated by bit-wise ORing the \( \phi^i(e_{i,y}) \) vectors for all events \( e_{i,y} \) such that \( e_{i,y} \prec e_{i,x} \). Thus, for every local event of process \( P_i \) that causally precedes \( e_{i,x} \), the corresponding bit is set to 1 in \( \phi^i(e_{i,x}) \).

2. \( \phi^m \) is a global clock representing the \( m \Rightarrow \) relation. It is an integer-vector of \( n \) components, one for every abstracted process (cell) in the system. \( \phi^m(e_{i,x})[k] \), the \( k^{th} \) component of event \( e_{i,x} \)’s global clock, identifies the last event on process \( P_k \) that causally precedes \( e_{i,x} \). For a message send event \( e_{i,x} \), the MSS of the sending cell takes the following action to update the hierarchical clock of the corresponding abstracted process: \( \phi^m(e_{i,x})[k] \), where \( k \neq i \), is set to the maximum of the \( k^{th} \) components of all \( e_{i,y} \) such that \( e_{i,y} \prec e_{i,x} \). \( \phi^m(e_{i,x})[i] \) is set to \( x \). \( \phi^m(e_{i,x}) \) is sent with the message as its vector timestamp. For a message receive event \( e_{i,x} \), the MSS of the receiving cell takes the following action: \( \phi^m(e_{i,x})[k] \) if \( k \neq i \) is set to the maximum of the \( k^{th} \) components of all \( \phi^m(e_{i,y}) \) such that \( e_{i,y} \prec e_{i,x} \) and the \( k^{th} \) component of the vector timestamp carried by the message. \( \phi^m(e_{i,x})[i] \) is set to \( x \).

The example of hierarchical clocks procedure is shown in Figure 5. The figure also show the \( \phi^i \) (local clock) and \( \phi^m \) global clock components of hierarchical clock for each event.

![Figure 5: Local and global clock components for a distributed computation](image)

7 Important Points regarding the Proposal of Dependency Sequences and Hierarchical Clocles

Main Contribution—This paper proposed two schemes (dependency sequences and hierarchical clocks) to overcome the drawback of vector clock (lack of scalability; inability to cope with the fluctuations in the number of mobile hosts; and large memory overhead).

Comparison of Dependency Sequences and Hierarchical Clocks—

1. In the dependency sequence approach, the amount of dependency information that is sent with each message on the wireline network could be large, especially if there are a large number of dependency gaps for message send events. In the hierarchical clock approach, on the other hand, each message needs to carry only a vector of \( n \) (number of cells in the system) integers.

2. In the hierarchical clock dependency chains have to be traversed to determine the causal predecessors of an event. On the other hand the dependency sequence approach no traversal of dependency chains is required, because, the set of dependency sequences always indicates the systemwide set of causal predecessors of an event.

3. Thus there is a tradeoff between the two approaches: the dependency sequence approach has high run-time communication overheads, however, it incurs no extra effort or time delay to determine the causal predecessors of an event. The hierarchical clock approach has low run-time communication overheads, but the determination of causal predecessors of an event incurs delay and communication overhead.
Interesting Ideas about these proposals— The maintaining of the past cone of events is the very well idea, because by this MSS can easily find the predecessors of any event. The periodic global checkpointing is also good idea because it will take care of length of dependency sequences by resetting the dependency sequences to empty if it cross the particular threshold.

Advantages— The proposed approaches having the following advantages over vector clock approach

- Less message overhead (in dependency sequences $O(n \times \epsilon)$ where $n$ is the number of cells and $\epsilon$ is the length of longest dependency sequence) than vector clock ($O(N)$) where $N$ is the number of MH in the system. And here $N \gg n$.
- The proposed approaches are adaptable to change in the number of MHs in the system.


8.1 Problem Definition

The adaptation of causal ordering protocol to a distributed system with mobile hosts raises two major problems: the mobility impact (i.e. the mobile hosts migration between different cells of system), and the delivery message inhibition (i.e. unnecessary waiting delay for the delivery of a message).

This protocol is based on new timestamping mechanisms proposed for mobile computing environment: dependency sequences and Hierarchical clocks as described earlier. The goal of this paper is to study the ability to use these mechanisms for implementing a protocol which resolves the causal ordering problem in mobile computing systems.

Definition (Causal Ordering)—If two messages M and M’ have the same destination and Send(M) → Send(M’), then the causal ordering insures that Delivery(M) → Delivery(M).

8.2 Principle

Each mobile service station $S_i$ maintains, for each mobile host $h_i$ located in its cell, the following data structures: an integer counter, $id_{messE}$, that is incremented each times a message is sent by or delivered to $h_i$. An identifier of the last message received by $h_i$, $id_{messD}$, that is equal to the value of $id_{messE}$ when this message is delivered to $h_i$. Information about the last messages sent by $h_i$ to others mobile hosts $h_j$ is stored in a set, LastSend$_{h_i}$. The one component of the set is denoted by $(h_j, id, \phi^i)$ where $id$ is the identifier of the last message sent by $h_i$ to $h_j$ and $\phi^i$ denotes the local clock value. The value of $\phi^i$ is represented as hierarchical clocks [1]. The number of messages sent between two given receptions is equal to the maximal size of $\phi^i$.

Construction of $\phi^i$

There are basically there kind of messages: messages sent by a mobile host and consequently by its base station, denoted E, messages received by a base station but not necessarily delivered to mobile host, denoted R, and messages delivered from a base station to a mobile host after verifying delivery condition, denoted D.

There are two basic types of events: First, reception events of delivered messages by a mobile host are independent, because a message can be delivered only if the delivery condition is satisfied at the base station level. Second, for a sending message E, it depends on the last received message D by a mobile host and on the messages E that are sent before E for the same mobile host but that are sent after the last received

Figure 6: Types of messages at the MH level
message D. This is illustrated in Figure 6, if the mobile host decides to send the message E in bold, then the causal dependency information of this message is built from the messages E and D represented in the figure by dotted arrows.

This dependency information can be tracked by defining a local clock \( \phi^i \). The clock \( \phi^i \) has three components: \( B_D C_{\text{Int}} B_E \). Where, \( B_D \) is a bit that is equal to 1 if the mobile host has already received a message (i.e. \( id_{\text{mess}} = 0 \)), otherwise this bit is omitted. \( B_E \) is a bit that is set to 1 and represents the current message to be sent. \( C_{\text{Int}} \) is a range of bits with a size equals to number of messages sent after the last received message by the mobile host and before the message to send. One bit of \( C_{\text{Int}} \) is equal to 1 if the corresponding message is already sent to the same mobile host, and 0 else. Figure 7 shows the value of \( \phi^i \) for each message emission by a mobile host \( h_i \).

\[
\phi^i \leftarrow \begin{cases} 1 & \text{if } m_1, \text{First message sent by } h_i, \\ 101 & \text{if } m_3, \text{sent to } h_2, m_3 \text{ sent to } h_3. \\ 0 & \text{else} \end{cases}
\]

**Data structures maintained at mobile host**–

- \( \text{LastSend}_{h_i} = \{(h_j, id, \phi^i), \ldots \} \) in order to keep information about the last message sent to any other mobile host \( h_j \). Therefore, build \( C_{\text{Int}} \) corresponds to look for an entry in \( \text{LastSend}_{h_i} \) where \( id \) is equal to the identifier of receiver host, after that complete the remainder of range by 0.

- \( \text{LastRcv}_{h_i} = \{(h_j, S_j, SD_j), \ldots \} \) that stores identifiers of messages received by \( h_i \) from \( h_j \). Where \( SD_j = \{id_1, id_2, id_3, \ldots \} \) and it is corresponds to a dependency sequences [1].

- \( \phi \), a vector of length \( NS \) (where \( NS \) is the number of base stations). One entry \( \phi_m[k] = \{(h_j, id), \ldots \} \) in the vector corresponds to a set of tuples of the form \( (h_j, id) \) and represents the predecessor messages identifiers of \( m \) in the base station \( S_k \) for which the delivery condition is not yet confirmed. We keep the identifier of message and for which mobile host this message is sent.

**Data structures maintained at base station**–

- \( \text{IdSend}_{S_i} \), an integer counter that is incremented each time a message is sent by a base station \( S_i \).

- \( \text{AtFile}_{S_i} \), a queue to store information about messages not yet delivered in this base station because the delivery condition is not satisfied.

### 8.3 Static Module executed by the MSS

#### 8.3.1 Emission Phase

Emission of a message \( m \) from a mobile host \( h_i \) of base station \( S_i \) to a mobile host \( h_j \) of base station \( S_j \) is done in two steps:

- First, the message is transmitted from the sending mobile host \( h_i \) to its base station \( S_i \) on wireless line.

- Next, the message is sent from \( S_i \) to the receiving base station \( S_j \) on the wired line with the control information piggybacked to the message.

The message to be sent should have the following form: \( (m, \text{IdSend}_{S_i}, \text{depend}_{h_i}, \text{IdLastSend}, \phi) \). The reception of message \( m \) by \( S_i \) from \( h_i \) causes the following operations to take place at \( S_i \):
• Incrementation of its counter \( IdSend_h \), that will be attached to message \( m \).

• The value of boolean variable \( depend_h \) depends on \( \phi' \). Hence, after building the corresponding \( \phi' \) of \( h \), we verify if \( \phi' \) has the form \( 0^*1 \), which means that this message is not constrained by any other message, whether is an internal message to mobile host \( (C_{int} = 0^*) \) or external \( (B_D \) is omitted). In this case, the value of \( depend_h \) is set to \( false \), and consequently the delivery of this message must be done as soon as the message is received by the base station \( S_j \). If not, \( depend_h \) will be set to \( true \), which means that this message depends on whether an internal message of mobile host \( h \) (i.e. depends on message sent by \( h \) before it), or it depends on an external message \( (B_D = 1, i.e. that \( h \) has received a message before the emission of \( m \)).

• The identifier of the last message already sent by \( h \) to \( h \) if it exists, also transmitted in the control information of message. Therefore, if there is \( (h_j, id, \neg) \in LastSend_h \), we assign the value \( id \) to \( id_{LastSend} \), otherwise \( id_{LastSend} \) will be assigned the value 0, which indicates that this is the first message sent from \( h \) to \( h \).

• After the emission of message \( m \), the base station update the global clock \( \phi \), then \( \phi[i] = \phi[i] \cup (h_j, id_{Send_h}, \phi') \), and we add the component \( (h_j, id_{Send_h}, \phi') \) to set \( LastSend_h \). Later on, we eliminate the redundant entries in set \( LastSend_h \), that means if they exist two entries \( (h_j, id_1, \phi_1') \) and \( (h_j, id_2, \phi_2') \) such as \( id_1 < id_2 \) then we remove the entry \( (h_j, id_1, \phi_1') \).

8.3.2 Reception Phase

The message \( m \) arrived to a receiving base station \( S_j \) is delivered to mobile host \( h \) only if the delivery condition is satisfied. This means that all messages causally preceding \( m \) are received by \( S_j \) and delivered to \( h \). The procedure which is followed to deliver the message to the mobile host are as follow:

1. If \( depend_m = false \), then this message is independent from any other message and it is delivered immediately.

2. If \( depend_m = true \), then there are two possibilities: whether \( m \) depends on other internal message to sender mobile host and which is sent to the same host \( h \). In this case, \( id_{LastSend_m} \neq 0 \), then it suffices to verify that this message has been received by \( h \). The reception of a message by a mobile host is expressed by an entry \( (h_i, S_i, Id) \) in the set \( LastRcv_{h_i} \) where \( Id = id_{LastSend_m} \). If this is the case, the message is delivered.

3. If \( depend_m = true \), and \( id_{LastSend_m} = 0 \), then there is no causal dependency is created by an internal message to \( h \) but by the reception of a message sent by another mobile host to \( h \). In this case, the delivery of message \( m \) depends on the delivery of its \( immediate \) predecessors on different base stations. These predecessors are identified by the global clock \( \phi \). An entry in is a set of pairs \( (idf, h_i) \) that represent the identifiers of predecessor messages on the corresponding base station to this entry and for which mobile hosts they are sent. For each entry in \( \phi \) where it exists a predecessor sent to \( h_j \), we search if it exists triplet \( (.., S_k, SD_k) \in LastRcv_{h_j} \) where \( idf \in SD_k \). If this condition is verified then we are sure that all immediate predecessors of \( m \) are received and consequently \( m \) will be delivered to \( h_j \), and so for all queued messages in \( AtFile_{S_j} \) that are waiting for \( m \). The delivery of these messages involves their deletion from \( AtFile_{S_j} \). Because there is no need to queue them now.

4. If none of the above condition \( (1, 2, 3) \) is satisfied, we say that the delivery condition is not verified and consequently the message \( m \) is queued in \( AtFile_{S_j} \). Because the predecessor messages of this message are not delivered to mobile host.

8.3.3 Handoff Module

Let a mobile host \( h \), resident in the cell covered by the base station \( S_i \) moves to cell covered by the base station \( S_j \). The first step established by \( h \) is to submit the message register \( (h_i, S_i) \) to \( S_j \) in order to inform \( S_j \) of its presence and of the identity of previous base station. When receiving this message, \( S_j \) will send handoff begin message to \( S_i \). After that \( S_i \) will transmit \( h \) datastructures to \( S_j \). Next \( S_i \), broadcast new location of \( h \) to all base stations except \( S_i \) and \( S_j \) in order to inform them about the new location of \( h \). After this if any message destined to \( h \) will receive by \( S_j \) then it will redirect it to \( S_j \) without processing it. After receiving the broadcast message about new location of \( h \) the other base stations update their knowledge about the location of \( h \).
8.4 Important Points Regarding this Paper

**Main Contribution**– The proposed protocol (Mobi.Causal) eliminate the unnecessary inhibition delay in delivering messages while maintaining the low message overhead. And this also resolved the causal ordering problem in mobile computing systems.

**Interesting Ideas about this protocol**– The two most interesting ideas about this protocol are: First, The maintainance of causal dependency among messages; Second, The use various delivery conditions at the base station and maintainance of queue for for ensuring causal ordering among messages.

**Advantages**–

- The safety property (the causal ordering is never violated) and liveness property (Each message is delivered after a finite time) are satisfied.
- Easily scalable and adaptable to dynamic change in the number of participating mobile hosts in the system.
- Less communication overhead and better performance (achieved by minimizing the use of wireless bandwidth).

**Drawbacks**– The drawback of this protocol is the non bounded evolution of LastRcv_h, which is result from the characteristics of dependency sequences mechanism.

**Future works**– To compensate the above limitation of this protocol, we can add a procedure to eliminate message identifiers from LastRcv_h, of which the knowledge of the knowledge of their delivery is not necessary. The other ideas for future works is that adaptation of this protocol to Multicast and Broadcast problems.

9 Conclusion

In this paper I have discussed two approaches namely Dependency sequences and Hierarchical clocks, both of these approaches eliminates the shortcomings of vector clock approach. Both approaches are having treadoff between them, therefore their use depends on the application. Following the approaches in this paper I have discussed the one of application of these approaches: called Mobi.Causal protocol for causal message ordering in mobile computing system. This protocol eliminates the unnecessary inhibition delay in delivering messages while maintaining the low message overhead. This protocol is also scalable, and can easily handle the dynamic change in the number of mobile hosts in the system.

References

