N° d'ordre : 04/2009 - D/I.N.

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Superieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene
Faculte d'Electronique et Informatique

THÉSE

Présentée pour l'obtention du diplôme de

DOCTORAT

En : INFORMATIQUE
Spécialité : Programmation et Systèmes

par

Chafika BENZAID

Intitulée

Ordre Causal dans les Environnements Mobiles

Soutenue le 09/07/2009, devant le Jury composé de :

Mme A. MOKHTARI-AISSANI	Professeur	USTHB	Présidente
M. N. BADACHE	Professeur	USTHB	Directeur de thèse
Mme Z. BOUFAIDA	Professeur	U Constantine	Examinatrice
M. A. BELKHIR	Maître de conférences	USTHB	Examinateur
Mme N. NOUALI	Maître de recherche	CERIST	Examinatrice

Abstract

Since, the send and receive of messages are the unique way enabling processes to co-operate and exchange data, then tracking causality between messages in the aim of efficient passing of messages with appropriate semantics is of great importance. To this end, the research community proposed the idea of "*causal message ordering*". The causal ordering concept has a considerable interest in the design of distributed systems and finds its application in several domains, such as management of replicated database updates, determining global snapshot, resource allocation, shared distributed memory, distributed virtual environments, teleconferencing, stock trading, collaborative applications, delivery in multimedia systems, etc.

As far as the mobile computing environment is considered, the design of causal ordering protocols must deal with the new characteristics of this environment, such as mobility, resource constraints on mobile hosts and the limited bandwidth of wireless links.

In distributed computing systems (either conventional or mobile), processes are often organized into groups for supporting various applications, such as computer-supported-cooperative work (CSCW), replicated services, news groups, etc. Group-based communication has proven an important paradigm for developing such distributed systems. In Group Communication Systems (GCS), causal ordering protocols are an essential tool to exchange information. However, achieving ordered delivery of messages for group communication in mobile environment is complicated by the fact that a group composition may change by the join of new members, the leave of existing members and the migration of members between cells.

In this work, we focused on *causal ordering* in mobile environments considering the different kinds of communication (unicast, broadcast and multicast) and dynamic groups.

The emerging trend towards multicast applications in which there is a need to communicate with several other hosts simultaneously (e.g. dissemination applications, collaborative applications, fault tolerant applications, conversational applications, etc.), drove us to make our first contribution. In this proposal, we tried to get benefit from the important characteristics of our unicast protocol (*Mobi_Causal*), such as elimination of unnecessary inhibition delay, low message overhead and scalability, in order to extend it

to allow multicast communication among a static group of MHs and which can fill gaps of existing protocols, especially in terms of control information's size appended to each message. The proposed protocol (*MMobi_Causal*) retains all these interesting characteristics. It provides an optimal communication overhead without causing inhibition effect in the delivery of messages. This optimality is reached using the knowledge of *immediate dependency relationships* between messages and transmitting each message with only the relevant information to compliance with the causal order in the delivery of messages. The protocol is proved to satisfy the safety, the liveness, and the exactly once delivery properties.

Our second contribution is based on our intuition that developing a protocol dedicated to causal broadcast should be more interesting than using a multicast protocol to ensure this broadcast, especially in terms of *control information*'s size appended to each message. The proposed solution (*BMobi_Causal*) proves the advantage of developing a *dedicated* causal broadcast delivery protocol based on *process behavior's awareness*. The originality of our proposal lies on showing how the rather well known behavior of a process can influence the control information's size. We showed that by viewing the behavior of the execution in the sense that the reaction to occurring events is *instantaneous*, tracking the dependency between messages is possible by only keeping information about the message's *immediate predecessor*. Hence, causal broadcast delivery of messages can be easily achieved in a mobile group with a small message overhead. The proposed protocol keeps all the interesting characteristics of our unicast protocol (*Mobi_Causal*). It outperforms its counterparts with respect to communication overhead (its message overhead is $O(1)$). The protocol is also proved to satisfy the safety, the liveness, and the exactly once delivery properties.

The last contribution was devoted to the extension of the two proposed protocols to support dynamic groups while guaranteeing causal consistency. The novelty of our contribution lies in considering the leave and join requests as *data messages*. Then, a control information is appended to these messages and they will be ordered with data messages. The idea of ordering the join/leave requests with regular messages guarantees a causal consistence property that is of a great importance of applications do not needing strong ordering (like total order) requirements for messages. Moreover, it makes the installation of a view fully decentralized and without need to a coordination phase which is a very important advantage, even more in mobile environments. One important property guaranteed by our view management procedure is the *virtual synchrony* semantic. This semantic allows group members, that survive the same group changes, to know for certain that they have delivered exactly the same set of messages. By ensuring that processes deliver the same set of messages in each view, this allows them to maintain consistency across membership changes.

Acknowledgement

I am grateful to the Almighty Allah for giving me the courage and the patience until the achievement of this work.

I had the pleasure and honor to receive support from many wonderful people to whom I would like to express my deepest gratitude.

First of all, I would like to thank Prof. Nadjib BADACHE, my supervisor for his invaluable support, guidance, and encouragement throughout these years. During these years, Prof. BADACHE has not only revealed his scientific qualities but also his human qualities. He is willing to share his knowledge and career experience and give emotional and moral encouragement. I feel very privileged to have had the opportunity to learn from, and work with him. I hope I managed to adopt some of his professional attitude and integrity.

I am very much honored by and grateful to Prof. Aicha AISSANI MOKHTARI, Prof. Zizette BOUFAIDA, Dr. Abdelkader BELKHIR, and Dr. Nadia NOUALI, my committee members, for their kind willingness to judge this work.

My thanks are also due to my family as well as my friends (you know who you are) for their tremendous love and support. These have been the driving force behind my accomplishments. Special thanks to my dear parents, for their love, encouragement and constant support. They are the best parents one could ask for.

Contents

List of Figures

Chapter 1

Introduction

As a fundamental concept, *causality* plays an important role in people's perception and knowledge of their living environment and the world. Actually, the central aim of many studies in the physical, behavioral, social, and biological sciences is elucidation of the *cause-effect relationship* among variables or events [74, 133, 145]. Human beings show a strong propensity to perceive causality between co-occurring events. For example, consider a ball (A) that moves toward a stationary ball (B) until they are adjacent, at which point A stops and B starts moving along the same path. In this example, an observer sees A as causing B's motion, though what actually reaches the observer's eye is nothing more than the movement of the two balls at different times and locations. In other words, the observer attributes causality to objects or events [187].

The causality principle is also recognized as fundamental of distributed computing and forms the basis for event orderings in many distributed systems and distributed service implementations. The notion of causality in the context of distributed systems was first formalized by Lamport [115]. It is based on the relation "*happened-before*" (denoted by $\rightarrow$), among events. The happened-before relation is the smallest transitive relation that satisfies, for any events e, e', $e \rightarrow e'$ if (1) e and e' are events on the same process and e occurred before e' , or (2) e is a send event in one process and e' is the corresponding receive event in another process. If neither $e \rightarrow e'$ nor $e' \rightarrow e$, we will say that e and e' occur *concurrently*.

The asynchronous execution of processes and the finite and non-bounded character of communications confer a *non-deterministic behavior* to distributed systems and make more complex the design, verification and analysis of distributed applications. As the emission and reception of messages are the only means allowing processes to cooperate and exchange data, then the efficient message passing with the appropriate *ordering* semantics is of a major relevance to take care of the *consistency guarantees* of distributed applications.

Several application semantics are precisely captured by *causal consistency*, which consists in respecting the *cause-effect relationships* between the exchanged messages.

In daily life, it seems clear that things (events) happen according to their causal order; the cause (event) must happen before the effect (event). However, this may not be the case in a distributed system due to asynchronous nature of communication channels. This leads to inconsistencies with regard to causality property. In order to overcome this problem, the *causal order of messages* is required.

the idea of "causal order of messages" has been introduced by Joseph and Birman [57] in the context of ISIS project [59]. The causal order of messages guarantees that the delivery order of messages does not violate the causality property in the system. The causal ordering concept is of a considerable interest for the design of distributed systems and find its application in several domains such as management of replicated data, observation of a distributed system, resource allocation, multimedia systems, and teleconferencing. Causal message ordering is typically used to ensure the causal consistency of processes in distributed systems. It has been regarded as an important building block for constructing reliable distributed systems.

Nonetheless, causal ordering has been opposed by some researchers (Cheriton and Skeen are best known within this group [70]), who reach the conclusion that this ordering property, and indeed about almost any type of "property" that a system might support, are undesirable. In their view, causal message ordering is, in almost all cases, more than the application needs (and more costly), or less than the application needs (in which case the application must add some higher-level ordering protocol of its own) [54]. Arguments advanced by Cheriton and Skeen have been criticized by other researchers [52, 75, 177]; Cheriton and Skeen do not address the fact that this approach greatly reduces the complexity, development time, and debugging effort needed when implementing distributed systems. And even if a small performance loss is sometimes incurred, the average user is more concerned with having a simple way to build reliable software than with having total control of the properties of the communication protocols used in an application.

The implementation of the causal order in an asynchronous system requires to attach a *control information* to each message. Thanks to this information, a message will not be delivered until all messages that it depends upon have been delivered. Nevertheless, the main obstacles facing implementations of causal ordering are the amount of this control information. Piggybacking less information in messages is one way to improve the performance and scalability of a causal message ordering protocol.

1.1 Problem Statement

Although distributed computing remains centered on traditional wired infrastructures with client systems on desktops and servers in the data center, we are seeing more and more interest and use of smaller mobile devices. The progression of this trend to enable mobile computing devices to get access and use desired information, resources or ser-

vices is leading to a new environment, which is pervasive or ubiquitous. This has made *mobile computing* possible. A mobile computing system extends a distributed computing system to support mobility of hosts by providing mobile hosts with continuous network connections.

As said above, causal ordering concept has proven to be of a considerable interest for the design of distributed systems. This ordering property is attractive also in the context of mobile systems. In a mobile computing environment, causal ordering is especially important for applications that involve human interactions from several locations. Some of the major applications of distributed mobile systems in which causal ordering is useful are teleconferencing, stock trading, collaborative applications, etc. Nevertheless, this environment comes with new challenges: First, distributed processes in mobile hosts have mobility. Second, mobile hosts are inherently weaker in computing, storage and energy capabilities than fixed hosts. Third, wireless communication has less bandwidth and higher error rate compared to wired communication. Therefore, causal message ordering in this environment requires algorithms that take these factors into account.

In this thesis, we aim at studying the impact of mobile environment characteristics and issues on the design of causal ordering protocols as well as the cost for causal delivery will typically pay off. Note that mobile wireless networks can be classified in two major categories: *cellular networks* (also known as *infrastructured networks*) and *ad hoc networks*. Infrastructured networks, with support base stations (MSSs) and mobile nodes (MHs), represent a major portion of current mobile systems architectures. So, in the sequel, we will be limited to this type of network.

1.2 Thesis Organization

The organization of this thesis is as follows:

Chapter 2 motivates the work by introducing causal ordering. This includes defining the causal ordering concept and explaining why this concept is useful for developing distributed applications. Our own observations and a thorough review of related literature lead us to propose a taxonomy based on different criteria according to which causal ordering protocols can be classified, like the underlying network model, the mechanism used to construct the control information, the consideration of environment assumptions, the communication approach, application domain, support of group communication, support of faults, etc. We also identify a number of key requirements that must be considered when designing causal ordering protocols, especially for cellular mobile environments.

In Chapter 3, we will focus on studying the "*causal communication protocols*" class. The rationale behind this choice is motivated by the fact that this class provides a broad, application-independent implementation of causal ordering, letting us to aim

at studying the influence of mobile environment characteristics and issues, rather than application-specific requirements, on the design of a causal ordering protocol for such an environment. The chapter presents a vast survey of most of the existing causal communication protocols. In order to structure the classification more clearly, we subdivide it into relevant work done in the field of static distributed systems and work relevant in the context of mobile distributed systems. Our presentation of these protocols is mainly focused on the principle used to capture causality between events, the communication and computation costs induced, the type of communication paradigm supported (i.e. unicast, multicast or broadcast), and in some cases, the way that protocols handle failures. We then conclude with a discussion about causal ordering in mobile systems and what is required to make further improvements.

The emerging trend towards multicast applications in which there is a need to communicate with several other hosts simultaneously (e.g. dissemination applications, collaborative applications, fault tolerant applications, conversational applications, etc.), drives us to make our first contribution in Chapter 4. In this proposal, we try to get benefit from the important characteristics of our unicast protocol [47, 43], such as elimination of unnecessary inhibition delay, low message overhead and scalability, in order to extend it to allow multicast communication among a static group of MHs and which can fill gaps of existing protocols, especially in terms of control information's size appended to each message.

The proposed protocol retains all these interesting characteristics. It provides an optimal communication overhead without causing inhibition effect in the delivery of messages. This optimality is reached using the knowledge of *immediate dependency relationships* between messages and transmitting each message with only the relevant information to compliance with the causal order in the delivery of messages. The protocol is proved to satisfy the safety, the liveness, and the exactly once delivery properties.

In Chapter 5, we present our second contribution which is based on our intuition that developing a protocol dedicated to causal broadcast should be more interesting than using a multicast protocol to ensure this broadcast, especially in terms of *control information*'s size appended to each message. The proposed solution proves the advantage of developing a *dedicated* causal broadcast delivery protocol based on *process behavior's awareness*. The originality of our proposal lies on showing how the rather well known behavior of a process can influence the control information's size. We show that by viewing the behavior of the execution in the sense that the reaction to occurring events is *instantaneous*, tracking the dependency between messages is possible by only keeping information about the message's *immediate predecessor*. Hence, causal broadcast delivery of messages can be easily achieved in a mobile group with a small message overhead. The proposed protocol keeps all the interesting characteristics of [47, 43]. It

outperforms its counterparts with respect to communication overhead.

The chapter is organized to give first an overview of our proposal while discussing the motivation behind how we have proceeded to construct the control information. We then present the detail of our causal broadcast protocol and the correctness proof of the protocol. Finally, we analyze the complexity of our protocol and compare it to more closely related works.

Chapter 6 motivates our last contribution (see Chapter 7), which consists on extending the two proposed protocols ($MMobi\_Causal$ (see Chapter 4) and $BMobi\_Causal$ (see Chapter 5)) to support dynamic mobile groups. We give an overview about various ideas and concepts behind group communication, with a focus on group management solutions and causal consistency. We first give preliminaries and important definitions for a good understanding of the subject. We then outline the traditional group communication solutions, designed for wired networks, as well as work undertaken in the context of mobile systems. We finally conclude by discussing how the existing group communication solutions and causal ordering protocols support each other.

Chapter 7 is devoted to the extension of the proposed protocols to support dynamic groups while guaranteeing causal consistency. In this chapter, we provide a detailed description of the algorithm required for the join and leave requests. The novelty of our contribution is that our group view management procedure is based on the idea of considering the leave and join requests as *data messages*. Then, a control information is appended to these messages and they will be ordered with data messages. This approach has the great benefit of rendering the installation of a new view fully decentralized. The membership algorithm presented here is used in conjunction with the message ordering algorithm.

To conclude, Chapter 8 summarizes the main contributions of this thesis and discusses potential future work directions.

Chapter 2

Causal Ordering Concept

2.1 Introduction

The asynchronous execution of processes and the finite and non-bounded character of communications confer a *non-deterministic behavior* to distributed systems and make more complex the design, verification and analysis of distributed applications. As the emission and reception of messages are the only means allowing processes to cooperate and exchange data, then the efficient message passing with the appropriate *ordering* semantics is of a major relevance to take care of the *consistency guarantees* of distributed applications.

Several application semantics are precisely captured by *causal consistency*, e.g. collaborative tools, which consists in respecting the *cause-effect relationships* between the exchanged messages. In daily life, it seems clear that things (events) happen according to their causal order; the cause (event) must happen before the effect (event). However, this may not be the case in a distributed system due to asynchronous nature of communication channels. This leads to inconsistencies with regard to causality property. As an example of the problem of inconsistency that can arise in a distributed system, consider a group of processes $P1$, $P2$ and $P3$. Node $P1$ sends a message m_1 to the group, initializing a data item. On receipt of $P1$'s message, $P2$ sends a further message m_2 to the group, updating the initialized item. Delivery of m_2 to $P3$ before its causal predecessor may result in inconsistent information at $P3$. In order to overcome this problem, the *causal order of messages* is required.

the idea of "causal order of messages" has been introduced by Joseph and Birman [57] in the context of ISIS project [59]. The causal order of messages guarantees that the delivery order of messages does not violate the causality property in the system. Ensuring that messages are delivered in causal order may require that some messages are delayed, so that they can be delivered in the correct order. The causal order concept is of a considerable interest for the design of distributed systems and find its application in several domains such as management of replicated data, observation of a distributed system, resource allocation, multimedia systems, and teleconferencing.

The implementation of the causal order in an asynchronous system requires to attach a control information to each message. Thanks to this information, a message will not be delivered until all messages that it depends upon have been delivered. Causal ordering reduces much of the communication medium asynchrony. The problem has inspired an abundance of literature, with a plethora of proposed algorithms following various approaches and complying with different requirements. Due to the importance of causal ordering in distributed systems and the availability of a significant body of literature on this topic, a classification of existing algorithms becomes necessary and useful at this stage. Based on our study of causal ordering protocols, we have retrieved different criteria according to which causal ordering protocols can be classified, like the underlying network model, the mechanism used to construct the control information, the consideration of environment assumptions, the communication approach, application domain, support of group communication, support of faults, etc. We provide, to the best of our knowledge, the first taxonomy for causal ordering algorithms.

We present in this chapter the definition of causal message ordering concept, as well as a set of examples showing the usefulness of this concept. Then, a taxonomy about causal ordering protocols is given.

2.2 Definition (Causal Ordering)

Introduced by Birman and Joseph in [57], *causal ordering* states that the order in which messages are delivered to the application must be consistent with the Lamport's "*happened-before*" relationship [115] of the corresponding sending events. More formally:

Definition 2.1. *If for any two messages m and m', $Send(m) \rightarrow Send(m')$ and m and m' have the same destination, then the causal ordering ensures that $Delivery(m) \rightarrow Delivery(m')$.*

It's important to distinguish between the arrival of a message and its delivery. The arrival of a message means that the communication network has placed the message in the recipient process's buffer. While, the delivery of a message means that the process has taken the message for its processing. Intuitively, for a message m the relation $Send(m) \rightarrow Delivery(m)$ is always verified.

Causal message ordering guarantees that the order of delivery of messages does not violate causality in systems of communicating processes. Typically, messages are delayed at the receiver until all causally preceding messages are delivered. Figure 2.1 shows a delivery that violates causality, where message m_1 is delivered after m_3, despite the fact that the $Send(m_1)$ event causally precedes the $Send(m_3)$ event.

Figure 2.2 shows how figure 2.1 might look if causal message ordering was enforced.

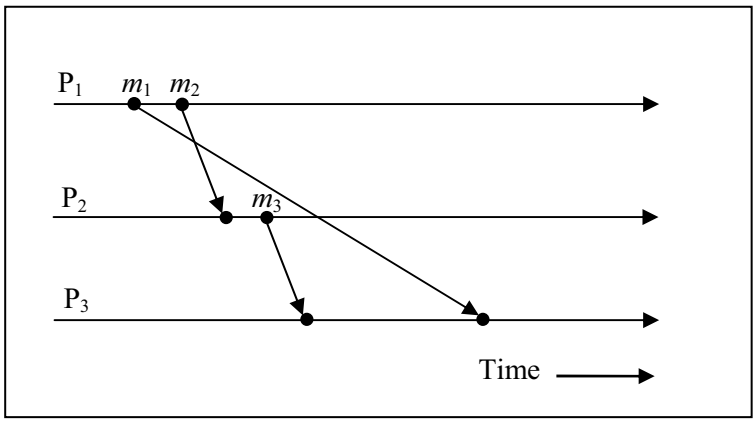

Figure 2.1: A message delivery ordering that violates causality.

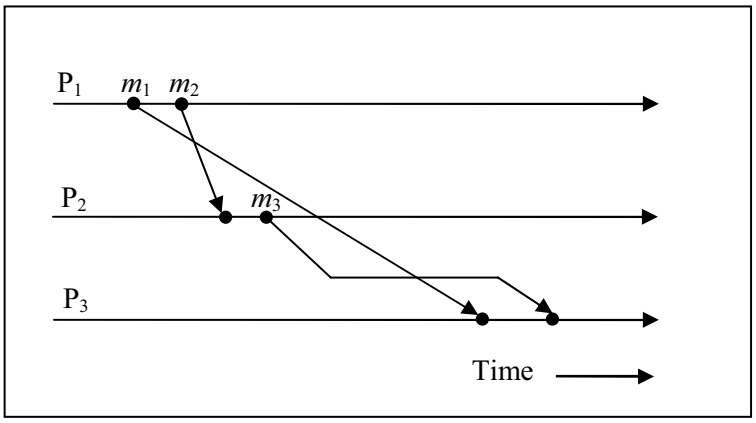

Figure 2.2: Message delivery with causal ordering applied.

2.3 Examples of the Usefulness of Causal Ordering

In fact, causal ordering is fundamental to many problems [176] occurring in distributed computing where causal consistency must be guaranteed. It can be used to maintain the consistency of replicated data located at different sites [58], observe behaviors of a distributed systems [176], and preserve semantic causality in news or teleconference applications [154]. In addition, many implementations and extensions of causally ordered communication have been done in distributed shared memory systems [7], multimedia systems [4], security domains [168], event notification systems [124], distributed virtual environments [187], mobile computing systems [14, 185, 121, 33, 65], etc.

For example, determining a global snapshot of a distributed computation essentially requires finding a collection of local snapshots which is consistent with causality [13]. Visualization of concurrent activities in distributed systems faces a similar problem. Here, causality determines the order in which events must be presented to ensure that every cause precedes its effect. Yet another example is the realization of multicast protocols; as it turns out, a strict atomic multicast semantics is often not required, but

a causal delivery order suffices and leads to more efficient protocols.

The causal order property is useful, among other things, for the following domains:

2.3.1 Management of a replicated data

Consider a data X replicated on various sites [156]. In order to ensure mutual consistency of the various copies x_1, x_2, ..., the updates to these copies must take place in the same order. In particular this introduces the need for mutual exclusion on the updates done by two different sites. This can be solved using a token; when in possession of the token, a site can update his copy of X and broadcasts the update to all other sites having a copy of X (Figure 2.3). Causal ordering ensures that all sites will receive all the updates in the same order (the update W_i will not be delivered before the update W_j, for $i < j$), which ensures mutual consistency of the copies [58]. The token ensures total ordering of the updates on X and causal ordering ensures that all the copies x_i are updated in this same order.

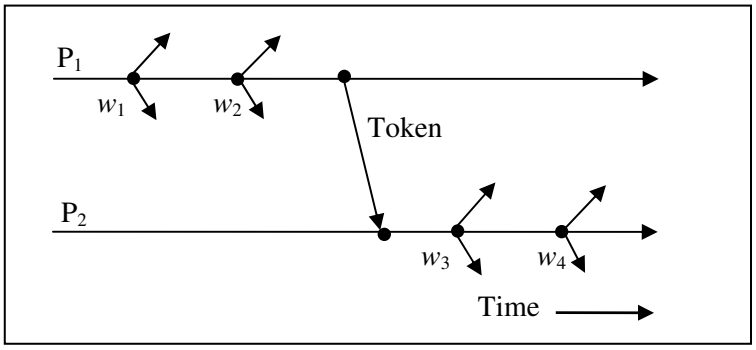

Figure 2.3: Management of a replicated data.

Consider now a second data Y, controlled by a second token, and two sites S_1, S_2 (with copies x_1, y_1, resp. x_2, y_2). Both sites see the same sequence of updates on x_1, x_2 and on y_1, y_2. The interleaving of these sequences can nevertheless be different on S_1 and S_2. In other words, causal ordering does not define a total order.

2.3.2 Resource Allocation

This problem can be solved by a resource allocator residing on a site, which receives allocation requests in a FIFO queue. With this scheme, the requests are honored in the reception order. This might however be considered unsatisfactory in some cases: it might be desirable to honor requests not in their reception order, but in their emission order; i.e. if two requests R_1 and R_2 are such that $R_1 \rightarrow R_2$, then R_1 should be honored before R_2. This is again a causal ordering problem. Note that this example, in the particular case of mutual exclusion, was used by Lamport in his paper introducing logical clocks [115]. Lamport's solution was to duplicate the resource allocator on each

site (each site being concerned with all the requests and having to acknowledge them). This can be avoided using causal ordering.

2.3.3 Teleconference

A teleconferencing application is an application in which more than one distant person can carry on a conversation. We assume that each participant in the conversation can send messages and respond to others at any given time, and that a message and its responses can be sent to a number of participants. We suppose that a user A sends a message and at the reception of this message by a user B, this last sends a response. The causal message delivery property ensures that no participant can receive the response from B before receiving the original message from A. without the causal ordering property, a participant can receive a comment about some ideas before the reception of the message that expresses the original idea. In this case, the idea may be wrongly interpreted.

2.3.4 Distributed Simulation

Ideally, distributed simulations should reproduce exactly the temporal relations among the events that occur in the real world being modeled. However, the heterogeneous delays associated with the computations and message transmissions over the network may lead to violations of such relations. For example, in a distributed simulation consisting of three federates (Figure 2.4): a tank, a target and an observer, the observer may see the target destroyed before it was fired upon by the tank (as shown by the dashed arrow), which obviously contradicts the observer's real-life experience. Infrequent occurrences of such phenomena may not be a serious problem to the game. However, if this kind of phenomenon occurs frequently, it may seriously affect a human participant's perception of the virtual environment.

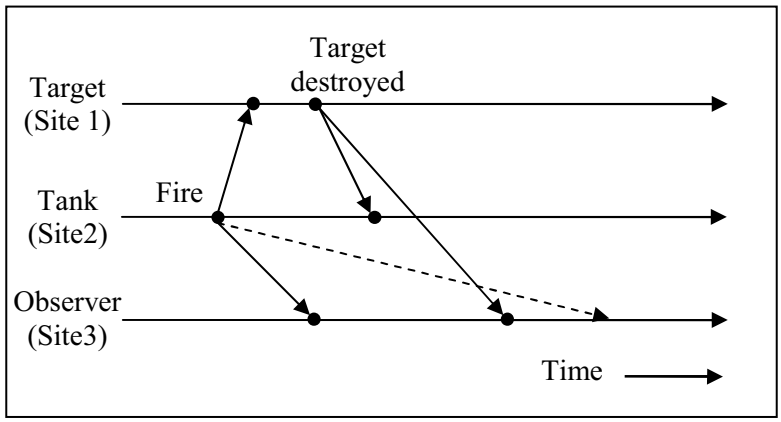

Figure 2.4: A simple distributed simulation scenario.

Causal order violations may also damage a participant's decision-making and the

interactions among the participants in a distributed virtual environment (*DVE*). For example, consider a scenario in a *DVE* for training drivers, as shown in Figure 2.5: The trainee stops the car when the traffic light turns red, and starts the car when the traffic light turns green. Suppose that the "green" message is delayed in the network, causing it to be received after the "start" message at the trainer's site. Thus, the trainer will think that the trainee has violated the traffic rule, and may issue a warning to the trainee or some of the trainee's points may be deduced. This may damage the training effect and my even puzzle the trainee. For example, the trainee may doubt whether it is right to start the car when the traffic light turns green. Note that the scenario in this example is rather simple. For more complex scenarios, such as when the trainee is learning to act with a new weapon system under various tactical situations, frequent occurrences of wrong warnings due to causal order violations may even cause the trainee to form bad habits and wrong knowledge of the weapon system and tactical rules. Therefore, causal order violations need to be reduced as much as possible in a *DVE*.

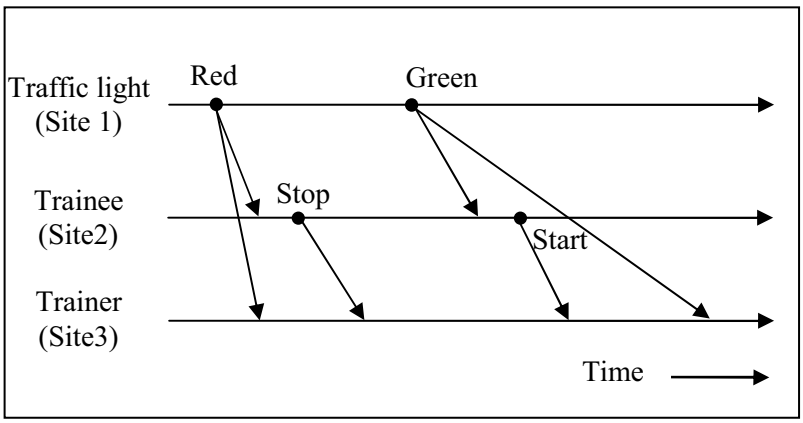

Figure 2.5: A distributed training environment.

2.3.5 CSCW

In a collaborative editing application, asynchronously interacting users may want to access a shared document that is composed of many chapters. Each user corresponds to a process that executes one or more transactions. A query transaction reads chapters of interest to the user. Causal consistency guarantees that the user will always get a set of chapters that include all causally preceding updates. It is possible that concurrent transactions initiated by different users update and define new versions of some chapters. Consistency criterion can be defined to deal with such concurrent updates. It can ensure that a user has a causally consistent view of all chapters and all users get the same 'last' version of each chapter at the end of an editing session.

2.3.6 Event Notification Service Systems

An event notification service paradigm can be used in many applications such as stock exchanges, weather and traffic reports, travel updates, alerting services, emergency notifications, banking, auction, etc. For instance, a stock exchange worries that overhead displays should preserve the causal order in which trades occurred; otherwise, traders might fail to recognize a related series of events. And, there are loose real-time constraints in a stock exchange application. By studying these sorts of considerations, one can ask how valuable it is for the underlying message transport system to provide causal order [52].

Another example showing the need of causal ordering is a distributed security trading system [120] that lets different marketplaces trade with each other. A basic business scenario of such a system can be described as follow: Clients post their offers through brokers, and orders are broadcast to the marketplaces. After the marketplaces find a buy and sell match, the marketplace broadcasts the matching orders to update the databases at each marketplace. Brokers communicate with each other directly to settle the match. If no restrictions are imposed on the order of occurred events, the violation of the following causal orderings can cause a trade to fail: a *Buy notification* must be completed before *Buy settlement details* starts, a *Sell notification* must be completed before *Sell settlement details* starts, and a *Trade notification* must be delivered before a *Transfer shares* is delivered.

2.4 A Taxonomy of Causal Ordering Protocols

In this section, we survey the state-of-the-art causal ordering protocols. Based on our own observations and a thorough review of related literature, we distinguish different criteria according to which causal ordering protocols can be classified, like the underlying network model, the mechanism used to construct the control information, the consideration of environment assumptions, the communication approach, application domain, support of group communication, support of faults, etc.

Overall, the causal ordering protocols can be broadly classified into two categories based on the underlying network model considered, namely: protocols devised for *static* distributed systems and protocols devised for *mobile* distributed systems.

2.4.1 Static distributed system

A *distributed system* is composed of a finite set of sites interconnected through a wired communication network. Each site has a local memory and a stable storage and executes one or more processes. Without lost of generality, we assume there is only one process per site. Local states of all processes are assumed to be disjoints, which means that processes do not share a common memory, and they communicate only by ex-

changing messages through channels of the underlying network. It is the multiplicity of sites connected through a network that makes the *system distributed*. The absence of a fixed bound on process relative speed and on message transfer delays makes the system *asynchronous*. Furthermore, an essential property of distributed systems is the absence of a global clock or local clocks perfectly synchronized.

In a distributed system, the behavior of each process is controlled by a local algorithm, and is completely characterized by a totally ordered sequence of *events* denoting local state changes (*internal* events) or communication activities (*send* and *receive* events). The concurrent and coordinated execution of all local algorithms forms a *distributed computation*.

The communication medium does not deliver messages in the same order to all sites, because messages of different sites can be propagated along different paths. Thus, it is necessary to every host to force a certain consistent order on messages, like causal order, in order that applications can have messages delivered in the same order, without taking care of the order in which messages arrived. Causal ordering protocols, in conventional distributed systems, mainly differ by the size of control information carried by messages.

2.4.2 Mobile distributed system

The term "mobile" implies *able to move while retaining its network connections* [97]. The model of a distributed system including mobile hosts is represented by two kinds of entities: mobile support stations (MSS) and mobile hosts (MH). The MSS nodes are fixed and connected among them via a wired network. Typically, the wired network has a high bandwidth and a low propagation delay. An MSS support the MH nodes in communicating among them and with the other $MSSs$. $MSSs$ and MHs communicate via a wireless network. Each MSS defines a connectivity area (*cell*) where MHs can reliably communicate. At any given time, an MH is assumed to be within the cell of at most one MSS, which is called its *local* MSS. Whenever an MH moves from one cell to another, a $hand-off$ procedure is performed in which the communication responsibilities of MH are transferred to the new local MSS.

As far as the mobile computing environment is considered, the design of causal ordering protocols must deal with the new characteristics of this environment, such as mobility, resource constraints on mobile hosts and the limited bandwidth of wireless links.

Based on whether a central controller is involved in message ordering, causal ordering protocols can be categorized as *centralized*, and *fully distributed* protocols.

2.4.3 Centralized approaches

In centralized approaches (e.g. [56, 104, 157]), a dedicated coordinator process serializes all messages exchanged in the system, effectively imposing a total order on delivery that is consistent with the causal order.

The centralized approach creates a performance bottleneck at the coordinator, resulting in performance degradation especially when message exchanges are frequent. This drawback makes centralized approaches unsuitable for large-scale and mobile systems.

2.4.4 Distributed approaches

In distributed approaches (e.g. [57, 165, 156, 166]), each process can send messages directly to any others without the intervention of a coordinator. Most of the proposed protocols adopt this approach.

Furthermore, these protocols can be classified into *piggybacking-based*, *logical time-based*, or *physical time-based* causal ordering techniques depending on mechanism used to construct the control information.

2.4.5 Piggybacking-based technique

One method for determining the causal relationship between events is to assign a complete *causal history* to each event. A causal history contains all events that causally precede a particular event.

In a piggybacking-based technique (e.g. [57]) also called *causal history*-based technique, each message carries a history copy of all causally prior messages. Thus when a message m is delivered to a process P, copies of all messages addressed to P that causally precede m also arrive with m or have arrived earlier.

The piggybacking approach may either suffer from unbounded growth of the information added to messages, or require a complex mechanism to prevent it.

2.4.6 Logical Time

The drawback of the previous technique is that a huge amount of information is piggybacked on every message. This can be very inefficient. The idea of the technique here is, rather than piggybacking with a message m the set of messages that causally precede m, to piggyback some information about the existence of these messages (e.g. their identifiers).

This technique is based on the concept of logical time [115], in particular vector time [90, 128]. So that, for example, instead of the contents of all causally preceding messages, only vector clock timestamps of causally preceding messages need to be carried in each message (e.g. [165, 156, 166]).

Logical time was first introduced by Lamport [115]. The idea is as follows: Each process, P_i, possesses a logical clock, LC_i. The requirement of clock LC_i is that it should satisfy the causality relation with respect to logical clocks at other processes in the system. More formally, the *clock condition* is:

$$\forall e, e' : e \rightarrow e' \Rightarrow LC(e) < LC(e')$$

Lamport Timestamps Lamport's time maps events to integers using the following rules:

1. Between any two successive events at P_i, LC_i is incremented by a positive value d.
2. A process P_i piggybacks LCi on every message sent.
3. Any process P_i, upon receiving a message with timestamp LC_m, sets $LC_i = max(LC_i, LC_m) + d$.

It is fairly easy to prove that these rules will satisfy the above clock condition [115], but does not characterize causality, as illustrated in Figure 2.6. For example, $LC(e_1^2) < LC(e_2^3)$ although e_1^2 and e_2^3 are causally independent. Therefore, Lamport time is not sufficient to characterize causality and can not be used to prove that events are not causally related.

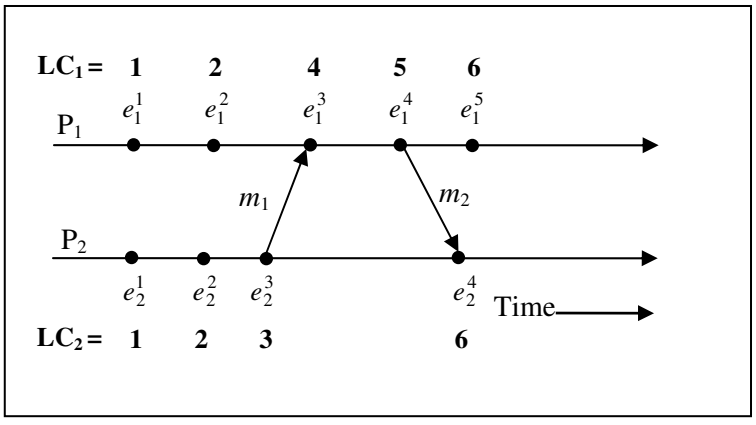

Figure 2.6: Lamport time.

Vector Timestamps Vector clock is the main mechanism to track causality among events in a distributed computation. This clock was defined independently by Fidge [90, 91] and Mattern [128, 169]. Rather than maintaining a single integer, each process P_i maintains a vector of integers, V_i, of size N, where N is the number of processes in the system. This is the vector clock of the process.

The vector time V_i of a process P_i is maintained according to the following rules:

1. $V_i[k] := 0$, for $k = 1, \ldots, N$ processes.
2. On each internal event, process P_i increments V_i as follows: $V_i[i] := V_i[i] + 1$.
3. On sending message m, P_i updates V_i as in (2), and attaches the new vector to m.
4. On receiving a message m with attached vector time V_m, P_i increments V_i as in (2). Next P_i updates its current V_i as follows: $\forall j \neq i, V_i[j] := max(V_i[j], V_m[j])$.

The vector clock has several interesting properties which adequately characterize causality. In particular, we have:

1. $e \rightarrow e' \Leftrightarrow V_e \leq V_{e'}$. The comparison between two vector time stamps is defined as follows:

$$V_e \leq V_{e'} \Leftrightarrow \forall i, V_e[i] \leq V_{e'}[i]$$

   Note that this defines a partial order, not a total order. It is then fairly trivial to show that causal message ordering is violated if and only if there exists a message m with a timestamp V_m which arrives at a destination S where the local time, V_S, is such that $V_S < V_m$.
2. $e \mid\mid e' \Leftrightarrow V_e \not\leq V_{e'}$ and $V_{e'} \not\leq V_e$.

Vector clocks allow processes to determine the precise causal relation between pairs of events in the system. However, fully distributed approaches exploiting vector clocks impose a size of $O(N^2)$ message header on every message (N is the number of participating processes), which has been proven necessary for system-wide causal message ordering [4, 67]. This overhead becomes intolerable when N grows large.

2.4.7 Physical Time

In this case, algorithms (e.g. [4, 39, 40, 2, 114]) rely on synchronized clocks, and use physical timestamps (timestamps coming from the synchronized clocks), instead of logical ones.

A *physical clock* consists of a crystal quartz that oscillates at a given frequency and two registers: (i) a *counter*, and (ii) a *constant* register. The value of the constant register depends on the frequency of the quartz. The two registers are used as follows: the counter is initialized with the value of the constant register. At each oscillation of the quartz, the counter is decremented. When the counter reaches 0, then the clock is incremented, and the counter is reset to the value of the constant register.

Although, it is impossible to have a common physical clock in a distributed system, there exist clock synchronization protocols [76, 82, 135, 181, 122] or technological

synchronization devices (e.g. Global Positioning System, GPS) that allow all the participants across a distributed system to agree on a common clock value whose drift with respect to physical time remains always bounded. The literature distinguishes two types of clock synchronization; namely, *external synchronization*, in which All clocks are synchronized to a real time standard such as Coordinated Universal Time (UTC) and *internal synchronization*, in which clocks are relatively synchronized to each other, but not necessarily with real time.

2.4.8 Hybrid Time

The work in [105] consists in employing an hybrid timestamping mechanism to ensure causal delivery within a static large-scale group composed of *local* subgroups interconnected in a wide area network, like Internet, through gateway processes. A subgroup composed of the gateway processes is a *global* one. Authors combine the use of both logical and physical clocks. Each local subgroup takes usage of physical clock and each gateway process uses vector clock to communicate with the other gateway processes. It is assumed that physical clocks in processes are locally synchronized in each local subgroup but not globally synchronized in a group. While reducing the message overhead, the protocol involves the introduction of unnecessary order between events due to the use of physical clocks.

In addition to the above, causal ordering protocols can be classified into three categories, namely, *unicast*, *multicast* and *broadcast* protocols depending on the supported communication paradigm.

2.4.9 Unicast protocols

In unicast protocols (e.g. [156, 68, 186, 14, 47]), a message is sent by one process and is destined to only one other process. These protocols are also called *point-to-point* protocols.

2.4.10 Multicast protocols

In multicast protocols (e.g. [57, 113, 162, 60, 150, 65, 39, 159]), a message is sent by one process and is destined to an arbitrary number of other processes.

2.4.11 Broadcast protocols

In broadcast protocols (e.g. [94, 64, 37, 33, 80, 2, 159]), a message is sent by one process and all other processes are recipients of the message.

The message overhead of some multicast (broadcast) protocols can be degenerated to $O(n)$. Observe that a multicast can be achieved by multiple unicasts but not vice versa. The reduction of complexity essentially results from abbreviating redundant,

repeatedly identical causality information of messages in the unicast-type protocol.

Focusing on environment assumptions, a classification of causal ordering protocols is possible depending on whether protocols are *environment assumptions-based* or *environment assumptions-free*.

2.4.12 Environment assumptions-based protocols

This category (e.g. [129, 170, 186, 143, 162, 5, 37, 25, 60, 51, 93]) is composed of solutions that assume either a certain network topology, a particular group structure, and/or execution models.

The main drawback of protocols that take into account a particular network topology is that they can only be used in applications where the topology is fairly static, or when it can be calculated at the moment of compilation or configuration. These protocols turn out to be completely inadequate in the cooperative applications, since it is impossible to know in advance when, where, or who will participate.

The other drawback of these protocols is that they usually introduce an additional delay in the causal delivery of messages, due to either the use of re-diffusion servers, or to the chosen execution model. Furthermore, they use a complex organization and synchronization structures.

2.4.13 Environment assumptions-free protocols

In this category, no assumptions are made about the group or topological network structure. Protocols are only concerned with identifying the necessary conditions to ensure a causal delivery of messages, and/or with arranging optimal coding to represent and transmit this information.

In these protocols, all participants interact without an intermediary and they all have the same role. This free protocols from needing to worry about any organization structure maintenance and the induced cost. For this raison, most proposed solutions either in the context of static or mobile environments, do not make any environment assumptions.

From the application domain perspective, the causal ordering protocols can be grouped into several categories, such as causally ordered communication protocols, causally consistent memory, causal message logging protocols, etc.

2.4.14 Causal communication Protocols

This class of protocols focuses its attention on developing communication infrastructures that deliver messages in causal order instead of providing causal consistency with regard to application-specific needs. This approach provides transparency, correctness

and efficiency for processes requiring causal ordering. In addition, it provides a broad, application-independent implementation of causal ordering.

The main objective of these protocols consists in presenting techniques aimed at reducing the amount of information transmitted through the network while ensuring a causal delivery of messages.

By utilizing Lamport's causal relationship, various causal message exchange protocols have been proposed. First, in a static distributed system, a causal multicast has been proposed [57]. This protocol uses a method of attaching a message transmitted previously to the respective messages. This method of attaching some information as a message header to the respective messages has also been utilized in various causal message exchange protocols (e.g. [166, 156, 165]) and the size of the header (i.e. message overhead) becomes an important evaluation scale of the protocols. In recent years, many causal message exchange protocols (e.g. [14, 185, 121, 33, 65]) between mobile hosts have also been proposed for distributed mobile systems.

As our contribution comes under (fits in) this class, we will discuss this category of protocols in more detail in the next chapter.

2.4.15 Causal Ordering in Real-Time Systems

Real-time systems require the notion of physical time because of the constraints imposed by the deadline on the data. Therefore, logical clocks-based causal ordering is not sufficient in real-time multimedia traffic, because it does not take into account that the message can be discarded if it arrives at its destination after the expiration of its deadline.

In order to ensure the respect of real-time delivery constraints, the existence of a causal clock synchronization protocol [134] is assumed. This protocol provides processes with a unique value representing the global physical time whose drift with respect to physical time is bounded and whose granularity and precision are such that all the causally dependent events are produced at different times.

Verissimo in [180] introduced a generic model for causal delivery protocols in real-time systems. The objective was to formalize a set of synchronism and ordering properties for these protocols. Later, Baldoni *et al.* [39] defined the notion of Δ-causal ordering which was firstly introduced in [184].

Definition 2.2 (Δ-Causal Order)**.** *A distributed computation $\hat{E}$ respects Δ-causal order if:*

1. *all messages in $M(\hat{E})$ that arrive within Δ are delivered within Δ, all the others are never delivered (they are lost or discarded);*

2. *all delivery events respect causal order.*

The concept at the basis of Δ-causal ordering is that typically in real-time applications, messages have a *lifetime*, Δ, after which their contents can no longer be used, moreover some of them can be lost. This new abstraction, called Δ-causal order, requires to deliver as much messages as possible within their lifetime in such a way that these deliveries respect causal order. Δ-causal order communication mode is particularly useful for multimedia real-time collaborative applications or groupware real-time applications. Among protocols proposed for implementing Δ-causal ordering are [4] and [39]. The protocol in [4] is based on the definition of a *minimum wait time*, say T_w, such that $\Delta < 3T_w$ and during which a process is prevented from transmitting each time it receives a message. There can be at most a transitive message dependency of length two or less because $\Delta < 3T_w$. If the dependency chain is longer, then the causally prior message will have expired. As such, it is merely necessary to transmit two vectors of length N, the process's vector time and the i^{th} row of the dependency matrix. Thus the cost is $O(N)$, however an extra latency is incurred by the waiting time. In [39], the Δ-causal ordering is ensured by tagging every message with its sending time and a matrix $SENT$ reflecting the sending process's best knowledge about the sending time of the last message sent by every process to any other process. Therefore, the message overhead of this protocol is $O(N^2)$. A variant of this protocol to the case of broadcast communication is proposed in [40] reducing the amount of control information carried by messages to $O(N)$. This reduction is obtained by taking into account transitive dependencies on the message sends. Another variant of this protocol is proposed in [2, 42]. The protocol relies on a hierarchical architecture aiming at ensuring scalability for multimedia applications.

In subsequent work, Rodrigues *et al.* [159] generalized the notion of Δ-causal ordering by proposing the notion of *Deadline-constrained* causal ordering. Deadline-constrained causal order resembles Δ-causal order in the sense that both associate a deadline with a message. However, in Δ-causal a message must wait for its predecessors if these predecessors are timely. In deadline-constrained causal order, each message has its own deadline and, if it arrives on time, never misses its deadline due to preceding messages. It should be noted that if all the messages have the same timeliness constraints, deadline-constrained causal order and Δ-causal order are equivalent. Two implementations was proposed in the context of multicast and broadcast communication with $O(N^2)$ and $O(N)$ message overhead, respectively. An example of distributed soft real-time applications that benefit from the use of a deadline constrained causal ordering is distributed trading systems.

Kulkarni and Arumuga [114] considered the tradeoff between causal delivery and timely delivery of messages by defining the notion of *approximate causal delivery*. This solution is intended for sensor networks that provide simple guarantees about the clock drift among sensors and about maximum delay of messages that are not lost. The

solution lets the sensors to choose the level of causality violations it can tolerate and the time for which it will have to buffer the received messages. Authors presented two algorithms for an approximate delivery. Based on the ability to tolerate causality violations, the first algorithm allows the observer to reduce the delivery time of messages. In the second algorithm, the system checks the message queue before delivery to determine if it might violate the requirements of causal delivery, and hence, reduce the causality violations further.

In [187], the authors addressed the causal ordering of events in distributed virtual environments (DVEs). In order to make a tradeoff between the real-time requirement of DVEs and causal ordering, the authors proposed a scheme aimed at maintaining information about only the immediate predecessor for a given event. Than, the protocol requires that each event carries with it a copy of its immediate causally preceding event.

2.4.16 Causal Memory Ordering

Distributed Shared Memory (DSM) emulates shared-memory systems in distributed asynchronous message passing systems. DSM is traditionally realized through a distributed *memory consistency system*(MCS) on top of a message passing system providing a communication primitive with a certain quality of service in terms of ordering and reliability [28]. The implementation of MCS enforces a given consistency criterion. Some of consistency criteria that have been considered are, from more to less constraining ones: Atomic [117], Sequential [116], Causal [8] and PRAM [123]. The causal consistency model has gained interest because it offers a good tradeoff between simplicity of programming and performance of the consistency model implementation.

A causal consistency is ensured if each processors order respects internal order and Write-Read causality. A number of algorithms implementing the causal memory model have been proposed in the literature (e.g. [8, 155, 88]). Most algorithms implementing causal memory, in order to increase concurrency, support replication of data.

In [8] an algorithm is introduced which implements causal consistency by using vector time stamps. Authors in [38] have studied causal memories in the context of emerging dynamic systems (e.g. peer-to-peer). In order to track causality order relations between operations, they implement a plausible clock system that is an adaptation of the one proposed by Ahamad *et al.* in [174]. Plausible clocks were also used by Ram *et al.* in [98] to implement a causal memory in a mobile environment. Authors in [93] proposed a type of consistency management, called *causal cluster consistency* which is aimed for applications where a large number of processes share a large set of replicated objects. Causal cluster consistency provides a dynamic interest management for processes on replicated objects. Processes can observe updates which correspond to their interest in optimistic causal order. This guarantees that an event will only be delivered if it does not causally precede an already delivered event. This work also describes a

protocol implementing causal cluster consistency, which provides a fault tolerant and dynamic membership algorithm to manage the cluster members.

Fernández *et al.* [88, 89] explored the interconnection of causal memory systems. Such systems may have different implementations, as far as they are propagation-based. They devised algorithms for interconnecting those systems. The interconnection algorithms proposed require the existence of reliable FIFO channels connecting processes from each system. The resulting system is also causal.

2.4.17 Causal Message Logging Protocols

Message logging is a common technique used to build systems that can tolerate process crash failures. These protocols require that each process periodically record its local state and log the messages it received after having recorded that state. When a process crashes, a new process is created in its place; the new process is given the appropriate recorded local state, and then it is sent the logged messages in the order they were originally received. All message logging protocols require that the state of a recovered process be consistent with the states of the other processes. This consistency requirement is usually expressed in terms of *orphan processes*, which are surviving processes whose states are inconsistent with the recovered state of a crashed process.

Message logging techniques are classified into *optimistic* (e.g. [172]), *pessimistic* (e.g. [99]), and *causal* (e.g. [86, 16, 17]). The causal message logging approach offers advantages over the two other message logging techniques. It never creates orphan (like pessimistic protocols) and it does not synchronously log to stable storage (like optimistic protocols). Causal message-logging protocols, however, do require the causal effects of message deliveries to be tracked.

Causal message logging sends message receive ordering information with each message. This information includes receives and their causal history since the last send. In [17, 18], a general causal message-logging protocol is given where causal history information for only f processes is included. This method then tolerates f simultaneous failures. Family-based logging [16] and Manetho [86] are two examples of causal message-logging protocols for the special cases $f = 1$ and $f = n$ respectively.

In [15], authors studied the cost of tracking causality in causal message-logging protocols. They specified six different methods of tracking causality on which all of the published causal message-logging protocols are based. They have shown that the tradeoff between excess piggybacking due to inaccurate causality and the extra piggybacked information to increase the accuracy of causality tracking is both complex and application specific. Lee *et al.* [119] tried to reduce the message overhead compared to the previous protocols by defining a new message log structure based on *inheritance relation* between logs.

Causal message logging suffers from complications associated with recovery. One

particular difficulty occurs when multiple processes fail simultaneously [85]. Solutions have been presented but they require blocking unfailed processes during recovery [16] or some synchronous logging to stable storage while recovery is ongoing [86, 17]. Although Elnozahys protocol [85] solves the problem, it requires a central recovery leader, which may be a performance bottleneck. Authors in [163] presented an approach for recovery in causal message logging which does not require unfailed processes to block or require special coordination among recovering processes. Elnozahys protocol [85], if combined with *independent checkpointing* [50], may force the system to recover to be in an inconsistent state. A solution to this problem is proposed in [9].

Bhatia *et al.* [51] presented an implementation of causal message logging that is designed for use in large scale, wide-area grid infrastructures. The protocol uses a hierarchy of proxies that act as caches for recovery information. Ahn *et al.* [11] are interested in designing a causal message logging protocol with independent checkpointing for handling the constraints of mobile environments. In a later work [10], Ahn proposed two fault-tolerant protocols for mobile computing systems; a causal message logging protocol and a receiver-based pessimistic message logging protocol for tolerating failures of mobile hosts and base stations respectively.

A further distinction may be made based on the requirement of reliability. We distinguish protocols proposed for *failure-free* systems and protocols proposed for *failure-prone* systems.

2.4.18 Failure-free protocols

Most proposed protocols, either in static or mobile distributed systems, assume a reliable environment (i.e. reliable processes and links) and rather concentrate on improving performance while reducing causal ordering related costs: computational and communication costs. A major cost is the size of control information that needs to be stored and exchanged to maintain causality. This latter has to be taken into account seriously as the consequences on the performance effect of the protocol, highly depend on it.

Apart from this general goal applicable to both static and mobile distributed systems, protocols for mobile systems are also concerned with the peculiarities of these environments namely, mobility, low bandwidth of the wireless links, tight constraints on power consumption and a significantly lower computing capability. Thus, protocols need to be specifically tailored to tackle these constraints, so as to minimize resources usage for a wireless environment.

2.4.19 Failure-prone protocols

Although causality makes the implementation of fault-tolerant reliable applications easier, it does not provide any such guarantee by itself [176]. Hence, to devise a fault-

resilient causal ordering protocols, researchers take the approach of ensuring that their algorithms work correctly in the faulty environments. In other words, they provide guarantees about causal message delivery despite faults.

A lot of work has been carried out achieving reliable causal delivery in the occurrence of faults [57, 165, 12, 5, 37, 170, 25]. The primary differences between the various solutions is in the nature of the assumptions that are made regarding failure modes. The failures commonly considered include: *fail-stop* [167] and *Byzantine* failures for processes and *omission* and *timing* [1] failures for links.

Crash failure. A crash failure is when a process fails by halting permanently. This means that it stops performing any activity including sending, transmitting, or receiving any message.

Byzantine failures. Byzantine failures are the most general type of failures. In the model with Byzantine faults, faulty processes may exhibit arbitrary behavior or even be controlled by an adversary. For instance, a faulty process may send messages including incorrect data or also incorrect header, may not send messages, and may send messages to incorrect destinations.

Omission failures. A message inserted in an outgoing message buffer never arrives at the other ends incoming message buffer.

Timing failures. Timing failures occur when the arrival of a message lies outside a specified real-time interval.

It should be apparent that Piggybacking-based protocols (e.g. [57]) will work with unreliable networks and processes that have stronger fault models than fail-stop. The problem of fault-tolerance in causal ordering protocols that allow sending messages to overlapping, but different, sets of recipients has been pointed out in [165]. Aiello *et al.* [12] use a *centralized* control based on the rotating coordinator paradigm and history buffers to recover from omission failures. The protocol proposed in [5] provides fault-tolerance by forcing processes to wait until their previous message is stable (i.e. delivered to all recipients) before sending the next one, and log their state on stable storage. The protocol by Baldoni *et al.* [37] is resilient to omission and crash failures by relying on group communication service and message stability mechanism. [170] supports the reliable delivery of messages in the presence of the Byzantine faults of processes. *ACP* [147] anticipates the detection of crashed processes by using *null messages* (See Section 3.2.1 (Page 33)). The algorithm presented by Dominguez *et al.* [81] tolerates the loss of messages by increasing the redundancy in the control information timestamped per message. This allows the recovery of the system in the presence of lost messages in a *forward error correction* manner. The protocol in [25] considers the omission failure for wireless channels.

Considering causal ordering protocols by application domain, we notice that real-time protocols are all tolerant to timing and omission failures while message logging protocols tolerate crash failures. Furthermore, protocols supporting group communication usually cope with failures through *group membership* and *virtual synchrony* services (See Chapter 6).

2.5 Conclusion

There is a significant amount of work in static and mobile distributed systems that address the causal ordering problem. To the best of our knowledge, however, no attempts have been made at classifying and comparing these solutions. In this chapter, we gave a contribution with the hope of contributing to fill this void. The main result is a set of classification criteria, a taxonomy and an identification of a number of key requirements that must be considered when designing causal ordering protocols, especially for cellular mobile environments. These requirements, like reduction of control information size, support for mobility, minimal participation by MHs in terms of computation, and communication over the wireless link, and scalability serve as the basis for the design and assessment of the solutions developed in the remainder of this thesis.

Recall that the focus of this thesis is not how to implement causal ordering for a specific application domain, but whether or not causal delivery is a useful concept for mobile applications and the cost for causal delivery will typically pay off. Then, in the next chapter, we will turn our attention to "*causal communication protocols*" class. The rationale behind this choice is motivated by the fact that this class provides a broad, application-independent implementation of causal ordering, letting us to aim at studying the influence of mobile environment characteristics and issues, rather than application-specific requirements, on the design of a causal ordering protocol for such an environment.

Chapter 3

Causal Ordering Algorithms

3.1 Introduction

By utilizing Lamport's causal relation [115], various causal communication protocols have been proposed. To classify these protocols, we define a three-levels classification, as shown in Figure 3.1, based on the following factors: (1) system model, (2) environment assumptions, and (3) type of communication. Then, we first classify protocols with regard to the fact that these protocols are devised for a *static* or *mobile* distributed systems. In both of these classes, protocols are then divided into two categories, namely *protocols do not making any environment assumptions* (i.e. assumptions concerning the group structure or topological network structure); these protocols are only concerned with finding the necessary and sufficient conditions to enforce causal message ordering, and/or finding an optimal encoding scheme to represent and transmit this information and *protocols that assume* certain topology and/or communication patterns, and/or certain group structure. Finally, protocols are one again classified depending on the considered communication paradigm, namely point-to-point (unicast), multicast or broadcast communication.

In light of these classification factors, we will survey and analyze, in this chapter, over thirty published protocols.

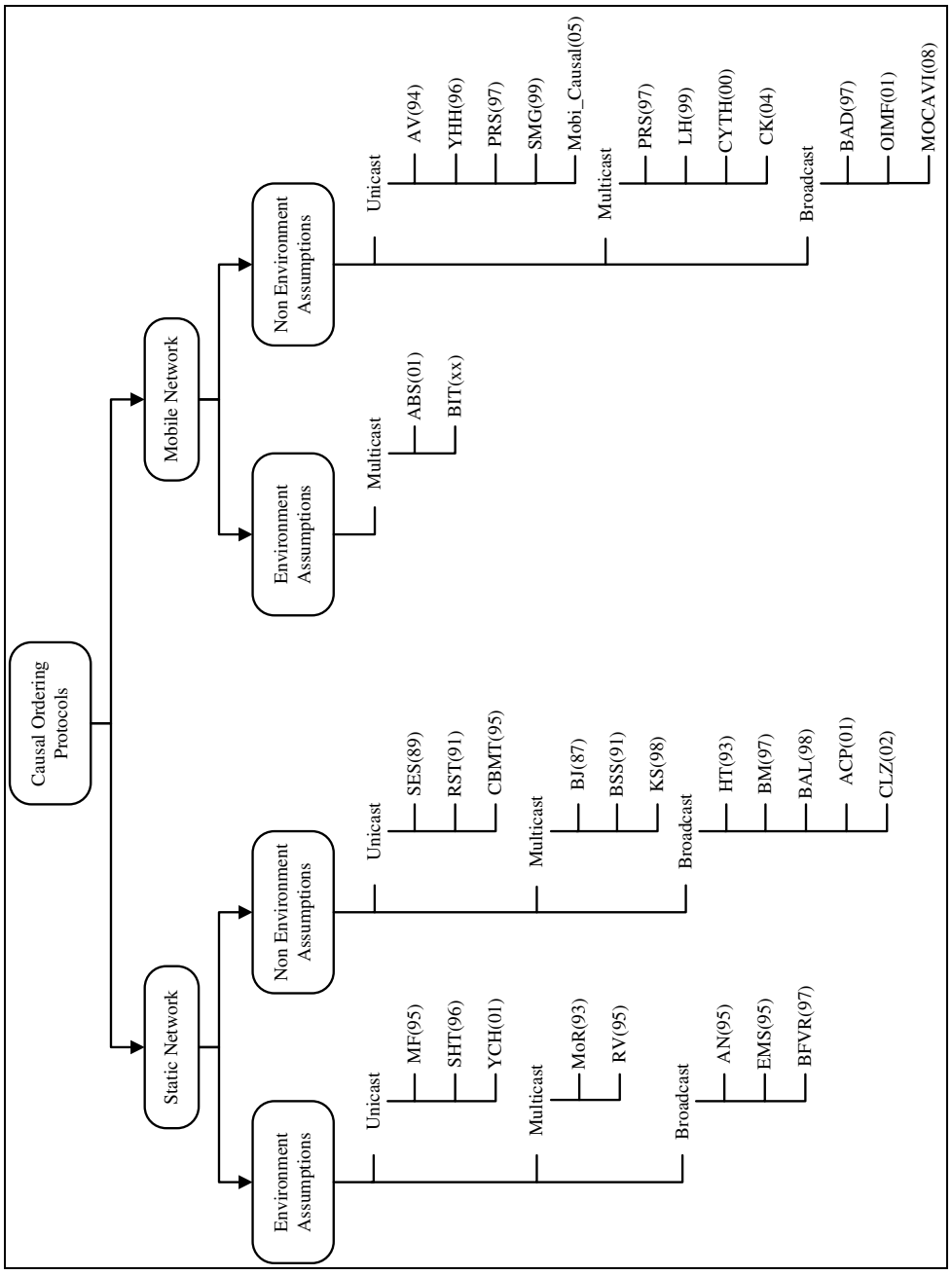

Figure 3.1: Causal ordering protocols classification.

3.2 Causal Ordering Protocols in Static Distributed Systems

3.2.1 Protocols without Environment Assumptions

Birman and Joseph Protocol

This protocol (*BJ*) [57] is the first implementation of causal message ordering. The protocol requires that each message carries with it a copy of all causally preceding messages that have not yet been acknowledged. Suppose a message m reaches a destination process after all its causal predecessors, meant for that destination process, have already been delivered. Then m is also delivered to the destination process. Otherwise, the appropriate pending message(s) are extracted from the list of predecessor messages carried by m and delivered to the destination. Then m is delivered to the destination.

The algorithm was implemented in the *Isis toolkit* as part of a reliable communications transport protocol based on multicast and process groups. It was implemented in an attempt to provide a highly fault-tolerant distributed programming environment.

It should be apparent that the algorithm is exceedingly expensive in terms of message size overhead. Without some form of garbage collection, the message size overhead is unbounded. Even with garbage collection the system is prohibitively expensive, and the current version of ISIS has abandoned it in favor of virtual synchrony.

Schiper, Eggli and Sandoz Protocol

The protocol (*SES*) [166] was motivated by the observation that Lamport logical clocks provide a necessary, but not a sufficient, condition for causal message ordering. Specifically, if causal message ordering has been violated then there exists a message m with a timestamp $T(m)$ which arrives at a destination S where the local time $T(S)$ is greater than $T(m)$. The converse does not hold.

In this implementation, each message m carries control information (and not messages as in *BJ* [57]) consisting of a bounded number of pairs $(destinationsite, vectortime)$. There can be up to $N-1$ such ordered pairs with each message. Thus, this algorithm has a message overhead of $O(N^2)$. The destination process uses the associated control information to determine when it is safe to deliver the message; it uses the associated control information to determine if a causal predecessor of m is yet to be delivered to the destination process. If not, then m is delivered to the destination process. Otherwise, m is buffered at the destination process until all its causal predecessors meant for the destination process have been delivered.

A major limitation of the *SES* algorithm is that it only works in a unicast environment.

Raynal, Schiper and Toueg Protocol

The Proposed protocol (*RST*) [156] is a modified implementation of *SES* [166]. Instead of using vector time, each process P_i maintains an $N * N$ integer matrix $SENT_i$ to reflect the process's best knowledge of the number of messages sent by every process to every other process and an N element integer vector $DELIV_i$ to count the number of messages delivered to P_i from every other process. Every message transmitted by P_i is tagged with the contents of $SENT_i$. On receiving a message, say m, process P_i can determine whether m can be delivered by comparing $DELIV_i$ vector with the i-th column of the $SENT$ matrix tagging m. If m can be delivered, $SENT_i$ as well as $DELIV_i$ are updated. Otherwise, m is buffered until all its causal predecessors meant for the destination are delivered.

The message overhead of this protocol, like *SES*, is $O(N^2)$ integers. However, when a message is always broadcast to all other processes, the matrix is reduced to a vector of length N.

The Horus, Transis and Amoeba implementations of causal message ordering are essentially variants of the *RST* algorithm.

Charron-Bost, Mattern and Tel Protocol

Charron-Bost *and al.* [68] describe a slightly optimized version of *RST* algorithm [156]. The algorithm (*CBMT*) is as follows. When a process P_i sends a message to process P_j it increments the matrix element $M[i, j]$ and appends the matrix to the message. On reception, the receiving process merges the local matrix and the received matrix. This is achieved by element-by-element comparison between the two matrices, selecting the larger as the merged value. A message from P_i to P_j is not delivered until: $\forall k \neq i, M[k, j] \leq M_j[k, j]$ and $M[i, j] = M_j[i, j] + 1$ where M is the transmitted matrix and M_j is the matrix at process P_j . The second condition forces FIFO order from P_i to P_j while the first prevents transitive causal ordering violations.

Birman, Schiper and Stephenson Protocol

The authors [165] pointed the problem of fault tolerance in causal ordering protocols that allow sending messages to overlapping, but different, sets of recipients. The protocol (*BSS*) can be seen as an adaptation of the *BJ* algorithm [57] of causal multicasting in overlapping groups, with a number of optimizations applied. The *BJ* protocol was improved by piggybacking the message identifiers rather than the messages themselves. This approach is formalized by the use of vector time.

This protocol may be summarized as follows: each process maintains a vector of integers per group, with one entry per member indicating the number of messages sent by this member within this group. Each message is timestamped with the whole set of vectors (a vector for each group in the system) of the sending process. When a process

receives a message m, if one or more of m's predecessor messages have not arrived, then m is placed temporarily into a *holding queue* until the appropriate messages arrive. When messages are sent to all processes, the size of timestamps appended to each message is N, where N is the number of processes in the network.

The protocol is currently used by *ISIS*. The major advantage of the current and previous Isis methods is that the handling of message and process failures are bundled with the ordering mechanism, providing a reliable transport protocol with well defined ordering semantics. The preceding approaches require modification or extension to cope with failures. However, *BSS* assumes knowledge of the grouping of processes into disjoint communication groups for optimization. Such knowledge requires a priori analysis of all possible communication patterns amongst the processes: a non trivial task. It would be desirable to have a causal message delivery protocol that does not require such knowledge. The protocol should be able to adapt to the communication pattern inherent in the application.

Kshemkalyani and Singhal Protocol

This work [113] proposes to only piggyback onto every message the bare minimum information concerning all prior causal messages that either are not yet guaranteed to be delivered in causal ordering, or are known to have been already delivered.

The authors identified the necessary and sufficient conditions on the information required for causal multicast, and proposed an algorithm (KS) to implement the conditions.

Let the *causal past* (resp. *future*) of an event e be the set $\{e'|e' \rightarrow e\}$ (resp. $\{e'|e \rightarrow e'\}$). The KS algorithm achieves optimality by storing in local message logs and propagating on messages, information of the form "*d is a destination of m*" about a message m sent in the causal past, as long as and only as long as the following constraints are verified:

- *Propagation Constraint I*: it is not known that the message m is delivered to d, and
- *Propagation Constraint II*: it is not known that a message has been sent to d in the causal future of $Send(m)$, and hence it is not guaranteed using a reasoning based on transitivity that the message m will be delivered to d in causal ordering.

Let $m.Dests$ denote the set of destination of m. The propagation constraints also imply if either (I) or (II) is false, the information "$d \in m.Dests$" must not be stored or propagated, even to remember that (I) or (II) has been falsified. In addition to the propagation constraints, the algorithm follows a delivery condition: A message m' that carries information "$d \in m.Dests$", where message m was sent to d in the causal past

of $Send(m')$, is not delivered to d if m has not yet been delivered to d. The message overhead of KS can be, in the worst case, $O(N^2)$.

Melliar-Smith and Moser Protocol

In [131, 132] Melliar-Smith and Moser describe the *Trans* protocol, a reliable causal order protocol that exploits the hardware broadcasts of local-area networks. By attaching an acknowledgment for the last message received in causal order to the next message sent and by exploiting transitivity of acknowledgments, processors can construct a chain of causal dependencies. The Transis system [21] adopted the same approach.

Hadzilacos and Toueg Protocol

Hadzilacos and Toueg [94] described an algorithm (HT) that is essentially the same as BJ algorithm [57]. The difference is that it is based on causally ordered broadcasts and so only those messages delivered since the last causally ordered broadcast need to be rebroadcast. The idea of transmitting prior messages does have some merit, even though the cost is excessive. To reduce the cost,as in BSS protocol, message identifiers, rather than messages, are transmitted.

Badache and Maddi Protocol

In [35, 34], authors have proposed a gradual design of a causal broadcast protocol for a static group.

The first attempt was a protocol ($BM1$) based on *events' identifiers* [169]. In this protocol, each message carries the projection of the causal history of its sending process. Thus, the amount of control information carried with each broadcast message increases indefinitely. To deal with the drawback of $BM1$, a second attempt was done. It consisted in a protocol ($BM2$) based on *states' numbers* [158]. The causal delivery condition in $BM1$ is replaced by the following inequality: $hist \leq HC_i$ where $hist$ is the state's number carried by the message received by a process P_i and HC_i its current state's number. The use of states' numbers in the construction of the causal history has the advantage of representing the causal history by an integer. Unfortunately, a state's number increases exponentially and consequently needs a mechanism to periodically bend its value (like mechanisms used in [143, 182]). The last attempt was a protocol ($BM3$) that intends to reduce the amount of control information in $BM1$ by deleting the redundant constraints and stopping the propagation of constraints that are no more needed to respect the causal message delivery. So, It will be sufficient to communicate to each receiver of a message, the last message delivered and received from each process in the group. The message overhead of $BM3$ is then reduced to $O(n-2)$, where n is the number of processes in the broadcast group.

Baldoni Protocol

This paper [36] introduced the notion of *causal window* to overcome the unbounded variation of a vector time's counters problem and proposed two causal broadcasting protocols (*BAL1*, *BAL2*) which exploit a *positive acknowledgement* method to limit the width of this window. A causal window of a process represents the range of variation in which all counters of a vector time of a just arrived message at that process fall.

The construction of control information, in both protocols, is based on a *modulok* (k greater than the width of the causal window) implementation of vector times. This implementation ensures causal ordering on deliveries by piggybacking, on each message, a vector time of $n(\log k)$ bits.

The protocols follow a stop-and-wait approach: The sender blocks until the receiver acknowledges message reception. In the first proposed protocol ($BAL1$), a process broadcasts a message and waits for all the acks before executing any other broadcast event. The drawback of this protocol lies in the fact that only one message at a time can be sent by each process, which implies a synchronization between any pair of successive messages sent by the same process. This reduces the potential concurrency of the message and induces a severe latency penalty on the transmitting process. To reduce the number of local synchronizations, authors presented a second protocol ($BAL2$) where a process can send a sequence of consecutive messages before receiving the corresponding acks.This reduction of local synchronization is payed by wider causal window.

Pomares Hernandez, Fanchon, Drira and Diaz Protocol

The proposed protocol (ACP) [147] is based on the propagation constraints established by Kshemkalyani [113]. The receiving host uses vector time to determine whether some message(s) need to be delivered before the current message is delivered. The sending host uses the principles proposed by [113] to transmit the bare minimum causal dependency information using the propagation constraint principle.

The authors bounded the amount of control information by forcing a process to generate automatically a *null message* each time that the control information becomes high. An heuristic mechanism is defined that aims to find the optimal size of control information in order to maintain a trade-off between the number of null messages and the control information size. The null message acts as a reset that reduces to zero the amount of control information. While the protocol guarantee causal delivery between normal messages, it only guarantees FIFO property between null messages.

The authors used another mechanism to reduce the amount of control information. It consists in partially "removing" from the communication all processes that have remained "mute" for a long time. These processes are no longer permitted to participate in the communication; they simply become listeners. Then, the process must rejoin the group in order to be allowed to participate again in the communication.

Cai, Lee and Zhou Protocol

The proposed protocol (CLZ) [64] is an improved version of the Prakash's algorithm (PRS) [151], that tries to eliminate indirect dependencies in broadcast as well as a multicast environment that PRS fails to eliminate.

This algorithm differs from Prakash's algorithm in the way that the delivery information is inferred and that the non-immediate predecessors are eliminated. Instead of using $Delivered_{P_i}$ and $CB_{P_i}[i]$, process P_i uses a vector time VT_{P_i} to keep track of the dependencies between messages and the information on messages that have been delivered at each process.

The elimination of non-immediate predecessors in CLZ does not come without a price. The price is paid with an increased size of the control information. The control information carried with messages is the same to that of Prakash's algorithm except that the counter Cnt_i is replaced by the vector VT_{P_i}. Note that the size of VT_{P_i} (i.e. $O(N)$) is greater than that of Cnt_i (i.e. $O(1)$). The message overhead of CLZ is $O(N^2)$.

The protocol was implemented in a middleware [175] to the implementation of a time management mechanism based on causal ordering delivery.

3.2.2 Protocols with Environment Assumptions

F. Mattern and S. Funfrocken

Mattern and Funfrocken [129] (MF) took the approach that rather than attempt to strengthen a FIFO algorithm, they would weaken a synchronous algorithm as much as possible while still maintaining causal message ordering. This led to the idea of implementing a *non-blocking synchronous* system. Each process maintains separate input, output and computation threads. The input and output threads consist of FIFO ordered buffers of messages. When a process (that is, the computation thread) performs a transmission, it simply copies the message to be transmitted to the output buffer. Thus, it does not block. Conversely, when it wishes to read a message, it reads one from the input buffer, or blocks until one is present. The output thread transmits messages in FIFO order from the output buffer. After transmitting a message, it will block until an acknowledgment is received. This ensures that the message must have been received before another message can be transmitted. Therefore no message can overtake another. The input thread simply places messages, in the order of reception, in the input buffer and then transmits an acknowledgment to the sender.

The protocol achieves a zero cost message size increase at the expense of $N \times T_w$ *non-causal latency*, where T_w is the worst message time between a process and any other process. Non-causal latency is latency due to the algorithm that is not causal.

Shima, Higaki and Takizaw Protocol

The authors [170] proposed an intra-group communication protocol (*SHT*) which supports the causally ordered delivery of messages for the processes within the group. In addition, the protocol supports the reliable delivery of messages in the presence of the Byzantine faults of the processes, i.e. they may send messages including not only incorrect data but also incorrect header, may not send messages, and may send messages to incorrect destinations. The faults are assumed to be independent.

In order to support the fault-tolerant group communication, each process is replicated into a collection of multiple *replicas*, which is named a *cluster*. Each process P_i is replicated to be a cluster of $li(\geq 2 * fi + 1)$ replicas if at most fi replicas are faulty in cluster of P_i.

In order to realize the fault-tolerant transmission of a message m from P_i to P_j, each replica in a cluster of P_i sends a message m to multiple replicas in a cluster of P_j in the group. In order to detect faulty replicas, each replica receives messages from multiple replicas.

The control information carried by messages is represented by the sequence number sn and the acceptance confirmation $ack_1, \ldots, ack_n$, where ack_i denotes the sequence number sn of the message which P_i expects to receive next from Pj.

The replicas in cluster of (P_j) have to receive message instances from more than $2 * fi$ replicas in cluster of P_i. If the replicas in cluster of P_j receive the same instances of a message m from more than fi replicas in cluster of P_i, the replicas can accept m.

Each replica in cluster of (P_j) decides that "$m_1 \rightarrow m_2$" if more than fi replicas in cluster of (P_i) notify that m_1 causally precedes m_2.

Yen, Chi and Huang

The scheme (*YCH*) [186] organizes the system into hierarchical clusters. Within each cluster a specific process is designated as the *agent* of the cluster, and all others are referred to as *clients*. All messages into or out of a cluster must be queued and served serially at the agent. Within a cluster, any existing causal message ordering (*CMO*) methods can be locally adopted. The approach decomposes system-wide *CMO* into a number of independent cluster-wide *CMOs*.

Authors extended the abstraction of conventional approaches by introducing at every site a *message relay* part (*MR*) between the application process and the causal message delivery part (*CMD*). *MR* takes charge in relaying messages between clusters. An intra-cluster message will be sent directly to its destination, whereas an inter-cluster message will be first directed to the origin's agent, then toward the destination's agent, and finally to the destination process. Since every cluster employs its own *CMO* protocol, the relay is accomplished by the locally adopted cluster-wide *CMO* protocol. An agent, which involves in communications between two clusters, must maintain two

independent *CMD* processes, one for each cluster. So, a site participating in k ($k >= 2$) clusters must maintain k independent *CMD* processes, one for each cluster. Authors assume that the relaying between two *CMD* processes must be done in FIFO order. Another restriction is that *MR* must relay all messages along the same direction.

As an implementation of the CMD process, authors adopted in every cluster the *RST* algorithm [156]. Compared to *RST* but provided that the cluster size is appropriately set, the message overhead can be reduced to only $O(N)$ or even $O(logN)$ if the whole system forms a two-layer or more than two-layer hierarchy, respectively. Whereas the message overhead decreases with the increasing of layers, the number of intermediate nodes to be traversed for propagating messages, namely hop count, can increase when more layers are introduced into the system. This implies longer communication delay; the effect of longer routing paths.

As mentioned earlier, authors claim the possibility of adopting a different *CMO* protocols for each cluster. This needs to bring various adaptations in order to make possible the relaying of a message through the different CMD processes. Nevertheless, we believe that such adaptations are obviously difficult and not straightforward due to the fact that *CMO* protocols relay on different approaches to track causality between messages and different delivery conditions to ensure causal ordering. Furthermore, these adaptations are computationally expensive.

Mostefaoui and Raynal Protocol

The algorithm [143] (*MoR*) requires some synchronization activities among the processes periodically, thus resulting in a loss in concurrency. It is aimed at reducing the control information of the *BSS* protocol [165]. In this protocol, each message has to carry only one vector whose size is equal to the number of groups. However, a synchronous execution model is assumed which requires additional re-synchronization messages so that events at the processes occur in synchronized phases. Re-synchronization may contribute towards delays in message communication. Hence, the algorithm is suitable only for those applications where delays and re-synchronization messages can be tolerated for reduced overheads in the computation messages.

Rodrigues and Verssimo Protocol

Rodrigues and Verissimo (*RV*) [162] observe that communication overheads can be reduced by compressing causal information using knowledge about the topology of the underlying communication structure. In essence, since every message from process P_i to process P_j must pass through some subset of the vertices of the physical network, those partitioning vertices may be used as a *causal separator*. Using causal separators, the causal information concerned with the elements of a *causal zone* (a set of processes bounded by one or more causal separators) may never need to be propagated outside

the boundaries of that causal zone. More simply, we need only transmit the causal history of the separators across the boundary.

Their approach, however, does not reduce the amount of control information used within the same subnet domain, which is detrimental to scalability. Moreover, a major problem with this approach is that it must either be implemented at the transport layer or it must be implemented as a logical network on top of a physical network. If it is implemented at the transport layer, then it slows down all messages routing, even for applications that do not require causal ordering. If it is implemented as a logical network, then it is likely that the mapping to the real network will cause severe efficiency problems, since it will potentially create paths that do not exist in the physical network and remove paths from consideration that do exist.

Adly and Nagi Protocol

The authors [5] proposed a causal ordering protocol (AN) on unreliable communication network. It is designed for a replicated system where a message is sent to every replica in the network. The protocol is based on a propagation scheme, HARP[], where nodes are organized into a logical, multilevel hierarchy: nodes are grouped into clusters, and clusters are organized into a tree, such that each cluster is assigned a father node in its parent cluster. The interesting property of this propagation scheme is that a node sends and receives messages from few nodes only. Following the propagation algorithm, a node receives every message, originating at any node in the network, either from a neighbor, the father or a son. Therefore, a node needs to keep track of messages received from those nodes and stamp messages with this information only in order to verify the causal ordering. The protocol cuts down the size of the timestamp appended to each message to be equal to $p+1$, where p is the number of nodes in a cluster. Their solution, however, requires processes to wait until their previous message is stable before sending the next one, and log their state on stable storage.

Authors have also discussed methods to deal with failures and network partitions.

Baldoni, Friedman and Van Renesse Protocol

The protocol(BFVR) [37] assumes certain group structure. The authors split the participating processes into *local groups*, and use causal servers to disseminate messages across these groups in the presence of processes faults. Thus the amount of control information is reduced to the local group size. All causal servers are also members of causal servers group.

The drawback of this technique is that the use of causal servers adds a delay to the delivery of messages that need to cross local group boundaries. Therefore, this technique is not suitable for applications with time constraints. The protocols that fall into this category are not easily scalable.

3.3 Causal Ordering Protocols in Mobile Distributed Systems

3.3.1 Protocols without Environment Assumptions

Alagar and Venkatesan Protocol

Authors [14] presented a suite of algorithms ($AV1$, $AV2$, $AV3$) that all use RST [156] to enforce causal order. These algorithms assume reliable wireless and wire-line networks. They use the fact that the wireless channel between MSS and each MH is FIFO, thus the log for each MH for causal ordering is maintained at the MSS level rather than at MHs.

In $AV1$ each message carries dependency information with respect to all the processes (i.e. to all MHs in the system). The algorithm is not scalable due to high communication overhead which is of $O(n_h^2)$. However, $AV1$ has a simple handoff procedure requiring $O(1)$ message only.

The second algorithm ($AV2$) eliminates the drawbacks of the $AV1$. $AV2$ is smart in realizing that it is sufficient only to maintain causal order among the $MSSs$; each MSS acts as proxy for all MHs in its cell, leading to $O(n_s^2)$ message overhead. Nevertheless, this yields to different vision between $MSSs$ and MHs on order of events, which involves 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. Therefore, reception of a message may violate causal ordering from an MSS's point of view; whereas its delivery to an MH may not violate causal ordering from the MH's point of view. Moreover, in this protocol, when handoff of the MH occurs to guarantee the causal relation, it is necessary to make an inquiry to all $MSSs$ resulting in $O(n_s)$ message exchanges to handle mobility.

To reduce the delay in delivering the messages to an MH due to inhibition, authors proposed a third algorithm ($AV3$). The algorithm achieves this by partitioning every physical MSS into k logical $MSSs$. The algorithm is the same as $AV2$ except for the fact that causal ordering is explicitly maintained among the logical $MSSs$. The size of the message header is $O(k^2 * n_s^2)$. Messages to MHs that belong to different logical $MSSs$ will not inhibit each other though the MHs may be in the same cell. Thus, as k increases, the unnecessary delay in delivering the message to MH decreases. However, as k increases, the size of the message header will increase and, as a result, the time to process the message header will become a dominating factor.

The computation load and the communication overhead on MHs are both low in all these algorithms, though the message overhead varies. However, none of these algorithms handles mobility and disconnection in an efficient way. $AV1$ is not scalable for frequent disconnections, while $AV2$ and $AV3$ require a time-consuming procedure

to handle host migrations. Hence, there is a need for algorithms that minimize the communication overheads as well as the delays in message delivery.

Yen, Huang and Hwang Protocol

Yen *et al.* (*YHH*) [185] proposed an algorithm that adapts *RST* [156] to achieve causal order in mobile networks. Each base station S_i maintains a matrix MSS_SENT_i with $N_s \times N_h$ entries where the j^{th} row denotes the knowledge of S_i about the number of messages sent by S_j to each mobile host h_k. Each message m transmitted by h_i is first received by its base station S_i. S_i appends its MSS_SENT_i matrix to message m and directs m to its destination h_j. Then, S_i increments the entry $MSS\_SENT_i[i,j]$ by 1. To each mobile host h_i in the system, a vector with n_s entries, denoted MH_DELIV_i, is associated, where the j^{th} entry of MH_DELIV_i denotes the number of messages sent by the base station S_j which are delivered to h_i. This vector is stored at the h_i's base station level. When the message m, destined to h_j, is received by h_j's base station, let be S_l, this last compares the content of MH_DELIV_j with the content of the matrix attached to message m, $m.S$, in order to decide if m can be delivered or not. A message m is delivered if $MH\_DELIV_j[k] \geq m.S[k,j], \forall k \in \{1, \cdots, N_s\}$, else the message m is put in the queue $PEND_j$ until the delivery condition of this message becomes satisfied. In the case where the message m is deliverable, S_l inserts it in the queue $WAIT\_ACK_j$ which serves to keep the message delivered to the mobile host h_j until the reception of a receipt acknowledgement from the destination host h_j. Upon the arrival of this acknowledgement, S_l executes the following actions: $MSS\_SENT_l[i,j]{+}{+}$; $MSS\_SENT_l = max(MSS\_SENT_l, m.S)$ and finally deletes m from $WAIT\_ACK_j$. The control information's size attached to messages in this algorithm lies between that of $AV1$ and $AV2$. Particularly, the message overhead is $O(n_s \times n_h)$, where n_h is the number of MHs. The unnecessary inhibition delay in this algorithm is lower (more less) than $AV2$. Its handoff module is also more efficient than $AV2$, while the drawback of this algorithm is the violation of liveness property [171]:

In figure 3.2, we present a scenario where the protocol YHH does not satisfy the liveness property. According to YHH, the delivery of message m_4 will be delayed because $m_4.M[1,2] > MH\_DELIV_2[1]$. And since after the arrival of m_4 to S_2, no other message is sent, the delivery of m_4 is indefinitely delayed.

Skawratananond, Mittal and Garg Protocol

The protocol (*SMG*) [171] is a variant of $AV2$ [14]. *SMG* maintains for each mobile host h_l a matrix $M_l[n_s, n_s]$, where $M_l[i,j]$ denotes the knowledge of h_l about the number of messages sent by S_i to S_j. Moreover, each base station S_i maintains two vectors, $lastsent_i[n_s]$ and $lastrcvd_i[n_s]$ where $lastsent_i[j]$ denotes the number of messages sent by S_i to S_j and $lastrcvd_i[j]$ denotes the number of messages sent by S_j which are

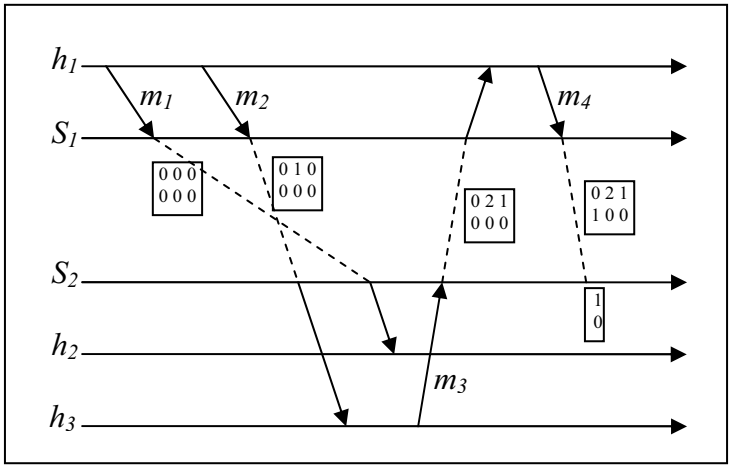

Figure 3.2: The liveness problem in the protocol YHH.

delivered to S_i. All these data structures are stored on the base stations in order to reduce the storage cost on the mobile hosts. To send a message m to another mobile host h_d located in the cell covered by the base station S_j, h_s sends first the message to its base station S_i. S_i increments then $lastsent_i[j]$ by 1 and sends $(m, M_s, lastsent_i[j])$ to S_j. After that, S_i sets $M_s[i,j]$ to $lastsent_i[j]$. Upon receiving the message $(m, M, seqno)$ by S_j, this last checks if m can be delivered. The message m can be delivered if $(lastrcvd_j[k] \geq M[k,j])$ and $(\nexists(m', M', seqno')$ sent by S_k to h_d and not yet delivered such as $seqno' \leq M[k,j])$. If the condition is satisfied the message is delivered to h_d, otherwise the message is inserted in $rcvQ_j$ until its delivery condition becomes true. Similarly to YHH, this protocol uses a queue $ackQ_d$ to keep the delivered message until the reception of acknowledgement asserting its reception by the destination host h_d. In this case, $M_d[i,j] = max(M_d[i,j], m.seqno)$ and $M_d[k,h] = max(M_d[k,h], m.M)$ for all k, h. The message overhead of this algorithm is $O(n_s^2 + n_h)$.

Benzaid and Badache Protocol

Mobi_Causal protocol [47, 43] is based on the timestamping mechanisms proposed for mobile environment: *dependency sequences* [152] and *Hierarchical clocks* [152] in order to compensate the loss of mutual concurrency information caused by the use of vector clocks as timestamping mechanism, with one entry for each MSS. This compensation permits the elimination of unnecessary inhibition delays.

A *dependency sequences* approach keeps with each event e (*send* or *receive* event), 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.

An *hierarchical clock* ϕ captures the causal dependency relation between events

(*send* and *receive* events) by two components: (1) ϕ^i is a local clock representing the causal dependency relation between events occurring within the same cell. It is a variable length bit-vector with one bit for each event that has occurred in the cell thus far. A bit of ϕ^i is set to 1 if the corresponding event causally precedes the current event, otherwise is set to 0. (2) ϕ^m is a global clock representing the causal dependency relation between cells. It is an integer-vector of n components, one for each cell in the system. The k^{th} component of event's global clock identifies the last event on cell covered by a base station S_k that causally precedes this event. the event is timestamped by the global clock only.

It is interesting to notice that the dependency sequences approach has high communication overhead but incurs no extra effort or time delay to determine the past of an event. The hierarchical clock approach has low communication overhead, but determination of the past of an event takes a large time delay.

To profit from the advantages of the two approaches, we combined them in order to implement a new protocol ensuring the causal delivery. Since the causal past of an event is directly determined from the set of its dependency sequences, then it is more interesting to use the dependency sequences mechanism to keep on information about messages received by a mobile host. This makes it possible to check directly if a message is deliverable or not without additional computations.

Information about the last messages sent by h_i to other mobile hosts h_j is stored in a set, $LastSend_{h_i}$. One component of the set is denoted by (h_j, id, ϕ^i) where id is the identifier of the last message sent by h_i to h_j and ϕ^i denotes the local clock value. This clock is represented by a bit vector of variable size and its construction is based on the principle of hierarchical clocks [152]. The maximal size of ϕ^i is equal to number of messages sent between two given receptions. The clock ϕ^i has the following representation: $B_D C_{Int} B_E$. Where, B_D is a bit that is equal to 1 if the mobile host has already received a message, otherwise this bit is omitted. B_E is a bit that is set to 1 and represents the current message to be sent. C_{Int} is a range of bits with a size equal to the number of messages sent after the last received message by the mobile host and before the message to send. One bit of C_{Int} is equal to 1 if the corresponding message is already sent to the same mobile host, and 0 else.

In the example in Figure 3.3, the value of ϕ^i associated to the emission of m_1, sent by $h_1(S_1)$ to $h_4(S_3)$, is equal to 1 (i.e. B_D omitted, $B_E = 1$ and C_{Int} empty) because m_1 is the first message sent by h_1 and the entry $(h_4, 1, 1)$ is added to $LastSend_{h_1}$. Whereas the value of ϕ^i associated to the emission of m_2 is equal to 01 (i.e. B_D omitted, $B_E = 1$ and $C_{Int} = 0$ because m_2 is sent to h_3 whereas m_1 is sent to h_4) and the entry $(h_3, 2, 01)$ corresponding to the emission of m_2 is added to $LastSend_{h_1}$.

Each mobile host h_i has also a set $LastRcv_{h_i}$ that stores identifiers of messages received by h_i from h_j. One component of the set is denoted by (h_j, S_j, SD_j) where

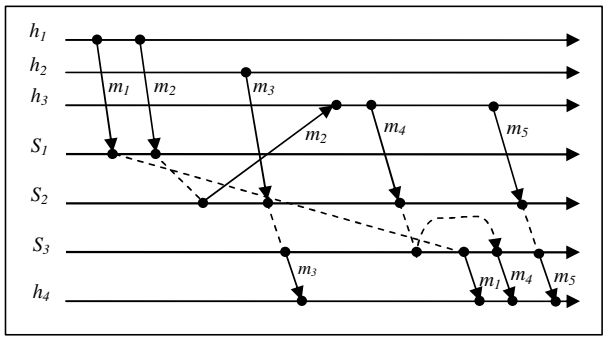

Figure 3.3: *Mobi_Causal*: An illustrated example.

$SD_j = \{id_1, id_2, id_3, id_4, ...\}$ and corresponds to a dependency sequences. So, its construction is similar to that of dependency sequences mechanism [152].

Each mobile host also maintains a vector ϕ of length n_S (where n_S is the number of base stations). One entry $\phi_m[k] = \{(h_j, id), ...\}$ in the vector ϕ 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. The delivery of these messages involves the update of ϕ by eliminating the corresponding entries from ϕ.

The message to be sent should have the following form: (m, $IdSend_{S_i}$, $depend_{h_i}$, $Id_{LastSend}$, ϕ). $IdSend_{S_i}$ is the identifier of message m. $depend_{h_i}$ is a Boolean variable which the value depends on ϕ^i; if ϕ^i has the form 0*1, which means that this message is not constrained by any other message, then the value of $depend_{h_i}$ 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. In the example, this is the case of the message m_2 then the value of $depend_{h_1}$ associated to m_2 is set to $false$. So, this message is immediately delivered when it is received by its receiving base station and its identifier is added to $LastRcv_{h_3}$ ($LastRcv_{h_3} = \{(h_1, S_1, \{2, 2\})\}$). If not, ϕ^i has not the form 0*1, $depend_{h_i}$ will be set to *true*, which means that this message depends on whether an internal message of mobile host h_i (i.e. depends on message sent by h_i to h_j before it. In the example, this is the case of the message m_5 which depends on the message m_4 sent by the same mobile host h_3 then the value of $depend_{h_3}$ associated to m_5 is set to $true$.), or it depends on an external message (i.e. that h_i has received a message before the emission of m. In the example, this is the case of the message m_4 which is sent after the reception of the message m_2 then the value of $depend_{h_3}$ associated to m_4 is set to $true$).

The message must transmit also, in the control information, the identifier of the last message already sent by h_i to h_j if it exists. So, if there is $(h_j, id, \_) \in LastSend_{h_i}$, the value id is assigned to $id_{LastSend}$, otherwise $id_{LastSend}$ will be assigned the value 0

to say that is the first message sent from h_i to h_j. In the example, the $id_{LastSend}$ of the message m_5 is set to 2 which corresponds to the identifier of the message m_4, however the $id_{LastSend}$ of other messages is set to 0.

In the case where $id_{LastSend_m} \neq 0$, the delivery of a message m depends on the reception of a message with an identifier $id_{LastSend_m}$ by h_j. The reception of a message by a mobile host is expressed by an entry (h_i, S_i, Id) in the set $LastRcv_{h_j}$ where $Id = id_{LastSend_m}$. In the example, at the reception of the message m_5, it suffices to verify that the message m_4 has been delivered to h_4 (i.e. verifying that the identifier of m_4 is included in the set $LastRcv_{h_4}$). Since the reception of m_5 has been occurred after the delivery of m_4 which is expressed by the entry $(h_3, S_2, \{2, 2\})$ in the set $LastRcv_{h_4}$, then m_5 is delivered to h_4.

In the case where the causal dependency is created by the reception of a message sent by another mobile host to h_i, the delivery of a message m depends on the delivery of its immediate predecessors on different base stations. These predecessors are identified by the global clock ϕ. For each entry in ϕ where it exists a predecessor sent to h_j, we search if it exists a 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 have been received and consequently m will be delivered to h_j.

Otherwise, if a received message m can not be delivered then we say that the delivery condition is not verified and consequently m is queued in a file, denoted $AtFile_{S_i}$. In the example, at the reception of the message $(m_4, , 2, true, 0, \begin{bmatrix} (1, h_4) \\ 0 \\ 0 \end{bmatrix})$ by the base station S_3 and when verifying ϕ_{m_4}, we find that m_4 depends on the first message sent by S_1. Hence $LastRcv_{h_4} = \{(h_2, S_2, \{1, 1\})\}$ does not contain information about the arrival of this message, then the delivery of m_4 will be delayed and m_4 will be queued in $AtFile_{S_3}$.

The delivery of a message m induces the delivery of all queued messages in $AtFile_{S_j}$ that are waiting for m. The delivery of these messages involves their deletion from $AtFile_{S_j}$. In the example, the delivery of the message m_1 to h_3 involves the delivery of the message m_4 waiting for m_1 and its deletion from the queue $AtFile_{S_3}$. The delivery of m_4 involves also the deletion of tuple $(1, h_4)$ from ϕ_{h_4} because this message is delivered and it is not necessary to transmit this information with the next messages.

Prakash, Raynal and Singhal Protocol

The algorithm (*PRS*) proposed by Prakash *et al.* [150, 151] combines an improved version of the *RST* causal order algorithm for static networks with that of Badrinath [3], an algorithm for multicast with "exactly-once delivery semantics" in mobile cellular networks. *PRS* assumes all wire-line and wireless networks to be reliable.

To reduce the amount of the control information, *PRS* appends only *direct depen-*

dency information to the messages. It is a refinement that observes that it is sufficient if only the immediate causal predecessors are known, rather than the entire transitive history. The algorithm is optimal in a broadcast environment. However, in a multicast environment, it fails to eliminate indirect dependencies in some cases because message delivery information propagates at most one message-hop away from the destination process:

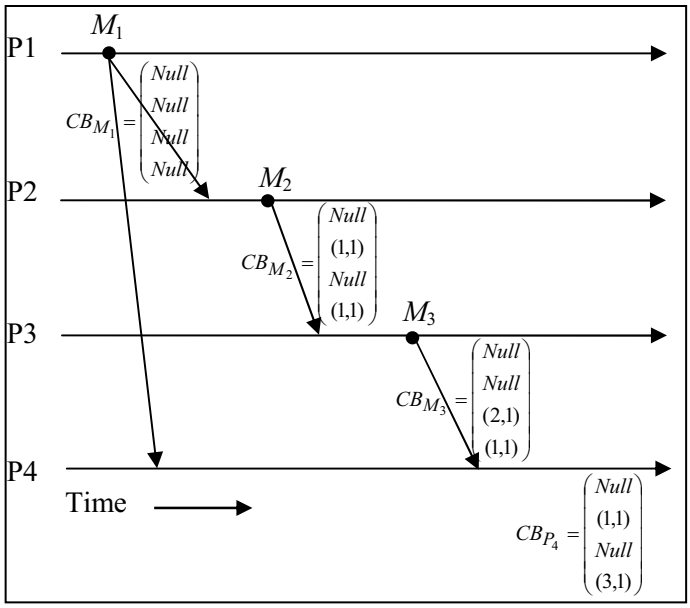

Figure 3.4: The indirect dependencies elimination problem in PRS.

In this example, since the delivery of message m_1 at process P_2 is two message-hops away from process P_4, PRS fails to infer that m_1 is no longer an immediate causal predecessor of any future message that may be sent by P_4 to P_2 and thus is unable to delete $(1, 1)$, the signature of message m_1 from $CB_{P_4}[2]$.

In their algorithm, each process P_i maintains a counter Cnt_{P_i} to keep track of the sequence number of messages it has sent so far to all other processes. The matrix $Delivered_{P_i}$ at each process stores the process's knowledge of the latest messages delivered to other processes. $Delivered_{P_i}[l, k] = x$, therefore, indicates that process P_i knows that all the messages sent by process P_l to process P_k with message sequence number less than or equal to x have already been delivered to process P_k. Each process P_i also keeps a vector CB_{P_i} of length N (i.e., the number of processes in the system) to capture information on its immediate causal predecessors. In particular, $CB_{P_i}[k]$ maintains the delivery constraints for the messages to be sent by process P_i to process P_k. It is a set of tuples of the form $(sender'spid, message\ sequence\ number)$. Every message carries a sender's message sequence number Cnt_{P_i} , a sender's pid i, the set of destinations of the message $Dest(m)$, and sender's causal barrier vector CB_{P_i}. The newly sent message will become the immediate causal predecessor of any future mes-

sage to be sent to the same destination. A process P_i can only deliver a message m from a process P_j if all m's causal predecessors have already been delivered to P_i.

The multicast message overhead and storage overhead of PRS is $O(n_h^2)$. In this protocol, the message header to be used becomes smaller when it is applied to the broadcast; the message overhead becomes $O(n_h)$. However, the message overhead of this protocol depends on the total number of mobile hosts n_h.

Li and Huang

This algorithm (LH) [121] does a broadcast among MSSs, and then within each cell to MHs. The protocol uses *two levels* of delivery: the *first level* corresponds to the MSS level delivery and the *second level* corresponds to the MH level delivery. Each MSS S_i maintains a vector of length n_s, called $DELIV_i$, to track the delivery information for S_i. The vector records the number of messages initiated from other $MSSs$ which have been delivered to S_i. Each MSS S_i maintains also two vectors, $DELIV\_MH_i$ and $SENT\_MH_i$, of length n_s for each of its local MH h_i. $DELIV\_MH_i$ denotes the greatest sequence number, on per-MSS basis, that has been delivered to h_i. $SENT\_MH_i$ denotes the knowledge of h_i about the number of messages that have been initiated from each MSS. The two vectors will be transferred to the new cell if the MH moves to a new cell.

The control information consists of the initiator id, destination ids and $SENT\_MH_i$ vector. Using this control information, the MSS level delivery condition ensures that S_i has delivered all the messages initiated by S_j that precede m and that S_i has delivered all those messages received by h_i before h_i makes the request. If the MSS level delivery condition is not verified, the message will be buffered until its causal predecessors meant for the MSS are delivered. However, if the message is delivered, it will be put in MSS_BUFER_i to terminate the MSS level delivery of the protocol. The MH level delivery of the protocol compares $DELIV\_MH_i$ with the timestamp of the message for each local destination h_i to determine if the message has been delivered to the destination MH at other MSS or not. If not, the message will be put in a deliver queue, $DELIV\_Q\_MH_i$, for each h_i. The wireless module then forwards the messages in $DELIV\_Q\_MH_i$ one by one to h_i.

The message overhead of the algorithm is $O(n_s)$. However, the protocol suffers from the introduction of unnecessary inhibition delays problem.

Chi, Yen, Tseng and Huang Protocol

Another causal multicast protocol, ($CYTH$), is presented in [71] where only a part of *MSSs* (called *mobility agents*) is involved in group computations. An MH always initiates causal multicasts via its *serving agent* (i.e. *MSS* of cell where it has been located for the first time) regardless of its current location. As consequence, *MHs*

appear stationary and this simplifies the development of the protocol. However, if we imagine the case where each *MSS* is a serving agent for at least one *MH* member of the group, then all *MSSs* will be included in the multicast group even if there is no member located in their cells.

The $CYTH$ [71] protocol provides the best handoff complexity $O(1)$ but its handoff procedure involves three entities to achieve the migration of an MH (its last MSS, its new MSS and its serving MSS). Each time a migration is done, the serving MSS must be contacted to update the current location of an MH. Thus, a supplementary delay is invoked in the execution of the handoff procedure and which can be considerable if the serving MSS is so far.

Chandra and Kshemkalyani Protocol

The proposed protocol (CK) [65] combines KS algorithm [113] and Badrinath's algorithm [3]. Every mobile host h has a log log_h and an array $RECD_h[1..n_s]$ associated with it. The log_h is used for maintaining causal order with respect to h, while the array $RECD_h$, initialized to all zeros, is used to guarantee that each message is delivered once to h. An MSS also maintains a sequence number M_{seq}, which is incremented each time after it forwards a message on behalf of any of the MHs in its cell.

Consider the sending of a message from h_i to h_j. h_i sends the message to its MSS, MSS_{init}, which updates log_i for causal delivery, then appends $M_{id} = (MSS_{init}, M_{seq})$ and log_i to the message, and broadcasts the message to all MSSs. Once the message is sent, MSS_{init} increments M_{seq} by one. When a MSS receives a message M destined for h_j from another MSS, it tests the following conditions: (1) destination h_j is present in the local cell, (2) $RECD_j[MSS_{init}] = M_{seq}$ and (3) the delivery condition [113] is satisfied for M.

If these conditions are satisfied, the message is delivered to h_j. For each h_k that is not a destination, $RECD_k[MSS_{init}]$ is incremented. The message is also buffered, irrespective of the outcome of the tests. On receiving the message, h_j sends an *ack* to its MSS, which now increments $RECD_j[MSS_{init}]$, updates log_j as per KS [113], deletes the buffered message, and forwards the *ack* to MSS_{init}. MSS_{init} then broadcasts message $Delete(M_{id})$ to all MSSs. When a MSS receives $Delete(M_{id})$, it clears any information about message M_{id} from its buffer. Handoff for h is handled by passing $RECD_h[1..n_s]$ and log_h from the old MSS to the new MSS. The transfer is delayed if a message has been delivered to h but the update of $RECD_h[1..n_s]$ and log_h have not taken place.

In spite of defining the necessary and sufficient conditions on the control information, the message overhead of CK is, in the worst-case, $O(n_h^2)$.

Badache Protocol

This protocol (BAD) [33, 34] is an adaptation of $BM3$ [35, 34] to a distributed system with mobile hosts. The protocol assumes a static group of n processes which exclusively communicate by broadcast of messages. The message overhead of the protocol is comparable to the one already obtained in $BM3$ (See Page 33).

Ohori, Inoue, Masuzawa and Fujiwar Protocol

The authors [144] proposed a protocol ($OIMF$) for realizing the causal broadcast among MHs. It is an extension to the Birman's protocol (BSS) [165]. BSS was first extended by reducing the message overhead by realizing the causal broadcast among MSSs. Hence, The message overhead of $OIMF$ is $O(n_s)$.

Authors claim that in the case where the handoff of an MH does not occur, the causal broadcast between MHs can be realized by performing causal broadcasts among MSSs. Thus, they attached a n_s-th-order vector clock to each message to preserve causality. However, this is not totally true because using a n_s-th-order vector clock involves a lost of the concurrency relation between messages sent by different MHs in the same cell. They said that when two MHs h_1, h_2 are connected to MSSs S_1, S_2, respectively, if $cbcast_1(m_1) \rightarrow cbcast_2(m_2)$ holds when the broadcast messages m_1, m_2 are transmitted, respectively, then $mss\_cbcast_1(m_1) \rightarrow mss\_cbcast_2(m_2)$ also holds. But even if $cbcast_1(m_1)||cbcast_2(m_2)$ holds, if h_1 and h_2 are in the same cell, then $mss\_cbcast(m_1) \rightarrow mss\_cbcast2(m_2)$ also holds, incurring a lost in concurrency relationship between messages.

To avoid duplication and absence of delivery messages when an MH migrates between cells, each MSS manages the n_s-th-order vector $RECV_j[h_i]$ which expresses the number of broadcast messages already delivered at each MH h_i connected to S_j. Here, $RECV_j[h_i][S_k]$ expresses the number of broadcast messages already transmitted to h_i among the messages broadcast from MSS S_k. However, to avoid the absence of messages, the broadcast messages already delivered at each MSS S_j are preserved in the queue $DELIV\_MES_j$ in order of delivery.

The header attached to the broadcast message in the static module expresses only the causal relation in the broadcast between MSSs, and does not express the causal relation between the broadcast messages transmitted by the same MH for different MSSs. For this reason, m_2 may be delivered ahead of m_1 in the separate MSS if a receiving MH be handed off between two cells. To avoid this problem, each MSS S_j keeps information about the number of broadcast messages which precede the broadcast messages to be transmitted next for every MSS and that by using a n_s-th-order vector $SENT_j$. In order to preserve the causal relation of the broadcast messages transmitted by h_i before and after handoff, it is sufficient for each MSS S_l that the value of $SENT_k[S_l]$ immediately after handoff of h_i be greater than that of $SENT_j[S_l]$ immediately before

handoff of h_i. Accordingly, the value of $SENT_j$ is transmitted to S_k at the time of handoff, S_k updates its $SENT_k$ vector by keeping the max between the local vector and the received one.

Dominguez, Hernandez and Gomez Protocol

The proposed protocol (*MOCAVI*) [80] achieves a causal broadcast among a static group of MHs according to the causal view that MHs perceive during the system execution, avoiding unnecessary inhibition delays. To achieve this, the tracking of causal dependency relationship in *MOCAVI* is inspired by our prior work [47, 43] where authors have exploited our idea of differentiating between local and global dependency information. Thus, *MOCAVI* distinguishes two communication levels, namely *intra-base* and *inter-base* communication levels. The intra-base communication level provides communication service between a base station and mobile hosts in its cell, while the inter-base communication level provides communication between base stations.

In contrast to our proposal where the execution of protocol is completely supported by the MSSs, *MOCAVI* involves the MHs in the causal ordering of messages. An MH maintains a bit vector Φ_i, for each base station S_i in the communication group. The size of Φ_i is equal to the number of MHs that are within the cell of S_i. These vectors are the control information to be attached to a message sent over the wireless channel.

In addition to the message overhead induced over wired links which is $O(n_h)$, this protocol incurs a significant message overhead over wireless links which is $O(n_h)$. Furthermore, authors don't say any things about how the handoff is handled in their protocol.

3.3.2 Protocols with Environment Assumptions

Anastasi, Bartoli and Spadoni Protocol

The causal multicast algorithm (ABS) presented here [25] uses *coordinators* at a level higher in a hierarchy that the MSSs. This algorithm is part of a larger suite of fault-tolerant message ordering algorithm in mobile systems, and hence the need for the coordinator nodes. The coordinators broadcast the messages to all the MSSs which in turn broadcast them to all MHs. The wired network is assumed to be FIFO and reliable, while the wireless network is assumed to be FIFO but messages can be lost. The algorithm has a message space overhead of $O(n_c)$, where n_c is the number of coordinators in the system model, but the decision of causal ordering delivery takes place at the *MH* level.

Coordinators are not involved in communication to obtain order, indeed they are a system's bottleneck.

Bittner Protocol

In [60], author proposed a multicast protocol (*BIT*) which ensures total order in a mobile environment. It uses sequencers organized in a tree-structure to obtain this total order.

Causal order in this protocol is a side effect from the tree-based overlay network structure used and the assumption of FIFO message exchange between links. The causal order can be obtained with the characteristic that messages in MSSs are handled in order of their arrival, the assumption of FIFO and the fact that there is a unique path in overlay network for messages of one MH to all other MHs. That also holds for messages of different groups in all MHs that are members of both of these groups.

3.4 Discussion

Our goal in this chapter was to give an overview about the causal communication protocols. In order to structure the classification more clearly, we subdivided it into relevant work done in the field of static distributed systems and work relevant in the context of mobile distributed systems. Our presentation of these protocols was mainly focused on the principle used to capture causality between events, the communication and computation costs induced, the type of communication paradigm supported (i.e. unicast, multicast or broadcast), and in some cases, the way that protocols handle failures.

As shown above, several algorithms have been developed to implement causal message ordering in conventional distributed systems. These protocols mainly differ by the size of control information carried by messages. In the following discussion, and relating to the scope of our thesis, we do not consider these protocols. Instead, we limit ourselves to protocols proposed for mobile systems. Here, we revisit and discuss them in the light of the message cost of attaching additional information (i.e. control information) to messages.

Many contributions, following various approaches, are proposed to implement causal message ordering protocols for mobile computing systems. Some of these contributions assume certain network topology. The main disadvantage of these protocols is that they are usable in applications where the topology is relatively static or can be computed at compile or configuration time. Protocols of this type are absolutely inappropriate to applications where it is not possible to know in advance when, where, nor who (user, process, agent, etc.) will participate, (e.g. all the cooperative internet applications and mobile applications). For this reason, most contributions done in the context of mobile environment, do not make any environment assumptions. To reduce computation and communication loads on mobile hosts, most of them store data structures relevant to causal ordering in $MSSs$, and the algorithm is executed by the MSSs on behalf of the

MHs. Doing so, most of these algorithms are based on the timestamping by vector clock with n_s entries, where n_s is the number of $MSSs$ in the system. This yields to different vision between *MSSs* and *MHs* on order of events, which creates 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. To eliminate this inhibition delay, we have proposed a new unicast protocol (*Mobi_Causal*) to implement the causal message ordering in a mobile environment [47]. *Mobi_Causal*, compared with previous proposals, is characterized by the elimination of unnecessary inhibition delay in delivering messages while maintaining low message overhead. The protocol requires minimal resources on mobile hosts and wireless links. *Mobi_Causal* protocol is also scalable and can easily handle dynamic change in the number of participating mobile hosts in the system.

The other emerging trend is the increasing importance of multicast applications for the purpose of multi-party conferencing, and mobile users will expect similar kinds of multicast applications to be supported on their portable devices with wireless capability. As far as multicasting is concerned, the functionality of multicast thus can be achieved only by means of multiple unicasts, which results in poor utilization of the network bandwidth [3]. A multicast protocol with exactly-once delivery semantics for a mobile environment is presented by [3]. However, this protocol does not enforce causal ordering. Prakash *et al.* (PRS) [150] presented an algorithm which combines an improved version of the RST causal order algorithm [156] for static networks with that in [3]. Its message overhead is $O(n_h^2)$ despite the use of direct dependency relation in the construction of control information. The causal multicast algorithm presented in [25] uses coordinators at a higher level in a hierarchy than the *MSSs*. Coordinators broadcast the messages to all the *MSSs* which in turn broadcast them to all *MHs*. The algorithm has a message space overhead of $O(n_c)$, where n_c is the number of coordinators, but the decision of causal ordering delivery takes place at the MH level. The algorithm (LH) in [121] also performs a broadcast among *MSSs*, and then within each cell to *MHs*. Another causal multicast protocol, ($CYTH$), is presented in [71] where only a part of *MSSs* (called *mobility agents*) is involved in group computations. An MH always initiates causal multicast via its *serving agent* (i.e. *MSS* of cell where it has been located for the first time) regardless of its current location. As consequence, *MHs* appear stationary and this simplifies the development of the protocol. However, if we imagine the case where each *MSS* is a serving agent for at least one MH member of the group, then all *MSSs* will be included in the multicast group even if there is no member located in their cells. For both LH and $CYTH$ protocols, the message overhead is $O(n_s)$. In [65], Chandra *et al.* (CK) adapt the KS optimal causal multicast algorithm [**?**] to mobile networks. In spite of defining the necessary and sufficient con-

ditions on the control information, the message overhead of *CK* is, in the worst-case, $O(n_h^2)$. Concerning the protocol *MOCAVI* [80], the drawback is that this protocol involve MHs in the causal ordering of messages, which results in $O(n_h)$ message overhead over wireless links. Furthermore, authors don't say any things about how the handoff is handled in their protocol.

In summary, we can notice that some proposals offer a better message overhead but incur an unnecessary inhibition delay in the delivery of messages, while some other solutions overcome this problem but by incurring high message overhead which can reaches $O(n_h^2)$. Moreover, most of them do not deal with group communication.

From this discussion it is apparent that a good algorithm designed for mobile networks will require minimal amount of communication, storage and computation either over wired or wireless network-parts.

3.5 Conclusion

The main contribution of this chapter is the definition of a classification for causal message ordering protocols, that makes it easier to understand the relationship between them. It also provides a good basis for comparing the algorithms and understanding some tradeoffs. Furthermore, the chapter has presented a vast survey of most of the existing algorithms and discussed their respective characteristics.

From this brief survey, we may conclude that both the message and storage overheads are crucial as far as scalability is concerned and must be treated with the same importance.

Chapter 4

MMobi_Causal: A Causal Multicast Protocol

4.1 Introduction

In this chapter, we tray to get benefit from the advantages of our unicast protocol [47, 43], in order to propose a variant of it allowing multicast communication in mobile groups and which can fill gaps of existing protocols, especially in terms of control information's size appended to each message.

The proposed protocol is optimal with respect to communication overhead because the construction of control information relies on the immediate dependency relationship between messages as well as the multicast of messages is limited to the $MSSs$ where the group members are located.

Here we assume no membership changes such as members joining or leaving a group. These cases will be handled by the membership algorithm described in Chapter 7.

Part of the results presented in this chapter has been published in [45].

4.2 System Model

The considered mobile computing system model is a cellular network. It consists of two kinds of entities: mobile support stations (MSS) and mobile hosts (MH). The MSS nodes are fixed and connected among them via a wired network. An MSS support the MH nodes in communicating among them and with the other $MSSs$. $MSSs$ and MHs communicate via a wireless network. Each MSS defines a connectivity area (*cell*) where MHs can reliably communicate. At any given time, an MH is assumed to be within the cell of at most one MSS, which is called its local MSS.

Our system model assumes that there are no host crash failures. This model is a reasonable starting model for mobile systems, given current hardware and software technology. Since we are exclusively concerned with techniques to enforce causal delivery, we assume that both wired and wireless channels are reliable, in the sense that

messages sent are always received by all correct addressed participants. We also assume that both wireless and wired channels take an arbitrary but finite amount of time to deliver messages and that wireless channels are FIFO. Furthermore, the union of all cells does not cover the entire area where MHs may be located. Whenever an MH moves from one cell to another, a $hand-off$ procedure is performed in which the communication responsibilities of the MH are transferred to the new local MSS.

We assume that a message can be sent to an arbitrary set of *MSSs* in one send event; that is, hardware multicast is used.

In our case, a *group* is a collection of *MHs* that cooperate towards a common goal. We define the view of a group by the set of *MHs* that are currently members of the group, denoted as MH_{View}, and the set of *MSSs* where these members are located, denoted as MSS_{View}. Thus, the group's view reflects not only the set of *MHs* that are currently members of the group but also the set of their *MSSs*. Each group and each process has a unique identifier.

4.3 Data structures

4.3.1 MSS's data structures

Each MSS S_i maintains the following data structures:

- $IdSend_{S_i}$: an integer counter that is incremented each time a message is multicast by S_i. The counter is set to zero at the beginning and will be incremented each time a message is sent by S_i.

- $Send_M$: a set of tuples of the form (id_m, nb_{ack}) where nb_{ack} is the number of intended *ack* for the message having id_m as identity. It's kept information about messages sent by S_i but not yet delivered to all group members.

- $Unst_M$: a set to keep a copy of each sending message till the delivery of this message to all MHs members of the group. We keep these copies in order to ensure the delivery of messages to MHs in movement. Each time a message is deliver to a member (i.e. an MH), its MSS sends back $(ack(id_m))$ to the sending MSS. At the reception of this *ack* by the sending MSS, nb_{ack} is decremented by one. When nb_{ack} becomes null, the sending MSS deletes information about this message from its data structures and sends a $delete(id_m)$ message to other MSSs. At the reception of this message, $MSSs$ delete in their turn information about this message from their data structures.

4.3.2 MH's data structures

For each MH h_i, the following data structures are maintained:

- ϕ: a vector of length n_s. One entry $\phi_m[k] = \{(id, Dest), ...\}$ in this vector corresponds to the set of predecessor messages' identifiers of m sent by the MSS S_k for which the delivery condition is not yet confirmed. We keep the message's identifier and for which MHs (i.e the set $Dest$) this message is sent. As soon as the delivery of a message is confirmed for all $h_i \in Dest$, the corresponding entry is deleted from ϕ. It's not necessary to keep information about messages that their delivery is confirmed, thereby reducing the control information's size.

- $LastRcv_{h_i}$: a set that stores identifiers of messages delivered to h_i. One component of the set is denoted by (S_j, SD_j), where S_j is the identity of the sending MSS and $SD_j = id_1, id_2, id_3, id_4, ...$ is the set of message's identifiers and corresponds to a dependency sequences [152].

- $AtFile_{h_i}$: a queue that stores information about messages not yet delivered to h_i because the delivery condition is not satisfied; i.e. out-of-order messages.

We note that the data structures introduced above concern only the $MSSs$ and MHs belonging to the group and they are all maintained at the $MSSs$ level.

4.4 Static module

4.4.1 Emission phase

The multicast of a message m starting from a group's member MH, $h_i(S_i)$, to a subset of members $Dest$ of the same group g_i (i.e. $Dest \subseteq MH_{ViewG}$) is done by sending the message from h_i to its MSS, S_i, which increments $IdSend_{S_i}$ by one, then piggybacks ϕ_{h_i} and $IdSend_{S_i}$ to the message and multicasts the message to MSSs, S_j, where group members are located (i.e. $S_j \in MSS_{ViewG}$).

Once the message m is sent, we update the global clock ϕ by setting, $\phi_{h_i}[i] := \phi_{h_i}[i] \cup (IdSend_{S_i}, Dest)$. We add also the couple $(IdSend_{S_i}, |Dest|)$ to $Send_M$ where $|Dest|$ represents the number of expected Ack so that the message becomes stable (i.e. delivered to all recipient members).

1 On receipt of (m,$Dest$) **by** S_i **from** h_i /* A multicast request of m from $h_i(S_i)$ to $Dest \subset g_i$ */

1. $IdSend_{S_i} := IdSend_{S_i} + 1$;
2. $Multicast(m, IdSend_{S_i}, \phi_{h_i}, Dest)$ to all $S_j \in g_i$;
3. $\phi_{h_i}[i] := \phi_{h_i}[i] \cup (IdSend_{S_i}, Dest)$;
4. $Update(\phi_{h_i}[i])$;
5. $Send_{M_{S_i}} := Send_{M_{S_i}} \cup (IdSend_{S_i}, |Dest|)$;
6. $Unst_{M_{S_i}} := Unst_{M_{S_i}} \cup (m, IdSend_{S_i}, \phi_{h_i}, Dest)$

2 Procedure Update(S) /*Eliminate the redundant couples by keeping the recent one*/

1. **for all** $h_j \in Dest$ **do**
2. **if** $\exists(Id_1, Dest_1)$, $(Id_2, Dest_2) \in S$ such that $h_j \in Dest_1$ and $h_j \in Dest_2$ and $Id_1 < Id_2$ **then**
3. $Dest_1 := Dest_1 - \{h_j\}$
4. **end if**
5. **end for**

4.4.2 Reception phase

Upon the arrival of m to an MSS S_j belonging to the group g_i, this message will be delivered to a receiving MH h_j located in the cell covered by this MSS only if the delivery condition holds; i.e. this message has not yet been delivered to h_j (to ensure *exactly once delivery* property) and all messages which causally precede the message m are received by S_j and delivered to h_j:

If m is already delivered to h_j (i.e. $id_m \in LastRcv_{h_j}$), then this message is ignored. Otherwise:

1. If $\forall k = \overline{1,n}$, $\nexists(\_, Dest) \in \phi_m[k]$ such that $h_j \in Dest$, then this message does not depend on any other messages and its delivery is immediate.

2. If $\forall k = \overline{1,n}, \exists(idf, Dest) \in \phi_m[k]$ such that $h_j \in Dest$, then we simply verify that the message, having idf as an identifier, has been delivered to h_j. The delivery of a message to an MH is expressed by the existence of an entry (S_k, SD_k) in the set $LastRcv_{h_j}$ such as $idf \in SD_k$. If so, the message m is delivered as well as all messages in the queue $AtFile_{h_j}$ that are awaiting m. The delivery of these messages led to their removal from the queue and the emission of $Ack(IdSend_m)$ to the sending MSS.

In the case where no one of the above two conditions is satisfied, then we say that the delivery condition is not satisfied, and consequently the message m is queued in $AtFile_{h_j}$.

3 On Receipt of $(m,IdSend_m,\phi_m,Dest_m)$ **by** S_j **from** S_i

1. $Unst_{M_{S_j}} := Unst_{M_{S_j}} \cup (m, IdSend_m, \phi_m, Dest_m)$
2. **for all** $h_j \in Dest_m$ located in S_j's cell **do**
3. **if** $(IdSend_m \notin LastRcv_{h_j}.SD_i)and(\forall k = \overline{1,n}, \exists(idf, Dest) \in \phi_m[k]$ and $h_j \in Dest$ such that $idf \in LastRcv_{h_j}.SD_k)$ **then**
4. Deliver m to h_j;
5. **else**
6. **if** $(IdSend_m \notin LastRcv_{h_j}.SD_i)$ **then**
7. $AtFile_{h_j} := AtFile_{h_j} \cup (m, IdSend_m, \phi_m, Dest_m)$
8. **end if**
9. **end if**
10. **end for**

4 On receipt of $(m,(IdSend_m, S_i))$ **by** h_j **from** S_j

1. Send$(Ack(IdSend_m, S_i))$ to S_j;

5 On receipt of $(Ack(IdSend_m, S_i))$ **by** S_j **from** h_j

1. Send$(Ack(IdSend_m))$ to S_i;
2. $\phi_{h_j} := \phi_m$;
3. $LastRcv_{h_j}.SD_i := LastRcv_{h_j}.SD_i \cup (IdSend_m, IdSend_m)$; {Manage the dependency sequences as in [152]}
4. $Build(\phi_{h_j})$; {garbage collection on ϕ_{h_j}}
5. **for all** $m' \in AtFile_{h_j}$ **do**
6. **if** $(\forall k = \overline{1,n}, \exists(idf, Dest) \in \phi_{m'}[k]$ and $h_j \in Dest$ such that $idf \in LastRcv_{h_j}.SD_k)$ **then**
7. Deliver(m') to h_j
8. $AtFile_{h_j} := AtFile_{h_j} - (m', IdSend_{m'}, \phi_{m'}, Dest_{m'})$
9. **end if**
10. **end for**

6 On receipt of $(Ack(IdSend_m))$ **by** S_i **from** S_j

1. Let $(IdSend_m, nb_{ack}) \in Send_{M_{S_i}}$
2. $nb_{ack} := nb_{ack} - 1$;
3. **if** $nb_{ack} = 0$ **then**
4. $Send_{M_{S_i}} := Send_{M_{S_i}} - (IdSend_m, nb_{ack})$
5. $Unst_{M_{S_i}} := Unst_{M_{S_i}} - (m, IdSend_m, \phi_m, Dest_m)$
6. **for all** h_l in S_i's cell **do**
7. **if** $\exists(IdSend_m, Dest) \in \phi_{h_l}[i]$ **then**
8. $\phi_{h_l}[i] := \phi_{h_l}[i] - (IdSend_m, Dest)$
9. **end if**
10. **end for**
11. Multicast$(Delete(IdSend_m, S_i))$ to all $S_j \in g_i$
12. **end if**

7 On receipt of $(Delete(IdSend_m, S_i))$ **by** S_j **from** S_i

1. $Unst_{MS_j} := Unst_{MS_j} - (m, IdSend_m, \phi_m, Dest_m)$
2. **for all** h_l in S_j's cell **do**
3. **if** $\exists (IdSend_m, Dest) \in \phi_{h_l}[i]$ **then**
4. $\phi_{h_l}[i] := \phi_{h_l}[i] - (IdSend_m, Dest)$
5. **end if**
6. **end for**

8 Procedure Build(ϕ_{h_j}) /*due to the delivery of m sent by h_i. Delete the useless information, i.e. information about delivered messages*/

1. $\phi_{h_j}[i] := \phi_{h_j}[i] \cup (IdSend_m, Dest_m - \{h_j\})$
2. Update($\phi_{h_j}[i]$)
3. **for all** h_l in S_j's cell **do**
4. **if** $\exists (IdSend_m, Dest) \in \phi_{h_l}[i]$ and $h_j \in Dest$ **then**
5. Delete(h_j,$Dest$)
6. **end if**
7. **if** $\forall k = \overline{1,n}, \exists (idf, Dest) \in \phi_{h_j}[k]$ and $h_l \in Dest$, such that $idf \in LastRcv_{h_l}.SD_k$ **then**
8. Delete(h_l,$Dest$)
9. **end if**
10. **end for**

4.4.3 Example

It is assumed that the view of the group g_i is composed of (see Figure 4.1): $MSS_{View_{g_i}} = \{S_1, S_2, S_3\}$ and $MH_{View_{g_i}} = \{h_1, h_2, h_3, h_4\}$.

It is also assumed that h_1 is located in the cell covered by S_1, that h_2 is located in the cell covered by S_2 and that h_3 and h_4 are located in the cell covered by S_3.

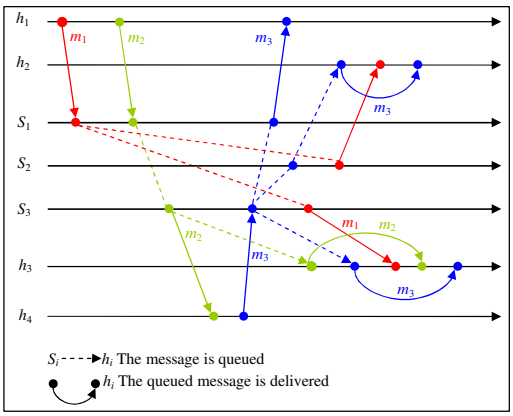

Figure 4.1: *MMobi_Causal*: An illustrated example

- At the emission of m_1 from h_1 to $\{h_2, h_3\}$
  $IdSend_{S_1} = 1$

$$Multicast(m_1, 1, \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \{h_2, h_3\})$$
$$\phi_{h_1} = \begin{bmatrix} (1, \{h_2, h_3\}) \\ 0 \\ 0 \end{bmatrix}, \; Send_{M_{S_1}} = \{(1, 2)\}$$

- At the emission of m_2 from h_1 to $\{h_3, h_4\}$
  $IdSend_{S_1} = 2$
  $$Multicast(m_2, 2, \begin{bmatrix} (1, \{h_2, h_3\}) \\ 0 \\ 0 \end{bmatrix}, \{h_3, h_4\})$$
  $$\phi_{h_1} = \begin{bmatrix} (1, \{h_2, h_3\})(2, \{h_3, h_4\}) \\ 0 \\ 0 \end{bmatrix} \xrightarrow[Update]{} \phi_{h_1} = \begin{bmatrix} (1, \{h_2\})(2, \{h_3, h_4\}) \\ 0 \\ 0 \end{bmatrix},$$
  $Send_{M_{S_1}} = \{(1, 2), (2, 2)\}$

- At the reception of $(m_2, 2, \begin{bmatrix} (1, \{h_2, h_3\}) \\ 0 \\ 0 \end{bmatrix}, \{h_3, h_4\})$ by (S_3)
  Checking ϕ_{m_2}, we find that the delivery of the message m_2 to the host h_3 depends on the delivery of the first message (i.e. m_1) sent by S_1 to h_3. But since the list $LastRcv_{h_3} = \{\}$ does not contain information about its delivery, which means that m_1 has not yet been delivered to h_3. Thus, the message m_2 will be waiting in the queue $AtFile_{h_3}$ until the delivery of m_1.

  Checking ϕ_{m_2}, we find that the delivery of the message m_2 to the host h_4 does not depend on the delivery of any other messages. Consequently, the message m_2 will be delivered to the host h_4, and the following updates will be done:
  $$\phi_{h_4} = \begin{bmatrix} (1, \{h_2, h_3\}) \\ 0 \\ 0 \end{bmatrix} \xrightarrow[Build]{} \phi_{h_4} = \begin{bmatrix} (1, \{h_2\})(2, \{h_3\}) \\ 0 \\ 0 \end{bmatrix},$$
  $LastRcv_{h_4} = \{(S_1, \{2, 2\})\}$

- At the emission of m_3 from h_4 to $\{h_1, h_2, h_3\}$
  $IdSend_{S_3} = 1$
  $$Multicast(m_3, 1, \begin{bmatrix} (1, \{h_2\})(2, \{h_3\}) \\ 0 \\ 0 \end{bmatrix}, \{h_1, h_2, h_3\})$$
  $$\phi_{h_4} = \begin{bmatrix} (1, \{h_2\})(2, \{h_3\}) \\ 0 \\ (1, \{h_1, h_2, h_3\}) \end{bmatrix}, \; Send_{M_{S_3}} = \{(1, 3)\}$$

- At the reception of $(m_3, 1, \begin{bmatrix} (1, \{h_2\})(2, \{h_3\}) \\ 0 \\ 0 \end{bmatrix}, \{h_1, h_2, h_3\})$ by (S_1)
  Checking ϕ_{m_3}, we find that the delivery of the message m_3 to the host h_1 does

not depend on the delivery of any other messages. Hence, the message m_3 will be delivered to the host h_1, and the following updates will be done:

$$\phi_{h_1} = \begin{bmatrix} (1,\{h_2\})(2,\{h_3\}) \\ 0 \\ (1,\{h_2,h_3\}) \end{bmatrix},\ LastRcv_{h_1} = \{(S_3,\{1,1\})\}$$

- At the reception of $(m_3, 1, \begin{bmatrix} (1,\{h_2\})(2,\{h_3\}) \\ 0 \\ 0 \end{bmatrix}, \{h_1,h_2,h_3\})$ by (S_2)

  Checking ϕ_{m_3}, we find that the delivery of the message m_3 to the host h_2 depends on the delivery of the first message (i.e. m_1) sent by S_1 to h_2. But since the list $LastRcv_{h_2} = \{\}$ does not contain information about its delivery which means that m_1 has not yet been delivered to h_2. Thus, the message m_3 will be waiting in the queue $AtFile_{h_2}$ until the delivery of m_1.

- At the reception of $(m_3, 1, \begin{bmatrix} (1,\{h_2\})(2,\{h_3\}) \\ 0 \\ 0 \end{bmatrix}, \{h_1,h_2,h_3\})$ by (S_3)

  Checking ϕ_{m_3}, we find that the delivery of the message m_3 to the host h_3 depends on the delivery of the second message (i.e. m_2) sent by S_1 to h_3. But since the list $LastRcv_{h_3} = \{\}$ is empty which means that m_2 has not yet been delivered to h_3. Thus, the message m_3 will be waiting in the queue $AtFile_{h_3}$ until the delivery of m_2.

- At the reception of $(m_1, 1, \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \{h_2,h_3\})$ by (S_2)

  Checking ϕ_{m_1}, we find that the delivery of the message m_1 to the host h_2 does not depend on the delivery of any other messages. Consequently, the message m_1 will be delivered to the host h_2, and the following updates will be done:

  $$\phi_{h_2} = \begin{bmatrix} (1,\{h_3\}) \\ 0 \\ 0 \end{bmatrix},\ LastRcv_{h_2} = \{(S_1,\{1,1\})\}$$

  Now, the message m_3 can be delivered to h_2. Consequently, the entry of this message will be deleted from the queue $AtFile_{h_2}$ and the following updates will be done:

  $$\phi_{h_2} = \begin{bmatrix} (2,\{h_3\}) \\ 0 \\ (1,\{h_1,h_3\}) \end{bmatrix},\ LastRcv_{h_2} = \{(S_1,\{1,1\})(S_3,\{1,1\})\}$$

- At the reception of $(m_1, 1, \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \{h_2,h_3\})$ by (S_3)

  Checking ϕ_{m_1}, we find that the delivery of the message m_1 to the host h_3 does not depend on the delivery of any other messages. Consequently, the message m_1 will be delivered to the host h_3, and the following updates will be done:

$$\phi_{h_3} = \begin{bmatrix} (1,\{h_2\}) \\ 0 \\ 0 \end{bmatrix}, LastRcv_{h_3} = \{(S_1,\{1,1\})\}$$

Now, the delivery of the message m_2 to h_3 can be done. Hence, the entry of this message will be deleted from the queue $AtFile_{h_3}$ and the following updates will be done:

$$\phi_{h_3} = \begin{bmatrix} (1,\{h_2\}) \\ 0 \\ 0 \end{bmatrix}, \phi_{h_4} = \begin{bmatrix} (1,\{h_2\}) \\ 0 \\ (1,\{h_1,h_2,h_3\}) \end{bmatrix}, LastRcv_{h_3} = \{(S_1,\{1,2\})\}$$

So, we can now deliver the message m_3 to h_3. Hence, the entry for this message will be deleted from the queue $AtFile_{h_3}$ and the following updates will be done:

$$\phi_{h_3} = \begin{bmatrix} (1,\{h_2\}) \\ 0 \\ (1,\{h_1,h_2\}) \end{bmatrix}, \phi_{h_4} = \begin{bmatrix} (1,\{h_2\}) \\ 0 \\ (1,\{h_1,h_2\}) \end{bmatrix}, LastRcv_{h_3} = \{(S_1,\{1,2\})(S_3,\{1,1\})\}$$

Delivering a message m, having id_m as identifier, to all its recipients, involves the send of an acknowledgement from their $MSSs$ to the sending MSS. When all acknowledgements, about the delivery of m, are received by the sending MSS S_i, the corresponding entries are removed from $Send_{MS_i}$, $Unst_{MS_i}$ and ϕ_{h_i} for all h_i located in the cell covered by S_i. After that, the sending MSS send the deletion request to all MSSs S_j belonging to the group in order that they delete information corresponding to this message from their data structures. So, the corresponding entries will be removed from $Unst_{MS_j}$ and ϕ_{h_j} for all h_j located in the cell covered by S_j. This permits to reduce considerably the message overhead.

4.5 Handoff module

The handoff module presented here is similar to that presented in [47, 43] except that we add the part concerning the update of the group view (especially the set MSS_{view}) caused by the migration of the MH.

The handoff for an MH h_i is handled by passing ϕ_{h_i}, $LastRcv_{h_i}$ and $AtFile_{h_i}$ from the old MSS to the new MSS. Upon the reception of this information, the new MSS verifies if there exist messages in $Unst_M$ destined to h_i but not yet received by it; i.e. for each message $m \in Unst_M | h_i \in Dest_m$ if $id_m \notin LastRcv_{h_i}$ then this message is delivered to h_i provided that its delivery condition is verified, otherwise the message is added to the queue $AtFile_{h_i}$.

The handoff procedure ends when S_j receives all data transmitted by S_i. The ending of handoff procedure will be expressed by the emission of message $handoff\_over(h_i)$ by S_i to S_j, and ever since S_j takes care of maintaining the causal ordering of messages destined to h_i.

If h_i migrates to another MSS, S_k, before the current handoff procedure is completed, the new $handoff - begin$ message issued by S_k will not be handled until the

current handoff procedure is completed.

If the migrated MH h_i resident in the cell covered by the MSS S_i is the last group member in this cell then S_i is deleted from the set MSS_{view}. Moreover, if the migrated MH h_i is the first group member to be located in the cell covered by the MSS S_j then S_j is added to the set MSS_{view}.

9 On the receipt of $(Register(S_i))$ **by** S_j **from** h_i

1. $Send(Handoff - begin(h_i))$ to S_i

10 On the receipt of $(Handoff - begin(h_i))$ **by** S_i **from** S_j

1. $Send(Data(\phi_{h_i}, LastRcv_{h_i}, AtFile_{h_i}))$ to S_j

11 On the receipt of $(Data(\phi_{h_i}, LastRcv_{h_i}, AtFile_{h_i}))$ **by** S_j **from** S_i

1. $Save(\phi_{h_i}, LastRcv_{h_i}, AtFile_{h_i})$
2. $Send(Handoff - over(h_i))$ to S_i
3. **for all** $((m, IdSend_m, \phi_m, Dest_m) \in Unst_{M_{S_j}} \mid h_i \in Dest_m$ and $IdSend_m \notin LastRcv_{h_i}.SD_l)$ **do**
4. **if** $(\forall k = \overline{1,n}, \exists(idf, Dest) \in \phi_m[k]$ and $h_i \in Dest$ such that $idf \in LastRcv_{h_j}.SD_k)$ **then**
5. $Deliver(m)$ to h_i;
6. **for all** $m' \in AtFile_{h_j}$ **do**
7. **if** $(\forall k = \overline{1,n}, \exists(idf, Dest) \in \phi_m[k]$ **and** $h_j \in Dest$ **such that** $idf \in LastRcv_{h_j}.SD_k)$ **then**
8. **Deliver**(m') **to** h_j
9. $AtFile_{h_j} := AtFile_{h_j} - (m', IdSend_{m'}, \phi_{m'}, Dest_{m'})$
10. **end if**
11. **end for**
12. **else**
13. $AtFile_{h_i} := AtFile_{h_i} \cup (m, IdSend_m, \phi_m, Dest_m)$
14. **end if**
15. **end for**

12 On the receipt of $(Handoff - over(h_i))$ **by** S_i **from** S_j

1. $Delete(\phi_{h_i}, LastRcv_{h_i}, AtFile_{h_i})$

4.6 Correctness Proof

4.6.1 Safety property proof

Let two messages m_1 and m_2, and let Id_{m_1} and Id_{m_2}, the identities of emission events of m_1 and m_2, respectively.

We suppose that $Send(m_1) \rightarrow Send(m_2)$ and $\nexists m \mid Send(m_1) \rightarrow Send(m) \rightarrow Send(m_2)$; which means that the emission of m_1 precedes immediately the emission of m_2. Two cases are to be considered:

Case *m_1 and m_2 are sent by the same MH $h_l(S_l)$:*

Since the emission of m_1 precedes immediately that of m_2, then the emission of m_1 is certainly the last event occurred in $h_l(S_l)$ before the emission of m_2. So, $(id_{m_1}, h_j) \in \phi_{m_2}$. We suppose that the receiving MSS S_j, where h_j is located, delivers m_2 before m_1. This means that the delivery condition for m_2, represented by $\exists(S_l, SD_l) \in LastRcv_{h_j}$ where $id_{m_1} \in SD_l$, is held. This condition can be verified only if S_j has delivered m_1 to h_j before the delivery of m_2. Which leads to a contradiction with the assumption that m_2 has been delivered before m_1.

Case *m_1 and m_2 are sent by two distinct MHs $h_k(S_k)$ and $h_l(S_l)$, respectively:*

We suppose that m_2 has been delivered before m_1 to the receiving MH h_j while we tray to prove that this is impossible.

Since the emission of m_1 precedes immediately that of m_2 compared to h_j, then $h_l(S_l)$ has certainly received a message m so that $(Id_{m_1}, h_j) \in \phi_m$ before the emission of m_2. So, $(h_j, id_{m_1}) \in \phi_{m_2}$. The delivery of m_2 by $S_j(h_j)$ is constrained by the condition: $\exists(S_l, SD_l) \in LastRcv_{h_j}$ where $id_{m_1} \in SD_l$ and this can be verified only if S_j has delivered m_1 before m_2. This leads also to a contradiction.

4.6.2 Liveness property proof

The delivery of a message m to a destination MH h_j can be delayed only by other messages destined to h_j that are not yet delivered to it. In order to proof that this waiting delay is finite, we proceed as follow: we consider all messages which are not yet delivered to $h_j(S_j)$. The happened before relationship can be used to define a partial order on the emission events of these undelivered messages. Let m' be one of these messages in the partial order whose the emission does not have a predecessor. Hence m' has not been delivered to h_j on being received, then the following condition must be true: $\exists k = \overline{1,n}, \exists(idf, Dest) \in \phi_{m'}[k]$ *and* $h_j \in Dest$ and this can be valid only if m' is causally related to another message (having *idf* as identifier) sent before it. This violates the assumption that among the undelivered messages to $h_j(S_j)$, m' does not have a predecessor. Thus, our protocol never delays a message indefinitely.

4.6.3 Exactly once delivery property proof

Our protocol ensures that exactly once copy is delivered for each multicast message. This can be inferred from the following observations:

The delivery of a message m to an MH h_i is expressed by the addition of its identity to the set $LastRcv_{h_i}$. Thus, the scan of this set either at the reception or handoff phase

prevents the delivery of another copy of the same message. Therefore, *at most-once* delivery of a message is ensured.

A multicast message m is buffered at every MSS till an explicit $Delete(id_m)$ message is received. This $Delete()$ message is sent by the sending MSS only after an acknowledgement from each MH in $Dest$ has been received. Which ensures *at least-once* delivery of a message.

Exactly-once delivery is thus ensured by at least-once and at most-once delivery of a message.

4.7 Complexity Analysis

The communication overhead of our protocol is $O(\sum_{i=1}^{n_s} l_i)$, where l_i is the number of tuples in the entry $\phi[i]$. l_i represents the number of messages sent by S_i and for which the delivery is not yet confirmed. In the worst case, this number can be equal to $|MH_{View}|$, where $|MH_{View}|$ is the number of MHs in the group. But since the delivery of each message involves the update of this clock then we can assume that this number remains low and $\sum_{i=1}^{n_s} l_i \ll n_s \times |MH_{View}| \ll n_s \times n_h$. Comparing to the related works, our protocol outperforms the CK and PRS protocols with respect to message overhead. $CYTH$ [71] and LH [121] protocols provide the best message overhead but these protocols create an unnecessary inhibition delay in the delivery of messages. In our protocol, each message sent by an MH is immediately delivered if the delivery condition is verified. The control information piggybacked to each message reflects the *effective causality* relation between messages, as perceived by their MHs. So, the delivery of message is never inhibited as in these protocols.

A message is multicast only to $MSSs$ containing the group members which permits to reduce the number of messages sent over wired channels even if the hardware multicast is not supported. So, the number of destinations of a given message is $O(|MSS_{View}|)$.

In our protocol, a $MSS \in MSS_{View}$ maintains a list $LastRcv$ to keep information about delivered messages. Since the construction of this list relies on the dependency sequences mechanism, then this can result in the non bounded evolution of $LastRcv$. We compensate for this problem by removing the identifier of a message becoming stable; i.e. a message delivered to all group's members. We will show in chapter 7 how we can offset this problem even more.

The defined data structures are independent from the number of MHs in the system, which renders our protocol scalable relatively to the number of MHs and more suitable for large groups. The execution of protocol is completely supported by the MSSs when the role of MHs is restricted to the emission and reception of messages. The messages transmitted over wireless links don't transport any control information which contribute to minimize the use of wireless bandwidth.

Reliable delivery of a multicast message to MHs requires that each recipient acknowledge receipt of the message. However, sending each ack separately to the MSS initiator of the multicast, would lead to an ack implosion problem. This can be tackled in our multicast schemes by having the local MSS collect acks from all recipients within its cell, before forwarding a list of such acks to the initiator.

4.8 Conclusion

Group communication systems need the support of causal multicast communication. In our proposal, we have tried to get benefit of the important characteristics of our unicast protocol $Mobi\_Causal$ [47], such as elimination of unnecessary inhibition delay, low message overhead and scalability, in order to adapt it to achieve causal requirement for a multicast communication among a static group of MHs.

The proposed protocol retains all these interesting characteristics. It provides an optimal communication overhead without causing inhibition effect in the delivery of messages. This optimality is reached using the knowledge of immediate dependency relationships between messages and transmitting each message with only the relevant information to compliance with the causal order in the delivery of messages. The protocol is proved to satisfy the safety, the liveness, and the exactly once delivery properties.

Chapter 5

BMobi_Causal: A Causal Broadcast Protocol

5.1 Introduction

Group communication is an abstraction which deals with multicasting a message from a source process to more than one interested recipients within the group. Group communication-based applications often require that the exchanged messages within the group's members to be addressed to all processes in the group and not only to a subset of group's members. We refer to this type of communication as *broadcast communication*. Several solutions [25, 121, 65, 71, 144, 33, 80, 45] have been proposed to ensure the causal consistency for a broadcast communication in mobile systems. Most of these solutions [25, 121, 65, 71, 80, 45] deal with the broadcast as a particular case of multicast where the message is intended for all processes.This should make the definition of data structures not optimal for applications using only the broadcast communication. In the same way, these solutions broadcast messages to all *MSSs* even those do not containing any recipient process and most of them does not consider the group communication.

Therein, we expect that developing a protocol dedicated to causal broadcast is more interesting than using a multicast protocol to ensure this broadcast, especially in terms of *control information*'s size appended to each message.

In this chapter, we focus on achieving causal requirement for a broadcast communication in mobile groups. The proposed protocol outperforms its counterparts with respect to communication overhead because the construction of Control Information relies on the *immediate dependency relationship* between messages as well as the diffusion of messages is limited to the $MSSs$ where the group members are located.

The system model we consider in this chapter is identical to the model described in Chapter 4 with the additional constrained that a sending message is always destined to all group's members. Throughout this chapter, as in Chapter 4, we assume no membership changes such as members joining or leaving a group (i.e. we consider a

static group). These cases will be handled by the membership algorithm described in chapter 7.

The rest of this chapter is organized to give first an overview of our proposal while discussing the motivation behind how we have proceeded to construct the control information (Section 5.2). In the following sections (Section 5.3, 5.4, 5.5), we present the detail of our causal broadcast protocol. Section 5.6 presents the correctness proof of the protocol. Finally, in Section 5.7, we analyze the complexity of our protocol and compare it to more closely related works.

Part of the results presented in this chapter have been published in [44, 46].

5.2 Motivation and Protocol Overview

We present an efficient causal broadcast protocol, ($BMobi\_Causal$), that aims to ensure a causal broadcast delivery in a mobile group while minimizing the *control information* transmitted. In order to reduce the amount of *control information*, our broadcast causal protocol is based on the *immediate dependency relationship*.

The originality of our proposal lies on showing how the rather well known behavior of a process in a distributed application influences the control information's size. In fact, the original motivation behind the use of *immediate dependency relationship* in the construction of control information stems from our observation of real world scenarios:

In real world, events known to be causally related are often attracted together across time and space. So, they are often closer in time and in space then independent ones [83]. Furthermore, observing the real world, we can show that a given behavior or phenomenon is often the result of an *instantaneous reaction* to the last occurring event. So we might say: the reaction is the direct effect of the last occurring event. Thus, finding the cause of a given behavior or phenomenon is possible by only *keeping knowledge about the last occurring event*.

At the time of this writing, we are not aware of any work that takes this approach into account when constructing the control information.

Developing applications induced human-being interaction or simulation of real-world applications should exhibit the similar behavior as in the real-world environment. By viewing the behavior of the execution in the sense that the reaction to occurring events is instantaneous, tracking the dependency between messages is possible by only keeping information about the message's *immediate predecessor*.

Definition 5.1. *A message m' is an **immediate predecessor** of a message m, iff $m' \rightarrow m$ according to the happened before relationship and there is no other message m'' such as $m' \rightarrow m'' \rightarrow m$.*

Then, the send of a message by an MH is the effect of the last message sent by or delivered to this MH. The message m' is either the last message sent by the same

MH (Figure 5.1.a), or the last message delivered to this MH (Figure 5.1.b), before the emission of m.

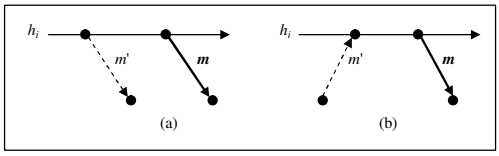

Figure 5.1: A message's immediate predecessor.

Therefore, the only causal information appended to each message m, by the sending MSS, corresponds to the identity of its immediate predecessor. Consequently, our protocol, by using the *Immediate Dependency Relationship* principle, is optimal in the sense that it transmits the *minimal* and *necessary* amount of *Control Information* to completely preserve the causal order.

The receiving MSS uses the dependency sequences mechanism [152] to keep the history of messages delivered to its MHs. At the reception of a message, the receiving MSS uses this history to determine, directly and without incurring additional computation, whether the message can be delivered to a given group member located in its cell or if its delivery should be delayed until its immediate predecessor meant for the member is delivered.

5.3 Data structures

5.3.1 MSS's Data Structures

Each MSS S_i maintains the following data structures:

- $IdSend_{S_i}$: an integer counter that is set to zero at the beginning and is incremented each time a message is broadcast by S_i.
- $Send_M$: a set of tuples of the form (id_m, nb_{ack}) where nb_{ack} is the number of intended *ack* for the message having id_m as identity. It's kept information about messages sent by S_i but not yet delivered to all group members.
- $Unst_M$: a set to keep a copy of each broadcast message till its delivery to all MHs members of the group. We keep these copies in order to ensure the delivery of messages to MHs in movement. Each time a message is deliver to a member (i.e. an MH), its MSS sends back ($ack(id_m)$) to the sending MSS. At the reception of this *ack* by the sending MSS, nb_{ack} is decremented by one. When nb_{ack} becomes null, the sending MSS deletes information about this message from its data structures and sends a $delete(id_m)$ message to other $MSSs$. At the reception of this message, $MSSs$ delete in their turn information about this message from their data structures.

5.3.2 MH's Data Structures

For each MH h_i, the following data structures are maintained:

- id_{predh_i}: a tuple that keep information about the message's immediate predecessor. The tuple is denoted by (id, S), where id is the identifier of the immediate predecessor and S_i is the identity of its sending MSS. This information is updated each time a message is sent by or delivered to this MH.
- $LastRcv_{h_i}$: a set that stores identifiers of messages delivered to h_i. ne component of the set is denoted by (S_j, SD_j), where S_j is the identity of the sending MSS and $SD_j = \{id_1, id_2, id_3, id_4, ...\}$ is the set of message identifiers and corresponds to a dependency sequences. Since all messages sent by an MSS are diffused to all group members, this allows to reduce the size of dependency sequences, compared with unicast and multicast communication, to one interval $\{id_{min}, id_{max}\}$ for each MSS where id_{min} and id_{max} denoted respectively the identity of the first and the last messages sent from this MSS and delivered to the MH.
- $AtFile_{h_i}$: a queue that stores information about messages not yet delivered to h_i because the delivery condition is not satisfied; i.e. out-of-order messages.

We note that the data structures introduced above concern only the $MSSs$ and MHs belonging to the group and they are all maintained at the $MSSs$ level.

5.4 Static module

5.4.1 Emission phase

The broadcast of a message m starting from an MH, h_i, of the MSS, S_i, to members h_j of the group G (i.e. $h_j \in MH_{ViewG}$) is done by sending the message from h_i to its MSS, S_i, which increments $IdSend_{S_i}$ by one, then piggybacks id_{predh_i} and $IdSend_{S_i}$ to the message and broadcasts the message to MSSs, S_j, where group members are located (i.e. $S_j \in MSS_{ViewG}$).

Once the message m is sent, it will become the immediate causal predecessor of any future message to be sent. This is done by setting id_{predh_i} to $(IdSend_{S_i}, S_i)$. We add also the couple $(T(G), IdSend_{S_i})$ to $Send_M$ where $T(G)$ represents the number of expected Ack and is equal to the number of MHs members of the group G.

13 On receipt of (m) **by** S_i **from** h_i

1. $IdSend_{S_i} := IdSend_{S_i} + 1$;
2. $Boradcast(m, IdSend_{S_i}, id_{predh_i})$ to all $S_j \in g_i$;
3. $id_{predh_i} := (IdSend_{S_i}, S_i)$;
4. $Send_{M_{S_i}} := Send_{M_{S_i}} \cup (IdSend_{S_i}, |MH_{view}|)$;
5. $Unst_{M_{S_i}} := Unst_{M_{S_i}} \cup (m, IdSend_{S_i}, id_{predh_i})$

5.4.2 Reception phase

When receiving a message m by an MSS belonging to the group G, this message will be delivered to a receiving MH h_j located in the cell covered by this MSS only if the delivery condition is verified; that means this message has not yet been delivered to h_j (to ensure *exactly once delivery* property) and all messages which causally precede the message m are received by S_j and delivered to h_j:

If m is already delivered to h_j (i.e. $id_m \in LastRcv_h$), then this message is ignored. Otherwise:

1. If $id_{pred_m} = (0, 0)$, then this message is independent from any other message and its delivery is immediate. In the example depicted in Figure 5.2, this is the case of the message m_1.

2. If $id_{pred_m} = (id_{pred}, S_{pred})$, then it's only necessary to verify that the message having id_{pred} as identity is received by h_j. The reception of a message by an MH is expressed by the existence of an entry (S_{pred}, SD_{pred}) in the set $LastRcv_{h_j}$ such as $id_{pred} \in SD_{pred}$. If this holds, the message m is delivered as well as all messages in the queue $AtFile_{h_j}$ which are waiting for the arrival of m. The delivery of these messages involves their deletion from the queue and the emission of $Ack(IdSend_m)$ to the sending MSS.

In the case, where no one of the two conditions is satisfied, we say that the delivery condition is not verified, and consequently the message m is queued in $AtFile_{h_j}$. As shown in the Figure 5.2, the delivery of the message $(m_2, 2, (1, S_1))$ is constrained by the delivery of the first message sent by S_1 (i.e. m_1). Hence $LastRcv_{h_3} = \{\}$ does not contain information about the arrival of m_1, then the delivery of m_2 to h_3 will be delayed and m_2 will be queued in $AtFile_{h_3}$ until the delivery of m_1.

14 On Receipt of (m,$IdSend_m$,id_{pred_m}) **by** S_j **from** S_i

1. $Unst_{M_{S_j}} := Unst_{M_{S_j}} \cup (m, IdSend_m, id_{pred_m})$
2. **for all** $h_j \in MH_{view}$ located in S_j's cell **do**
3. **if** $(IdSend_m \notin LastRcv_{h_j}.SD_i)and((id_{pred_m} = (0, 0))$ or $(id_{pred_m} \neq (0, 0) = (id_{pred}, S_{pred})$ and $\exists(S_{pred}, SD_{pred}) \in LastRcv_{h_j} | id_{pred} \in SD_{pred}))$ **then**
4. Deliver (m) to h_j;
5. **else**
6. **if** $(IdSend_m \notin LastRcv_{h_j}.SD_i)$ **then**
7. $AtFile_{h_j} := AtFile_{h_j} \cup (m, IdSend_m, id_{pred_m})$
8. **end if**
9. **end if**
10. **end for**

15 On receipt of (m,$(IdSend_m, S_i)$) **by** h_j **from** S_j

1. Send($Ack(IdSend_m, S_i)$) to S_j;

16 On receipt of $(Ack(IdSend_m, S_i))$ **by** S_j **from** h_j

1. Send$(Ack(IdSend_m))$ to S_i;
2. $id_{predh_j} := (IdSend_m, S_i)$;
3. $LastRcv_{h_j}.SD_i := LastRcv_{h_j}.SD_i \cup (IdSend_m, IdSend_m)$; {Manage the dependency sequences as in [152]}
4. **for all** $m' \in AtFile_{h_j}$ **do**
5. **if** $(id_{pred_{m'}} = (IdSend_m, S_i))$ **then**
6. Deliver(m') to h_j
7. $AtFile_{h_j} := AtFile_{h_j} - (m', IdSend_{m'}, id_{pred_{m'}})$
8. **end if**
9. **end for**

17 On receipt of $(Ack(IdSend_m))$ **by** S_i **from** S_j

1. Let $(IdSend_m, nb_{ack}) \in Send_{M_{S_i}}$
2. $nb_{ack} := nb_{ack} - 1$;
3. **if** $nb_{ack} = 0$ **then**
4. $Send_{M_{S_i}} := Send_{M_{S_i}} - (IdSend_m, nb_{ack})$
5. $Unst_{M_{S_i}} := Unst_{M_{S_i}} - (m, IdSend_m, id_{pred_m})$
6. **for all** h_l in S_i's cell **do**
7. **if** $id_{predh_l} = (IdSend_m, S_i)$ **then**
8. $id_{predh_l} := (0, 0)$
9. **end if**
10. **end for**
11. Broadcast$(Delete(IdSend_m, S_i))$ to all $S_j \in g_i$
12. **end if**

18 On receipt of $(Delete(IdSend_m, S_i))$ **by** S_j **from** S_i

1. $Unst_{M_{S_j}} := Unst_{M_{S_j}} - (m, IdSend_m, id_{pred_m})$
2. **for all** h_l in S_j's cell **do**
3. **if** $id_{predh_l} = (IdSend_m, S_i)$ **then**
4. $id_{predh_l} := (0, 0)$
5. **end if**
6. **end for**

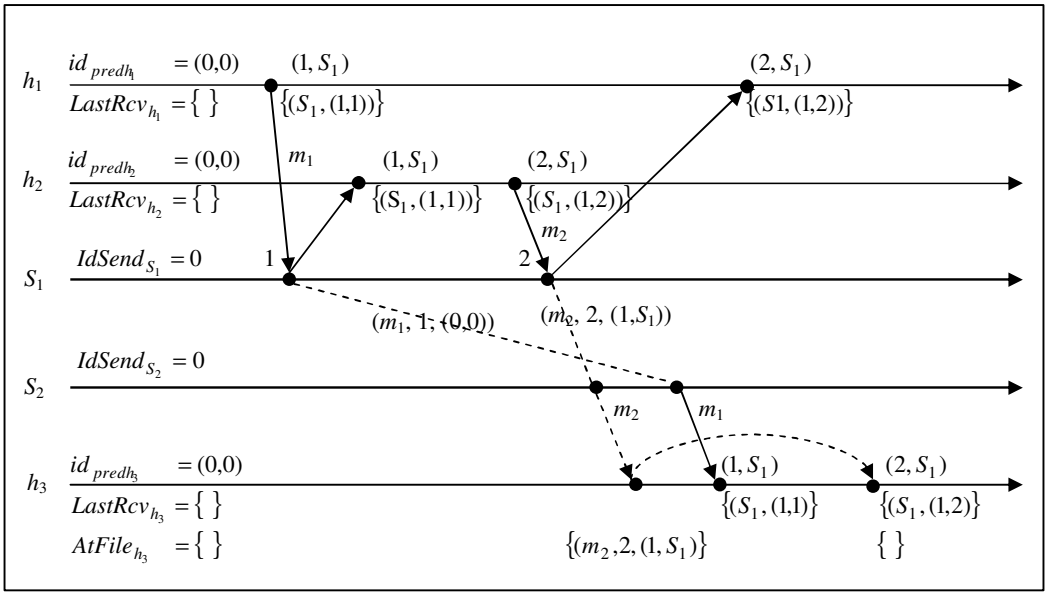

Figure 5.2: $BMobi\_Causal$: An illustrated example.

5.5 Handoff module

The handoff module presented here is similar to that presented in [47, 43].

The handoff for an MH h_i is handled by passing id_{predh_i}, $LastRcv_{h_i}$ and $AtFile_{h_i}$ from the old MSS to the new MSS. Upon the reception of this information, the new MSS verifies if there exist messages in $Unst_M$ not yet received by h_i; i.e. for each message $m \in Unst_M$ if $id_m \notin LastRcv_{h_i}$ then this message is delivered to h_i provided that its delivery condition is verified, otherwise the message is added to the queue $AtFile_{h_i}$.

If h_i migrates to another MSS, S_k, before the current handoff procedure is completed, the new $handoff - begin$ message issued by S_k will not be handled until the current handoff procedure is completed.

19 On the receipt of $(Register(S_i))$ **by** S_j **from** h_i

1. $Send(Handoff - begin(h_i))$ to S_i

20 On the receipt of $(Handoff - begin(h_i))$ **by** S_i **from** S_j

1. $Send(Data(id_{predh_i}, LastRcv_{h_i}, AtFile_{h_i}))$ to S_j

21 On the receipt of $(Data(id_{predh_i}, LastRcv_{h_i}, AtFile_{h_i}))$ **by** S_j **from** S_i

1. $Save(id_{predh_i}, LastRcv_{h_i}, AtFile_{h_i})$
2. $Send(Handoff - over(h_i))$ to S_i
3. **for all** $((m, IdSend_m, id_{pred_m}) \in Unst_{M_{S_j}} \mid IdSend_m \notin LastRcv_{h_i}.SD_l)$ **do**
4. Let $id_{pred_m} = (id_{pred}, S_{pred})$;
5. **if** $(id_{pred} \in LastRcv_{h_i}.SD_{pred})$ **then**
6. $Deliver(m)$ to h_i;
7. **for all** $m' \in AtFile_{h_i}$ **do**
8. **if** $(id_{pred_{m'}} = (IdSend_m, S_l))$ **then**
9. **Deliver**(m') **to** h_i
10. $AtFile_{h_i} := AtFile_{h_i} - (m', IdSend_{m'}, id_{pred_{m'}})$
11. **end if**
12. **end for**
13. **else**
14. $AtFile_{h_i} := AtFile_{h_i} \cup (m, IdSend_m, id_{pred_m})$
15. **end if**
16. **end for**

22 On the receipt of $(Handoff - over(h_i))$ **by** S_i **from** S_j

1. $Delete(id_{predh_i}, LastRcv_{h_i}, AtFile_{h_i})$

5.6 Correctness Proof

5.6.1 Safety property proof

Let two messages m_1 and m_2, and let Id_{m_1} and Id_{m_2}, the identities of emission events of m_1 and m_2, respectively.

We suppose that $Send(m_1) \rightarrow Send(m_2)$ and $\nexists m \mid Send(m_1) \rightarrow Send(m) \rightarrow Send(m_2)$; which means that the emission of m_1 precedes immediately the emission of m_2. We show that if the delivery of messages respects the order of their diffusion for all pairs of messages in *immediate dependency relationship*, then the delivery will respect the causal delivery for all messages. Two cases are to be considered:

Case *m_1 and m_2 are sent by the same MH $h_l(S_l)$:*

Since the emission of m_1 precedes immediately that of m_2, then the emission of m_1 is certainly the last event occurred in $h_l(S_l)$ before the the emission of m_2. So, $id_{pred_{m_2}} = (id_{m_1}, S_l)$. We suppose that the receiving MSS S_j, where h_j is located, delivers m_2 before m_1. This means that the delivery condition for m_2, represented by $\exists (S_l, SD_l) \in LastRcv_{h_j}$ where $id_{m_1} \in SD_l$, is held. This condition can be verified only if S_j has been delivered m_1 to h_j before the delivery of m_2. Which leads to a contradiction with the assumption that m_2 has been delivered before m_1.

Case *m_1 and m_2 are sent by two distinct MHs $h_k(S_k)$ and $h_l(S_l)$, respectively:*

We suppose that m_2 has been delivered before m_1 to the receiving MH h_j while we tray to prove that this is impossible. Since the emission of m_1 precedes immediately that of m_2, then the delivery of m_1 is certainly the last event occurred in $h_l(S_l)$ before the the emission of m_2. So, $id_{pred_{m_2}} = (id_{m_1}, S_l)$. The delivery of m_2 by $S_j(h_j)$ is constrained by the condition: $\exists(S_l, SD_l) \in LastRcv_{h_j}$ where $id_{m_1} \in SD_l$ and this can be verified only if S_j has delivered m_1 before m_2. This leads also to a contradiction.

5.6.2 Liveness property proof

The delivery of a message m to a destination MH h_j can be delayed only by other messages that are not yet delivered to h_j. In order to proof that this waiting delay is finite, we proceed as follow: we consider all messages which are not yet delivered to $h_j(S_j)$. The happened before relationship can be used to define a partial order over the emission events of these messages. Let m' one of these messages in the partial order of which the emission has no predecessor. Hence m' has not been delivered to h_j at its reception, then the following condition must be true $id_{pred_{m'}} \neq (0, 0)$ and this can be valid only if m' is causally related to another message sent before it, which leads to a contradiction with the hypothesis that m' has no predecessor. This means that our protocol never delays a message indefinitely.

5.6.3 Exactly once delivery property proof

Our protocol ensures that exactly once copy is delivered for each broadcasted message. This is can be inferred from the following observations:

The delivery of a message m to an MH h_i is expressed by the addition of its identity to the set $LastRcv_{h_i}$. Thus, the scan of this set either at the reception or handoff phase prevents the delivery of another copy of the same message. Therefore, *at most-once* delivery of a message is ensured.

A broadcast message m is buffered at every MSS till an explicit $Delete(id_m)$ message is received. This $Delete()$ message is sent by the sending MSS only after an acknowledgement from each MH in g has been received. Which ensures *at least-once* delivery of a message.

Exactly-once delivery is thus ensured by at least-once and at most-once delivery of a message.

5.7 Complexity Analysis

In this section, we will analyze our protocol ($BMobi\_Causal$) and compare it with related works. Tables 5.1 and 5.2 show the complexity results of some existing causal ordering protocols for mobile systems. We will discuss the communication overhead, the

storage overhead, the handoff complexity and the possibility of unnecessary inhibition delay.

Protocol	Message overhead (wired channels)	Number of Dest. sites	Message overhead (wireless channels)
PRS	$O(n_h^2)$	$O(n_h)$	0
LH	$O(n_s)$	$O(n_s)$	0
$CYTH$	$O(n_s)$	$O(n_s)$	0
CK	$O(n_h^2)$	$O(n_s)$	0
$MOCAVI$	$O(n_h)$	$O(n_s \in g)$	$O(n_h)$
$MMobi\_Causal$	$\sum_{i=1}^{n_s} l_i \ll n_s \times \lvert MH_{View}\rvert$	$O(\lvert MSS_{View}\rvert)$	0
$BMobi\_Causal$	$O(1)$	$O(\lvert MSS_{View}\rvert)$	0

Table 5.1: Comparison between related works.

Protocol	Storage overhead	Handoff complexity
PRS	$O(n_h^2)$	$O(n_h^2)$
LH	$O(n_s)$	$O(n_s)$
$CYTH$	$O(n_s)$	$O(1)$
CK	$O(n_h^2 + n_s)$	$O(n_h^2 + n_s)$
$MMobi\_Causal$	$O(n_s)$	$O(n_s)$
$BMobi\_Causal$	$O(n_s)$	$O(n_s)$

Table 5.2: Comparison between related works (Cont.)

In our protocol, we need to append only information about the immediate predecessor to each message to maintain causal broadcast. This information is represented by the tuple id_{pred}. Thus, the message overhead of our protocol is $O(1)$. A message is broadcast only to $MSSs$ containing the group members which permits to reduce the number of messages sent over wired channels. So, the number of destinations of a given message is $O(|MSS_{View}|)$. From Table 5.1, it can be seen that our protocol outperforms the related works with respect to communication overhead. We can thus easily notice the advantage of developing a causal protocol dedicated to a broadcast communication instead of considering it as a particular case of a multicast communication.

The storage overhead refers to the size of the protocol related data structures maintained by a MSS for MHs. The amount of data needs to be as small as possible. In our protocol, an $MSS \in MSS_{View}$ maintains a list $LastRcv$ and a tuple id_{pred} for each local MH member of the group. Unlike our unicast and multicast protocols, this protocol compensates for the problem of non-bounded evolution of $LastRcv$ which results from the characteristics of dependency sequences mechanism. Thus, the storage overhead for each MH is $O(n_s)$. The defined data structures are independent from the number of MHs in the system, which renders our protocol scalable relatively to the number of MHs and more suitable for large groups.

Handoff complexity indicates the amount of information exchanged between the

base stations involved in the execution of the handoff procedure. To handle a handoff in our protocol, one message containing several scalars and $LastRcv$ list requires to be transferred between the two $MSSs$. So, $O(1)$ message of size $O(n_s)$ is needed. Comparing to the related works (see Table 2), we show that the $CYTH$ [71] protocol provides the best handoff complexity but its handoff procedure involves three entities to achieve the migration of an MH (its last MSS, its new MSS and its serving MSS). Each time a migration is done, the serving MSS must be contacted to update the current location of an MH. Thus, a supplementary delay is invoked in the execution of the handoff procedure and which can be considerable if the serving MSS is so far.

One advantage of our broadcast protocol ($BMobi\_Causal$), like our unicast and multicast protocols, is the elimination of unnecessary inhibition delays. In the protocol PRS [150], causal ordering is explicitly maintained among MHs. Hence, inhibition delay never occurs but, this protocol requires $O(n_h^2)$ message overhead.

5.8 Conclusion

In this chapter, we presented a solution that proves the advantage of developing a dedicated causal broadcast delivery protocol based on process behavior's awareness. We discussed how the rather well known behavior of a process can influences the control information's size. We showed that by viewing the behavior of the execution in the sense that the reaction to occurring events is instantaneous, tracking the dependency between messages is possible by only keeping information about the message's *immediate predecessor*. Hence, causal broadcast delivery of messages can be easily achieved in a mobile group with a small message overhead.

The proposed protocol keeps all the interesting characteristics of [47, 43]. It outperforms its counterparts in terms of communication overhead.

Chapter 6

Group Communication Concept

6.1 Introduction

Multicasting and broadcasting mechanisms are intrinsically tied to *group concept*. In real-world, a *group* is a set of people, organizations, or things which are considered together because they have something in common [73]. Obviously, people have a natural tendency to gather around a common goal or interest, and form groups. Similarly, in distributed computing systems, processes are often organized into groups for supporting various applications, such as dissemination applications, collaborative applications, fault tolerant applications, conversational applications, etc. All these applications are based on *group communication* paradigm. Group communication is an abstraction which deals with multicasting messages from source(s) to members of a group.

The causal multicast and broadcast delivery protocols presented in Chapter 4 and Chapter 5, respectively, assume the *static* group model; that is, a group in which group composition is known at system initialization and does not change in time. However, a static group has practical limitations. In practice, it is often desirable that processes join and leave groups at runtime. Removing a crashed process from the group can also be desirable. This requires the support of *dynamic* group communication. The key component of a dynamic group is the *group membership* service, which is responsible for adding and removing processes during the computation.

The aim of this chapter is to motivate our last contribution (see Chapter 7), which consists on extending the two proposed protocols ($MMobi\_Causal$ (see Chapter 4) and $BMobi\_Causal$ (see Chapter 5)) to support dynamic mobile groups. We give an overview about various ideas and concepts behind group communication, with a focus on group management solutions and causal consistency.

The chapter is organized as follow: The next sections (Section 6.2, 6.3, 6.4) give preliminaries and important definitions for a good understanding of the subject. Section 6.5 outlines the traditional group communication solutions, designed for wired networks, while Section 6.6 deals with work undertaken in the context of mobile systems. In Section 6.7, we discuss how the existing group communication solutions and

causal ordering protocols support each other. Finally, conclusion is given.

6.2 Definitions

The *group communication* paradigm [72] serves as building blocks, or middleware for reliable and fault-tolerant distributed systems. Group communication provides an abstraction to many processes communicating on events so that processes which share a common interest can receive events disseminated by other processes sharing the same interest, while processes which do not share the same interest are not involved in the communication process. Processes sharing a common interest are said to form a *group*. Groups can be either static or dynamic. A *static group* is a group having a constant membership; that is, the set of processes members of the group does not change over time. A group whose membership changes over time is called a *dynamic group*.

With dynamic groups new problems need to be addressed. One problem is how to add and how to remove processes from the group. This problem is called the *group membership* problem. the successive memberships of the group are called the *views* of the group [72]. A membership service allows processes to join and leave the group dynamically and every group member is allowed to communicate with other processes by using an event dissemination service.

One distinguishes among three different ways of how processes in a group communicate:

- *unicast communication* considers a peer-to-peer communication where a single process informs another process about an event,
- *multicast communication* considers a single process informing a subset of group's processes about the same event,
- *broadcast communication* considers a single process to multicast an event to all processes in a group.

Another problem is to adapt the definition of group communication primitives from static groups to dynamic groups. With dynamic groups, the basic communication abstraction is called *view synchrony*, which ensures that processes deliver the same set of messages between two membership changes. It can be seen as the counterpart of reliable broadcast in static systems [79]. Roughly speaking, view synchrony adopts a similar definition, while relaxing the Agreement property. Causal order broadcast in a system with dynamic groups can thus be specified as view synchrony, plus a property of causal order.

6.3 Typical Group Communication Services

A group communication system (GCS) (See Figure 6.1) usually integrates a *group membership service* with a *reliable multicast service*. The former deals with group composition and group dynamics where the latter handles data delivery among group members. Furthermore, several GCSs offer semantic models such as the virtual synchrony [56], and the extended virtual synchrony [139].

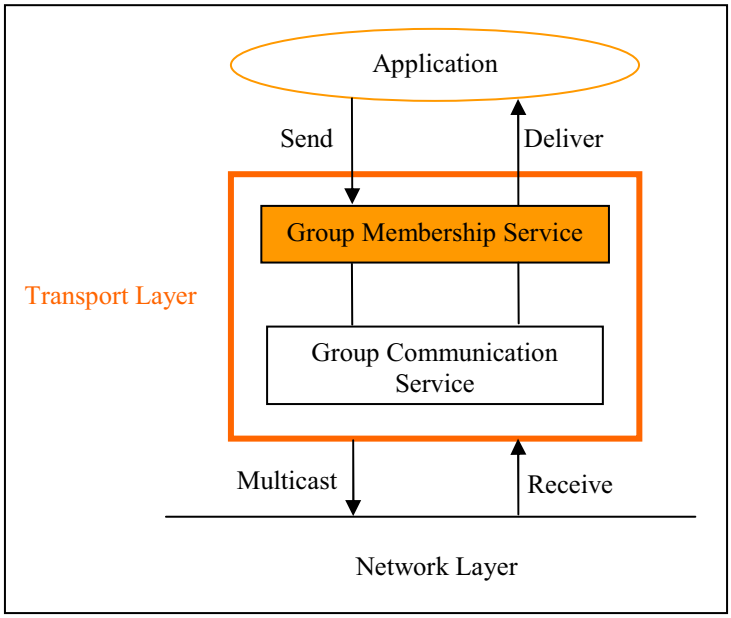

Figure 6.1: Architecture of a group communication system.

In *GCSs*, a *group* is a set of processes which are *members* of the group. Each group is associated with a unique identifier representing its logical *name*. The GCS is in charge of transmitting messages exchanged within the group to all concerned group members. A process can join an existing group. It can also leave its group and be eliminated from the group due to a failure, a disconnection, or a partitioning.

6.3.1 Group Membership Service

The task of a group membership service is to maintain a list of the currently alive and connected group members (processes). In essence, this service tracks the dynamic evolution of the list as it evolves over time, taking into account both voluntary requests to join and leave the group as well as process failures and network partitions. This list is called *view*. The membership service must handle such dynamics in a coherent way, i.e., all members of the group must have a consistent view of the group's composition despite failures [72]. Typically, a membership service does not distinguish when a member voluntarily leaves the group or when it leaves the group due to system faults i.e., group membership services equally respond to network failures (e.g., process crashes, link

failures and recoveries) and to requests by processes to join or leave.

When communication links fail, the network can become partitioned; that is, some of the nodes can no longer communicate, as all links between them are broken. When this happens, a group can be split into several isolated subgroups (or partitions). There are two main approaches to cope with such situation: (1) the primary component membership, and (2) the partitionable membership.

A *primary component* membership service always runs only on one partition, which is called primary partition. Processes in the parts of the network which are not connected to the primary partition are considered faulty. This kind of service is appropriate in small stable networks, e.g., in wired LANs. On the other hand, *partitionable* group membership service allows multiple network partitions to run the service simultaneously. That is, if a group splits due to network partitions, all its parts continue working. If the network becomes connected again, the parts of the group merge. Partitionable group membership service should be used in WANs and mobile networks.

In a primary component membership service, views installed by all the processes in the system are totally ordered. In a partitionable one, views are only partially ordered (i.e., multiple disjoint views of the same group may exist concurrently in different network partitions).

6.3.2 Group Communication Service

The group communication service multicasts messages to group's members by ensuring different properties like reliable, causal, atomic or agreement multicast. It is usually constructed relying on the group membership service, although that exist some works which propose to implement the multicast service directly over a non reliable failure detector[5].

The classical group communication provides a one-to-many or many-to-many communication model typically based on a reliable multicast protocol which permits to a group's member to send messages to all members of this group. Several message ordering criteria in addition to reliable multicast are provided by group communication [95]. Message ordering simplifies application programming by ensuring that each message is handled in a predictable context resulting from previous messages.

Group communication mechanisms typically offer different delivery services, ranging from *best-effort unreliable* delivery to *reliable* delivery, which ensures that messages sent among non-faulty processes are not lost; from *unordered* services that deliver messages in arbitrarily different orders to different group members, to *totally* ordered services that deliver messages in the same order to all the group members passing by *causally* ordered services that deliver messages accordingly to the cause and effect relationship.

Unreliable Delivery According to this scheme, there is no guarantee on the delivery and the order according to which events are notified to a given group communication service. this in essence a best-effort service.

Simple Reliable Delivery According to this scheme, only the reliability property is satisfied without any guarantee on the order of the delivery process. As a consequence, a simple reliable delivery assures that each generated event will be eventually delivered by all the correct processes.

FIFO Order messages are delivered in order as sent by the sender. If a process multicast a message m before it multicast a message m', then no correct process delivers m' unless it has previously delivered m.

Causal Order messages are delivered in order as happened before sent order. If the multicast of a message m causally precedes the multicast of a message m', then no correct process delivers m' unless it has previously delivered m.

Total Order messages are delivered in same order by all processes. If correct processes p and q both deliver messages m and m', then p delivers m before m' if and only if q delivers m before m'.

6.3.3 Virtual Synchrony Service

Most GCSs strive to present different members of the same group with mutually consistent perceptions of the communication done in the group. This perception is known as Virtual Synchrony semantics [56].

A key property of the virtual synchrony execution model is *virtually synchronous delivery* [53, 72]; this property specifies that if the two views are delivered consecutively to several processes, then exactly the same multicast messages are delivered to these processes between these two views.

Virtual synchrony assumes message omission faults and fail-stop process faults; i.e., a process that fails can never (or is not allowed to) recover. When network partitioning occurs, virtual synchrony ensures that processes in at most one connected component of the network, the primary component, are able to make progress; processes in other components become blocked.

To overcome the limitations of virtual synchrony to cope with network partitions and re-merges, and with process recoveries, several variations on virtual synchrony semantics for both primary component and partitionable membership have been suggested, such as enriched view synchrony [29], weak virtual synchrony [92], extended virtual synchrony [140] or optimistic virtual synchrony [173].

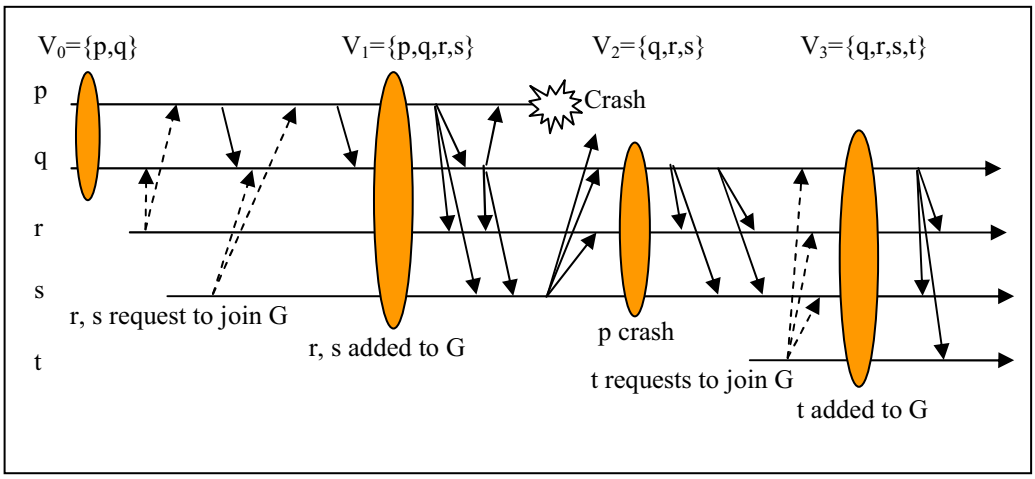

Figure 6.2: Virtual synchrony model.

6.4 Specification

It is important for group communication services to maintain a *well-defined* semantics of the service. A developer of an application can be based on these semantics in order to design correct applications. The semantics must specify the considered hypothesis and the provided guarantees.

Several studies present formal specifications for GCSs in distributed computing [95, 26, 96, 78, 87, 31, 72, 164]. These works discuss and formalize aspects related to group communication services. The specification of static group communication primitives can be found in [95]. Contrary to the specification of static group communication concerned only with *communication primitives*, the specification of dynamic group communication has two parts: a communication primitives part, and another part dedicated to *group membership changes*. Comprehensive surveys of group membership and view synchrony properties are found in [96, 72]. Fekete *et al.* [87] and Babaoglu *et al.* [31] present a formal specification for a partitionable group communication service. The work done in [164] extends the specification in [95] to the case of dynamic groups.

The specifications capture a core set of properties that is commonly provided by group communication systems and that have been shown useful for facilitating implementations of many distributed applications and other, stronger, group communication properties (see [72]).

6.4.1 group membership safety properties

Property 6.1 (Self Inclusion)**.** *If process p installs view V, then p is a member of V.*

Property 6.1 requires that a process always is member of its view, because a membership of a group reflects the ability to communicate with the processes in the group and a process can always communicate with itself.

Property 6.2 (Local Monotonicity)**.** *If a process p installs a view V after having installed a view V' then the identifier of V must be greater than that of V'.*

Property 6.2 requires that view identifiers of the views that each process installs, are monotonically increasing. This guarantees that each view is installed only once and that views are installed in the same order.

Property 6.3 (Initial View Event)**.** *All events (for example emission and reception events) must occur within some view.*

Property 6.4 (Primary Component Membership)**.** *The set of installed views forms a sequence such as the intersection of two consecutive views in this sequence is not empty.*

While properties presented above are fulfilled by partitionable as well as primary-component membership services, *Property* 6.4 concerns only primary-component ones. This property enforces a total order on views. It implies that for every pair of consecutive views, there is a process that survives from the first view to the second.

6.4.2 Multicast Safety Properties

Property 6.5 (Delivery Integrity)**.** *For every receive event there is a preceding send event of the same message.*

Property 6.5 requires that there should be no spontaneously generated messages by the GCS.

Property 6.6 (No Duplication)**.** *Two different receive events with the same content cannot occur at the same process.*

Property 6.6 states that messages are not duplicated by the GCS; that is, every message is received at most once by each process.

Property 6.7 (Sending View Delivery)**.** *If a process receives message m in view V, and some process q (possibly $p = q$) sends m in view V', then $V = V'$.*

Property 6.7 guarantees that a message should be delivered in the same view as it was sent.

Property 6.8 (Same View Delivery)**.** *If processes p and q both receive message m, they receive m in the same view.*

Property 6.9 (Virtual Synchrony)**.** *If processes p and q install the same view V in the same previous view V', then any message received by p in V' is also received by q in V'.*

6.5 Group Communication Solutions in Static Environment

ISIS [57, 165, 56, 55, 59] is one of the first system to propose group communication. It is a primary partition system based on the concepts of process group and virtual synchrony. The main layers of the ISIS architecture include the group membership layer and the multicast layer. The former handles group membership changes due to processes voluntarily joining or leaving the group, or due to process failures. The latter is responsible for multicasting messages to process groups. Two multicast primitives are provided: The $CBCAST$ primitive guaranteeing causally ordered message delivery and the $ABCAST$ primitive providing a totally ordered delivery. The $ABCAST$ primitive is constructed over the $CBCAST$ primitive.

The *V* system [69] provides group communication services at the operating system level. It was the first to utilize hardware multicast to implement process group communication. However, only non-reliable, best-effort, unordered delivery service is provided. Similar services for wide area networks are provided by the IP-multicast protocol.

The *TPM* protocol [153] is not partitionable (it only supports primary partition membership). The TPM main principle is to achieve totally ordered broadcast by circulating a token around a logical ring imposed on the group members participating in the current view. The token contains the next sequence number to be stamped on new messages. TPM also provides a dynamic membership and token regeneration algorithm.

Delta-4 system [148] addressed reliability, fault-tolerance, and real-time constraints, specially oriented towards factory applications. Delta-4 is supports a primary component group membership service [161]. Delta-4 didn't use the concept of virtual synchrony, but uses group protocols providing similar functionality. One of the more protocols developed in this project is xAMp [160]. xAMp provides a reliable causal multicast service. Delta-4 uses a special hardware for message ordering and failure detection. This seems to be a strong limitation on the project's usability.

The *Amoeba* [103] distributed operating system uses a reliable multicast protocol [102] to support high level services such as fault-tolerant directory service [101]. Amoeba is based on the use of a centralized-sequencer approach for totally ordering multicasts and storing multicasts that have not been acknowledged yet. The Amoeba system is resilient to any pre-defined number of failed processors, but its performance degrades as the number of allowed failures is increased.

The *Trans* and *Total* protocols [131, 142, 132] provide reliable ordered broadcast delivery in an asynchronous environment. Trans [132] uses an acknowledgement mechanism and exploits transitivity of acknowledgements to define a partial order on messages. The Total protocol, layered on top of the Trans protocol, extends the partial

order of Trans into a total order. The Trans and Total protocols maintain causality and ensure that operational processors continue to order messages even though other processors have failed, provided that a resiliency constraint is met. A membership protocol [141] is implemented on top of Total. If a processor suspects another processor, it sends a fault message for the suspected processor. When that message is ordered, the membership is changed to exclude this processor. The limitation of that architecture is that if Total cannot order the membership messages (e.g. because the resiliency constraint is not met), the system is blocked.

In *Psync* [136, 146], processes dynamically build a causality graph of messages they receive. Psync then delivers messages according to a total order that is an extension of the causal order. Based on the causal order provided by Psync, a membership algorithm is constructed. Using this algorithm, processors reach eventual agreement on membership changes. The algorithm handles processor faults and allows a processor to join a pre-existing group asymmetrically. Network partitions and re-merges are not supported.

The *Newtop* protocol [126, 125] replaces the context graph of Psync [146] by the notion of *causal blocks*. Each causal block defines a set of messages. All the messages within a block are causally independent. The blocks are totally ordered. The messages in a block are delivered together, in some deterministic order. Newtop is based on a partitionable group membership service that handles processor crashes and network partitions. However, process recoveries and network re-merges are not addressed. The Newtop platform leaves it to applications to decide whether or not they should maintain more than one subgroup in such a situation.

The *Horus* project [178, 179] implements group communication services, providing unreliable or reliable FIFO, causal, or total multicast services. Horus is extensively layered and highly configurable, allowing applications to only pay for the overhead of services they use. The layers include the COM layer which provides basic non-reliable multicast, the NAK layer which provides reliable FIFO multicast, the MBRSHIP layer that provides membership maintenance, the STABLE layer which provides message stability, the CAUSAL and TOTAL layers, the LWG layer which maintains process groups, the EVS layer which maintains extended virtual synchrony, and many more.

Transis [127, 21, 20] provides group communication services using non-reliable hardware multicast and tolerating network partitions and merges as well as processor crashes and recoveries. Three multicast primitives are provided according to the extended virtual synchrony semantics: causal multicast, agreed multicast for total order delivery, and Safe multicast that provides even stronger guarantees. Two different reliable multicast protocols are implemented in Transis. Lansis [21], the earlier protocol, uses a direct acyclic graph (DAG) representing the causal relation on messages to provide reliable multicast. The causal order mechanism in Lansis is derived from the Trans

protocol [132]. The second reliable multicast protocol in Transis is the Ring protocol [22]. The membership algorithm [20] of Transis is a symmetric protocol that was the first to handle network partitions and re-merges. The basic idea of this membership algorithm was adopted by Totem and Horus.

The *Totem* system [6, 22, 23] provides reliable multicast and membership services across a collection of local-area networks. The Totem system is composed of a hierarchy of two protocols. The bottom layer is the Ring protocol which provides reliable multicast and processor membership services within a broadcast domain. The upper layer is the Multiple-Rings protocol [6] that provides reliable delivery and ordering across the entire network. The membership layer, apart from detecting failures and defining views, recovers token and messages that had not been received by some members when failures occur. The recovery layer completes the membership layer, by ensuring the extended view synchrony property.

RMP [183] is GCS directed towards embedded applications, specially in the space industry (mainly, NASA). The RMP protocol has been influenced by Chang-Maxemchuks's atomic broadcast algorithm [66]. In RMP, the membership layer is split into two parts: *fault-free* membership and *fault-tolerant* membership. The fault-free membership handles joins and leaves in the absence of failures, using the underlying atomic broadcast layer. The role of the fault-tolerant membership layer is to avoid blocking by excluding processes that suspected to have crashed. The fault-tolerant membership protocol is based on a two-phase commit protocol [49] among the surviving processes. This fault-tolerant has also the responsibility to ensure the view synchrony property.

JGroup [138, 137] is a group-enhanced extension of the Java RMI distributed objet model. It is based on the formal specification for partitionable group membership and its algorithms given by Babaoglu *et al.* [31]. Groups present a collection of *server objects* that cooperate to offer distributed services. The states of the objects in different partitions are merged when the partitions remerge. Coordination among group members is obtained through the facilities provided by the *partitionable group membership* service, the *reliable group communication* service and the *state merging* service included in Jgroup.

Relacs [30, 32] is a GCS which is designed to fit the wide area networks. To reduce the cost of membership agreement, Relacs distinguishes the members to multiple roles: *core* members, *client* members, and *sink* members. Core members manage the group membership and message delivery. Client members send and receive messages by means of communicating with core member, but do not involve the membership change. Sink members only receive the message from core members. Relacs provides view synchronous communication, which allow the partitionable group membership.

Spread [19] is a partitionable client/server based GCS for WANs. It is based on an overlay network that provides a messaging service resilient to faults across local

and wide-area networks. It provides several types of reliability (Unreliable, Reliable), ordering (Unordered, FIFO, Causal, Agreed), and stability (Safe) services for messages. The Spread system is based on a daemon-client model where daemons establish the message dissemination network and provide membership and ordering services, while clients can reside anywhere on the network and will connect to the closest daemon to gain access to the group communication services. *Secure Spread* [24] is a variant of Spread that integrates security services.

Keidar's approach [27, 106, 107, 108] decouples two key components of GCSs Virtual Synchrony and Membership. Such decoupling has been regarded as critical for providing scalable and efficient group communication services in wide-area networks: It allows for the utilization of a scalable architecture for group membership in which a small set of membership servers maintains group membership of a large set of clients. Keidar's approach avoids the delivery of obsolete views and reaches agreement upon the membership change in a single round.

Layered approach [112] is not a full GCS; it only provides a group membership service. It is designed to improve the scalability using a hierarchical architecture of membership servers. Only the membership servers maintain the membership information on the group view, and participate in the view change. These servers are configured in multiple layers and achieve gradual agreement (See Figure 6.3). The grouping of membership servers is base on geographical proximity.

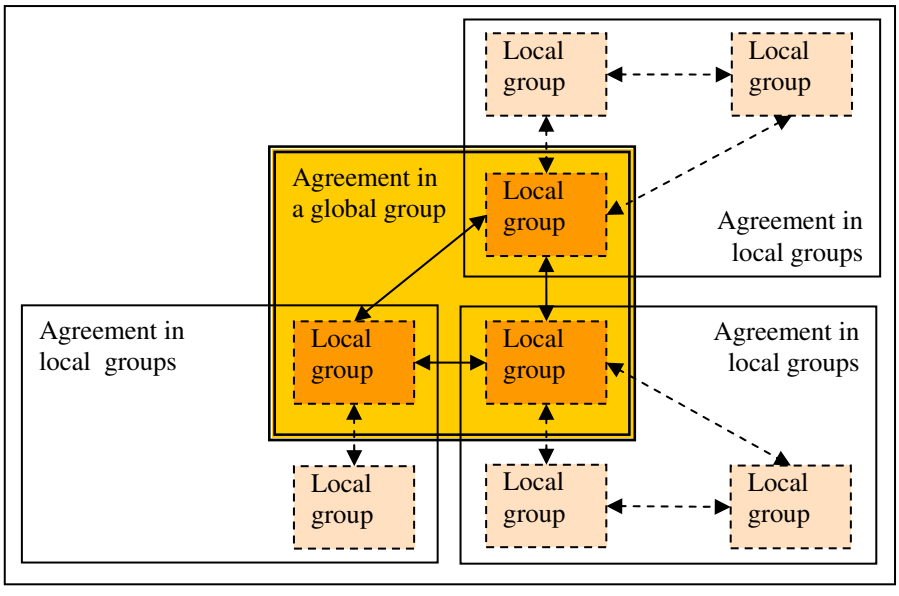

Figure 6.3: The layered approach architecture.

The *InterGroup* system [48] is designed specifically with the intention of scaling to the Internet and to large numbers of participants. The system is divided into four major components: a control information service, a reliability service, a delivery service, and a membership service. A cornerstone of the InterGroup approach is the recognition

that the message order and reliability constraints of a GCS can be met by keeping only the processes currently sending messages in the group membership. The levels of the message delivery service range from unreliable unordered to reliable group timestamp ordered.

6.6 Group Communication Solutions in Mobile Environment

6.6.1 Issues in Mobile Environment

The design, development and deployment of group membership and communication systems in mobile environments raise challenging issues. The major issues are outlined below:

- GCS should handle a handoff situation. In handoff process, a mobile host may be disconnected with the group temporally and reconnected, thus it may receive multiple copies of the same message from different $MSSs$ or not receive some messages.
- The system must distinct a disconnected mode and an idle mode from failure. Users could pass through areas where wireless communication is not possible, either because of physical obstructions or because the area is not covered by the wireless network. Moreover, since the mobile node has limited resources, the mobile nodes may switch the idle mode or disconnected mode to reduce the power consumption. However, these members do not leave the group intentionally, and rejoin the member some time later.
- Technological limitations in the lifetime of batteries make power consumption a major issue. The energy necessary of sending a message across a wireless link may be much greater than the energy for receiving a message.
- Mobile hosts could be resource-poor devices (such as the so-called Personal Digital Assistants) that have, in particular, limited storage and computing capabilities.

As a consequence it is necessary to re-think and re-design traditional group management solutions. Several efforts are done to construct the kind of group management infrastructure required to support collaborative applications in environments characterized by wireless connectivity and user/device mobility.

6.6.2 Acharya *et al.*'s Approach

Acharya and Badrinath [3] proposed a host-view membership protocol to support the multicast communication in mobile Internet. Instead of separately tracking the location

of every MH in the group, it maintains a set of locations ($MSSs$) such that every member of the group resides in some cell in this set (noted $host-view$). They manage changes to a group's host-view that are caused solely due to mobility of group members, i.e. they assume membership of a multicast group does not change during the group's lifetime. The protocol followed the approach of using a *central coordinator* to serialize changes to the set $host-view$. Handoff is enriched by the actions necessary for location view management, in particular, when a group member moves to an empty cell or when the last group member leaves a cell. However, the protocol does not guarantee the semantics of group membership service and it does not manage the temporary deconnexion/reconnexion of mobile hosts.

6.6.3 Bartoli's Approach

Bartoli [41] presented a protocol for totally-ordered multicast communication within a dynamic group of mobile recipients. Bartoli's approach assumes the *incomplete spatial coverage* of wireless links; that is, the union of all cells may *not* cover the entire area where mobile hosts may be located. The group-based multicast protocol is composed of MHs and gateways connected by wired network. One of gateways is defined as *coordinator*, which maintains group view and supports the reliable order communication. An MH intending to join or leave a group sends a message having a join or leave tag to coordinator. For a message transmission, messages are transmitted to the coordinator, and the coordinator multicasts that message to the group with a sequence number. Each gateway broadcasts a beacon message to know his group member periodically. Received a beacon message, an MH sends a greeting message. Each gateway maintains a set of mobile host identifiers called $local$ and of a set of group member identifiers called $local-m$. When a group member switches between cells, the related gateways do not exchange any information, but they simply update autonomously the respective $local$ and $local-m$. The handoff can be handled without view change since the group view is managed by coordinator. However, this protocol does not guarantee group membership semantics like the virtual synchrony semantic.

6.6.4 MGCM

Mobile group communication model (MGCM) [84] is a GCS on networks with MHs based on ISIS system [57]. MGCM is a layered model (Figure 6.4) where the main layer is the mobile multicast protocol, MM-Cast. It provides exactly one delivery of multicast messages to MHs irrespective of their location or mobility. In absence of failure, the MM_Cast protocol ensures atomicity. Causal ordering and total ordering provide the main ordering requirements for the primitives of the model. The causal ordering protocol is based on the use of vector clocks, while the total ordering is ensured by using a centralized scheme which assigns one MSS as a coordinator. The case of an MH

changing its group membership is handled by the topmost layer which supports dynamic group membership changes. When the MSS receives group view change message, it operates a flushing protocol. At first, MSS sends message reporting the start of a new group view installation to all $MSSs$. On receiving the flush message, each MSS stores requests from local MHs, installs the new group view, and sends a reply message to all $MSSs$. On receiving the flush reply from all $MSSs$, a MSS deals with stored requests from his MHs. This layer provides virtual synchrony execution during membership changes. However, this approach has a drawback on the performance because the whole systems should execute view change to preserve group communication semantics in temporarily disconnected situations. Moreover, MGCM supports only smooth hand-off, and does not handle long-term handoff.

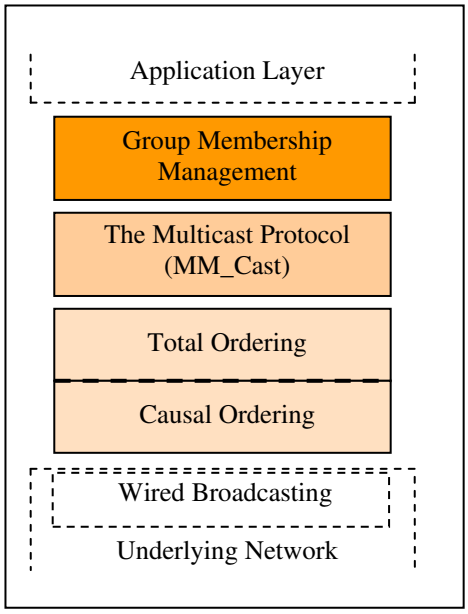

Figure 6.4: The MGCM architecture.

6.6.5 SGM for Mobile Internet

The authors in [110, 111] proposed a scalable group membership scheme to support an Internet environment with MHs. The proposed scheme exploits a layered approach (Figure 6.5) similar to that proposed in [112], but, to isolate mobility and temporary disconnection from the group view, the hierarchical view management of the layered approach has been extended. In the proposed scheme, each wireless cell becomes a local group. The group membership service is supported by each MSS in the network. To manage the group view, each MSS maintains three lists: *global-view*; to maintain a global view, i.e., a list of all members in the group. *local-view*, which represents the list of group members located in its cell. And *suspected-view*, which represents the list of the temporarily disconnected group members. Unlike the existing approaches,

the proposed scheme localizes the temporary view inconsistency due to mobility and disconnection without a view change. That is, in temporary disconnection of an MH, MSS simply moves the entry corresponding to this MH from the local view to the suspected view. When a handoff occurs, the update of local view is only done between the two servers participating to the handoff procedure. Moreover, if handoff occurs during the view installation, this last is blocked until completing handoff procedure to avoid view changes. The handoff and the disconnection/reconnection of an MH are detected by a failure detector using timeouts.

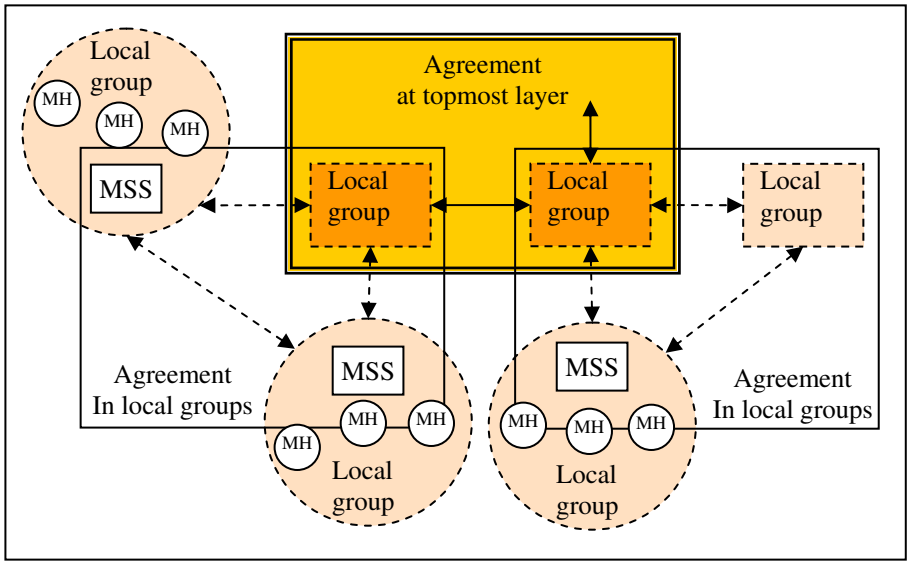

Figure 6.5: The SGM for Mobile Internet architecture.

6.6.6 MaGMA

MaGMA [118] (Mobility and Group Management Architecture) is an architecture for groupware support in mobile networks. MaGMA's architecture consists of a collection of *Mobile Group Managers* (MGMs), which manage group membership and also implement a multicast overlay for data delivery. Each MH interacts with an MGM proximate to it. MaGMA supports a subscription model in which nodes can request to be notified of other nodes' mobility.

Authors proposed a number of group management protocols. The first two protocols are based on existing mobility solutions, namely Mobile IP and DNS-based mobility. In Mobile IP case, the responsibility for mobility management is delegated to Mobile IP, and the MGMs only map group names to nodes' home addresses. The main drawback of this solution is the uncontrolled delay and QoS degradation resulting from Mobile IP's triangle routing. In DNS-based solution, similar to the Mobile IP based solution, the mobility handling is delegated to the nodes' home domain DNS servers, and the MGMs map each group to its subscribed MHs in a domain name format. With this

approach, a retrieve operation first gets from the MGM a list of MH names and then employs DNS queries to translate the MH names to actual IP addresses. However, the retrieve-translate procedure can take a substantial amount of time.

Two additional protocols were proposed by authors, in which mobility is explicitly handled by the MGMs and not delegated to other services. In the first protocol, MGMFlood, each MGM forwards (floods) to all other MGMs all control messages (join/leave/move) received from MHs in its domain. When an MH crashes, its local MGM detects the crash and sends an appropriate leave message to all other MGMs. MGMFlood is simple and allows for seamless handoff due to its prompt reaction to mobility updates. However, it entails high control message overhead, as all MGMs keep views of all groups, including groups not residing in their domains. This solution may not scale well, especially if there are many small groups and localized memberships. The second solution, MGMLeader, reduces the overhead by propagating updates only to those MGMs that have group members in their domains. When an MGM receives an MH's message regarding a group that is represented in its domain, it extracts the MGMs that have members in the group from its local view, and forwards the message only to those MGMs. If an MGM receives a control message (join or move) for a group that does not yet exist in its domain, then it needs to discover the group's up-to-date view, and to forward the event to the appropriate MGMs. In order to minimize the control overhead and ensure view consistency, only one of the participating MGMs sends the view to the new MGM. To this end, one MGM is designated as the *coordinator* of the group. MGMLeader is preferable for sparse groups, whereas MGMFlood is more adequate for dense groups in which all or most MGMs participate.

6.6.7 Proximity Groups

An extension of classical process groups, called *proximity groups*, has been identified as a useful communication paradigm for mobile applications [130]. Proximity groups allow to potentially mobile components of an application to join a proximity group and to interact consequently with its members when they are found in the same geographic area.

In classical group communications, a group is defined by its functional aspect, e.g. its name. The notion of proximity group involves both location and functional aspects; i.e., to be able to apply for group membership, a node must firstly be located in the geographical area corresponding to the group and secondly be interested in the group. From functional point of view, the localization often makes sense to define a group in a mobile application in terms of a geographical area. We can easily imagine many cases where this would be interesting: in traffic management, for example, the area around a traffic-light could be used to define a group with cars in that vicinity becoming members of the group to receive notifications of changes to the state of the lights; in a similar

way, we might want to define a group corresponding to the area around an ambulance in order to inform nearby cars to yield the right of way. In geographical terms, we can use location information to, for example, anticipate partitions and hence take preventive measures to ensure consistent group views when these partitions happen.

Definition of proximity groups

The definition of a proximity group relies on the definition of the *area* that it covers. This area is defined as a *geometric shape* with an associated *reference point*. The reference point is either attached to a *fixed* point in space or to an identified node, its so-called *navel*.

The figure below (Figure 6.6) illustrates this notion of an area. The first shape S on the left is associated with a reference point R. This reference point is relative to the shape. The definition of the area is not complete since R has not been attached to a point (possibly moving) in space. For the second shape, R has been attached to the point $(0, 0)$, making the area *absolute*. The reference point of the third shape has been attached to a node M which represents the navel of this *relative* proximity group.

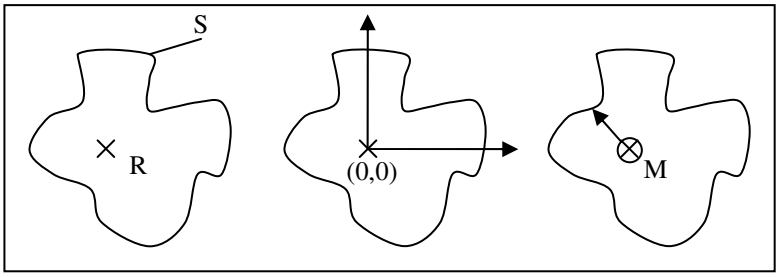

Figure 6.6: Area definition.

Recall that in typical group communication, a group is defined by a topic (or a name) and nodes can join this group if they are interested in its topic. This is also necessary for proximity groups because a node in the area of the group may or may not be interested in joining the group. So, a proximity group can be defined as follow [109]:

Definition 6.1. *A proximity group G is completely defined by the shape, the reference point, the navel and the name:*

$$G = \{Shape, ReferencePoint, Navel, Name\}$$

Prakash and Baldoni's Approach

Prakash and Baldoni [149] presented an architecture for group communication in the context of mobility. Three different types of mobile networks are considered: (1) A cellular and (2) a virtual cellular model in which base stations are mobile, too, and (3) a fully mobile ad hoc network without base stations. The authors provided a first

attempt to define a location-based group membership service. It is assumed that the network stays connected and there are no failures. Only changes due to mobility are considered. The architecture (Figure 6.7) proposed is composed of a *proximity layer* between the *group membership layer* and the underlying mobile network.

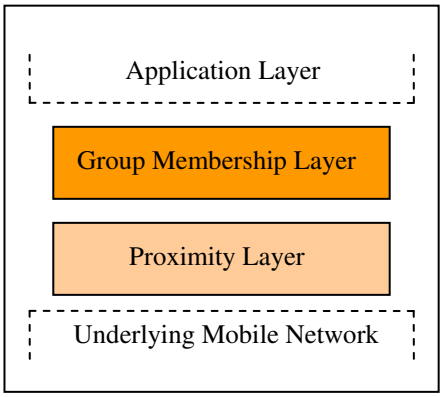

Figure 6.7: Prakash and Baldoni's approach architecture.

The *proximity layer* links the group membership layer with the underlying medium access control (MAC) sublayer. The MAC sublayer of the mobile network provides point-to-point communication and periodic beacons within transmission range d. Then, a $D-proximity$ test tries to find all nodes within distance D from a given node p. If $D \leq d$ then a node p determines its $D-proximity$ set of nodes by just listening to the *location stamped* beacon messages. For $D > d$, a multi-round synchronous algorithm is executed. The algorithm relies on resource-consuming flooding of discovery messages.

The group membership layer works on top of the proximity layer. There, a process p identifies all group members in $D-proximity$. The authors used a three round protocol derived from [77] to solve the group construction task in virtual cellular and ah-doc networks. First, p sends REQUEST messages to all nodes in its $D-proximity$. This request may contain further constraints on the prospective group members, such that not all nodes in $D-proximity$ can join the group. In the second round a node can acknowledge the join request, and in this case p sends to the node a confirmation of its group membership.

Notice that the construction of groups takes into account only the physical proximity of mobile nodes, thus, ignoring application specific needs.

AGAPE

AGAPE [61, 62, 63] is a location-aware group membership middleware, for collaborative applications in pervasive computing, that bases group management decisions on the location of users/devices. The underlying assumption is that users located in the same network locality are likely to require reciprocal collaboration more frequently than with members located in different physical network areas. So, instead of a global

view on group membership, location-dependent local views are considered. If for some application, the global group membership view is needed, it can be obtained by merging the local views. Two mobile network models are considered: cell and virtual cell models. The network consists of cells, each cell contains a base station which manages local group membership views in its cell. In particular, it processes join and leave requests, and detects network partitions.

AGAPE is centered around the domain abstraction. An AGAPE domain is mapped over one single network locality (*cell*), but a single locality may contain different domains. In addition, AGAPE domains are hierarchically organized implementing an $n-nary$ tree in order to achieve scalability (Figure 6.8). A group is defined by the set of its constituent domains found on different localities.

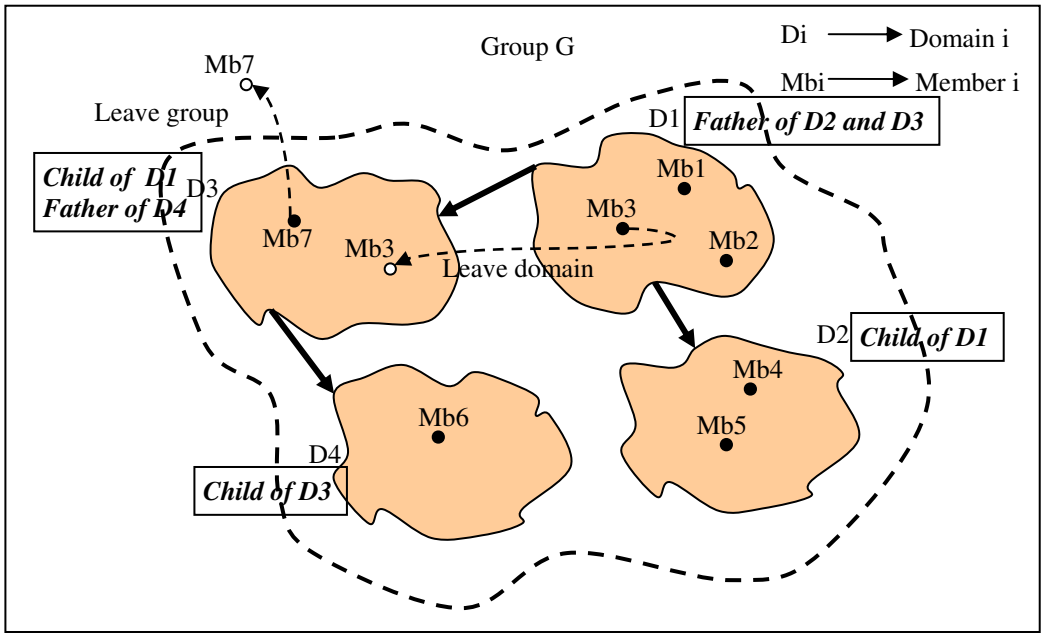

Figure 6.8: An example of an AGAPE group hierarchy.

Two class of entities can be distinguished in each domain (Figure 6.9): *Domain Management Entity* (DME) which allows users to join/leave a group domain. The DME maintains an updated list of group members currently active within a specific domain. Coordination among DMEs is only required in the case where members of the same logical group are located in other domains in different localities. *Final Entity* (FE) which represents users/devices exploiting the group management support of AGAPE in their current network locality. When entering a locality (cell), one FE has visibility of which AGAPE group domains are locally available. To join a domain, FE must firstly contact the DME of the group domain of FE interest. The FE is, then, authenticated and authorized by the DME; if admitted into the group, the FE becomes a group member. The status of the FE remains valid until it explicitly requests to leave the group. In the moving of an FE, it can decide, when arriving at a new locality, either

to become an active member by immediately connecting to the group domain of its interest or a non active member by deferring the connection to the group domain until expressly required. FE interoperation is enabled only for active group members.

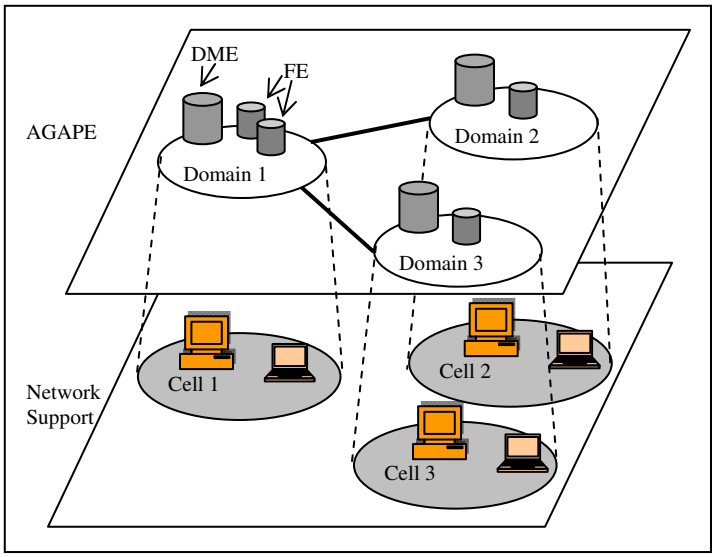

Figure 6.9: The AGAPE model in a wireless network scenario.

AGAPE provides a *context-aware* group communication service [63]. As key features, this service allows co-located group members to communicate via the following two basic communication patterns: (1) *context-based any-cast* communication pattern that enables unreliable and asynchronous point-to-point communication between two interoperating group members. When one group member has to communicate, one and only one co-located member entity that matches a specified profile is selected; (2) *context-based multi-cast* communication pattern that enables unreliable and asynchronous point-to-multipoint communication among various interoperating group members. In particular, the pattern permits to deliver the same message to all the co-located entities matching a desired profile.

6.7 Causal Ordering Relying on Group Communication

Several causal ordering protocols, supporting group communication, have been proposed. We distinguish those proposed as a communication primitive of a GCS and those providing primitives to deal with dynamic membership. In the first category, the *CBCAST* primitive of Isis [57] was perhaps the first implementation of (Reliable) Causal multicast. Other GCSs that provide this service level include: Transis, Ensemble, Horus, Newtop, and xAMp. As we have pointed out in Section 6.6, several efforts are done to re-think and re-design traditional group communication solutions in environments characterized by wireless connectivity and user/device mobility. However, none of these proposals, except for MGCM [84], provides a full GCS or supports causal

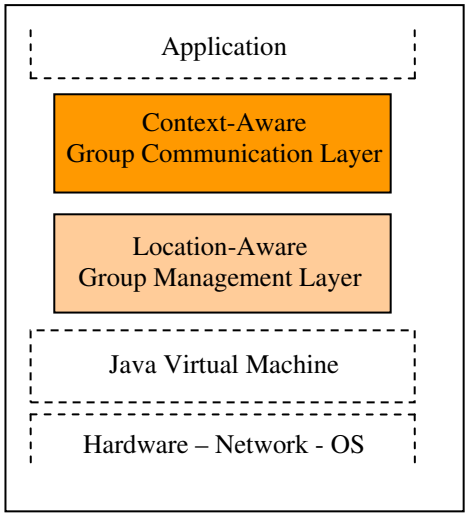

Figure 6.10: The AGAPE architecture.

ordering. Most of them focus on extending the group membership service to manage mobility-induced changes in group composition.

In the second category, several protocols have been devised either for static or mobile distributed systems. Some of these protocols include [35, 147, 170, 5, 37] for static distributed systems and [33, 80, 121, 60] for mobile systems. In [35, 34], authors considered a static group. *ACP* [147] is based on static group model with a *partial remove* of *mute* processes. That is, a mute process is no longer permitted to participate in the communication; it simply become listener. Authors force mute processes to rejoin the group in order to be allowed to participate again in the communication, but without explaining how this should be done. *SHT* [170] supports the fault-tolerant group communication, by replicating each process into a collection of multiple *replicas*, which is named a *cluster*. However, authors do not consider dynamic changes in group composition. Processes in *AN* [5] are organized into a logical, multilevel hierarchy: nodes are grouped into clusters, and clusters are organized into a tree, such that each cluster is assigned a father node in its parent cluster. Authors have discussed methods to deal with failures and network partitions. In *BFVR* [37], the participating processes are split into *local groups*, with use of causal servers to disseminate messages across these groups in the presence of processes faults. All causal servers are also members of *causal servers* group. The protocol rely on services provided by an existing GCS, like Horus [179], ISIS [56], etc. These services are failure detection and automatic reconfiguration in the event of a failure or a join of a new member, reliable fifo point-to-point delivery, stability detection, and automatic reissuing of messages from failed members that were received by only some members (but not by all of them). If a non-server member of the group fails, then the GCS takes care of resending all messages that originated by that member and were received by only part of the members of the same

local group. When a causal server fails, the local group it belongs to is reconfigured and another member is elected to be the causal server for that group. However, this server must be updated regarding the messages that were delivered in the servers group, but have not made it to the local group when the previous causal server crashed. Thus, whenever a causal server fails, all messages that were still not reported by it as stable in its local group are recorded. When the new causal server joins the servers group, a state update message is sent to it, which includes copies of all such messages. The new causal server can then forward these messages in its local group. Overlapping groups are addressed in [165, 143, 100]. The protocol proposed in [143] deals with static overlapping groups. While, [100] presented three causal ordering protocols for overlapping dynamic groups which use mechanisms proposed in [125] to cope with dynamic changes in group membership.

In the context of mobile environments, we find *BAD* [33, 34] and *MOCAVI* [80] protocols which assume a static group model. The *LH* [121] protocol proposes a centralized solution to deal with group changes where all join and leave requests must be treated by an *administrator site*. We know that the drawback of a centralized solution is the overloading of this administrator site which can become a bottleneck and also its crash which can cause the failure of the system. In [60], authors proposed an algorithm for dealing with dynamic group changes due to an MH changing its multicast groups or going offline/online.

6.8 Conclusion

In this Chapter, we have touched upon some aspects in group communication to motivate our remaining contribution (See the next Chapter). We specifically focused on group management solutions and the respect of causal consistency in the presence of dynamic group changes.

After giving the basic backgroud on group communication paradigm, we reviewed existing proposals devised for both static and mobile celullar networks; we also explained the challenges that arise in mobile environments. We also discussed how the existing group communication solutions and causal ordering protocols support each other.

Chapter 7

Group View Management

7.1 Introduction

In Chapter 4 and Chapter 5, we discussed the design of a causal multicast and broadcast protocols within a static group. However, as we have stated in Chapter 6, it is desirable to a group-based application to support the dynamic group membership changes. Changes occur when hosts voluntarily join or leave the group or when they crash.

Thus, this chapter is devoted to the extension of the proposed protocols to support dynamic groups while guaranteeing causal consistency. By our failure assumption, the membership management only processes the voluntary requests to join and leave the group.

In this chapter, we provide a detailed description of the algorithm required for the join and leave requests. The novelty of our contribution is that our group view management procedure is based on the idea of considering the leave and join requests as *data messages*. Then, a control information is appended to these messages and they will be ordered with data messages. This approach has the great benefit of rendering the installation of a new view fully decentralized. The membership algorithm presented here is used in conjunction with the message ordering algorithm.

Part of the results presented in this chapter have been published in [45, 44, 46].

7.2 Group's view construction

As said earlier, the group's view is defined by two components:

- MH_{View}: The set of MHs that are currently members of the group, and
- MSS_{View}, the set of $MSSs$ such that at least one MH from MH_{View} is local to every MSS in this set.

An MH is added to the view of a group (i.e. to the MH_{View} list) if it has explicitly expressed a request to join the group, and its MSS is added to MSS_{View} list if it does not yet exist. In the same way, an MH is removed from the view of a group (i.e.

from the MH_{View} list) if it has explicitly expressed a request to leave the group, and its MSS is deleted from the MSS_{View} list if this MH is the latest member localized in this MSS's cell. We assume that only voluntary disconnections (i.e. expressed by group members) can occur, in other words, the failures are not considered.

Furthermore, MSS_{View} may also change due to mobility of individual members although MH_{View} remains invariant. So, an MSS is added to the MSS_{View} list when a group member (an MH) moves to its cell such that no other MH in MH_{View} is local to this cell, and it must be removed from this list if the last member located in its cell migrates to another cell.

For simplicity and without lost of generality, we consider the existence of only one group.

7.3 Join/Leave message

This message (i.e. *join* or *leave* message) is considered as a *data* message and it will be ordered with them. If each host receives all *data* messages which precede the *join* or *leave* message, then this host installs the new view else, the installation of the view must be delayed until the delivery of these messages. So, the *virtual synchrony* semantics [57, 72] are ensured.

7.4 Join procedure

A mobile host h_i, wishing to join the group, sends a request $join(h_i)$ to its MSS S_i. At the reception of this request, S_i broadcasts a $Collect()$ request to all $MSSs$ within the group in order to collect information about messages not yet delivered to all group members. These messages are called *unstable messages*. This collection of information about unstable messages ensures that *a message sent in the old view must be delivered in the old view.*

23 On the receipt of $join(h_i)$ **by** S_i **from** h_i

1. $Boradcast(Collect())$ to all $S_j \in MSS_{view}$;
2. $nb_{Reply} := |MSS_{view}|$;

Upon the reception of this request, each MSS responds by a message $Reply(Unst_{Mess})$ which contains the list of unstable messages' identifiers sent by this MSS.

Case *MMobi_Causal:*

24 On the receipt of $Collect()$ **by** $S_j \in MSS_{view}$ **from** S_i

1. $Unst_{Mess} := \emptyset$;
2. **for all** $(IdSend_m, nb_{ack}) \in Send_{M_{S_i}}$ **do**
3. $\quad Unst_{Mess} := Unst_{Mess} \cup \{(IdSend_m, Dest_m)\}$;
4. **end for**
5. $Send(Reply(Unst_{Mess}))$ to S_i;

Case *BMobi_Causal:*

25 On the receipt of $Collect()$ **by** $S_j \in MSS_{view}$ **from** S_i

1. $Unst_{Mess} := \emptyset$;
2. **for all** $(IdSend_m, nb_{ack}) \in Send_{M_{S_i}}$ **do**
3. $\quad Unst_{Mess} := Unst_{Mess} \cup \{IdSend_m\}$;
4. **end for**
5. $Send(Reply(Unst_{Mess}))$ to S_i;

From this information, S_i calculates the control information to be attached to the *Install* request and broadcasts this last to all group members. The control information is represented by a global clock ϕ.

26 On the receipt of $Reply(Unst_{MessS_j})$ **by** S_i **from** $S_j \in MSS_{view}$

1. $\phi_{Install}[j] := [Unst_{MessS_j}]$;
2. $nb_{Reply} := nb_{Reply} - 1$;
3. **if** $(nb_{Reply} = 0)$ **then**
4. $\quad IdSend_{S_i} := IdSend_{S_i} + 1$;
5. $\quad Broadcast(Install, IdSend_{S_i}, \phi_{Install})$ to all $S_j \in MSS_{view}$;
6. $\quad id_{predh_i} := (IdSend_{S_i}, S_i)$;
7. $\quad Install(V_x, h_i, (J, h_i, S_i))$
8. **end if**

Case *MMobi_Causal:*

One entry $\phi[i] = [(id_{MUnst_1}, Dest_1)(id_{MUnst_2}, Dest_2), ...]$ corresponds to the set of unstable messages' identifiers and their destinations, sent by the MSS i. Upon receipt of the message ($Install$, $id_{Install}$, $\phi_{Install}$) by an MSS S_j, it verifies for each group's member h_j located in its cell, its delivery condition: $\forall(id_{MUnst}, Dest) \in \phi_{Install}[k]$ and $h_j \in Dest, \exists(S_k, SD_k) \in LastRcv_{h_j}$ such as $id_{MUnst} \in SD_k$.

27 On the receipt of $(Install, id_{Install}, \phi_{Install})$ **by** S_j **from** S_i

1. **for all** $h_j \in MH_{view}$ located in S_j's cell **do**
2. **if** $\forall k = \overline{1,n}$: $\forall (id_{MUnst}, Dest) \in \phi_{Install}[k] \mid h_j \in Dest$, $\exists (S_k, SD_k) \in LastRcv_{h_j}$ such that $id_{MUnst} \in SD_k$ **then**
3. $Install(V_x, h_j, (J, h_i, S_i))$;
4. $\phi_{h_j}[i] := (id_{Install}, MH_{view})$
5. $\forall l = \overline{1,n} \mid l \neq i, \phi_{h_j}[l] := 0$;
6. **else**
7. $AtFile_{h_j} := AtFile_{h_j} \cup \{(Install, id_{Install}, \phi_{Install})\}$;
8. **end if**
9. **end for**

Case *BMobi_Causal:*

One entry $\phi[k] = [id_{MUnst1}, id_{MUnst2}, ...]$ corresponds to the set of unstable message's identifiers sent by the MSS S_k. At the reception of the message $(Install, id_{Install}, \phi_{Install})$ by an MSS S_j, it verifies for each group member h_j located in its cell, the following delivery condition: $forall\ id_{MUnst} \in \phi_{Install}[k]$, $\exists (S_k, SD_k) \in LastRcv_{h_j}$ such as $id_{MUnst} \in SD_k$.

28 On the receipt of $(Install, id_{Install}, \phi_{Install})$ **by** S_j **from** S_i

1. **for all** $h_j \in MH_{view}$ located in S_j's cell **do**
2. **if** $\forall k = \overline{1,n}$: $\forall id_{MUnst} \in \phi_{Install}[k]$, $\exists (S_k, SD_k) \in LastRcv_{h_j}$ such that $id_{MUnst} \in SD_k$ **then**
3. $Install(V_x, h_j, (J, h_i, S_i))$;
4. $id_{predh_j} := (id_{Install}, S_i)$;
5. **else**
6. $AtFile_{h_j} := AtFile_{h_j} \cup \{(Install, id_{Install}, \phi_{Install})\}$;
7. **end if**
8. **end for**

We can easily notice that the control information attached to an *Install* message differs from that attached to a *Data* message. Thus, the entry "**On** receipt of $(Ack(IdSend_m, S_i))$ **by** S_j **from** h_j", of *BMobi_Causal* protocol, has to be extended to handle these messages:

29 On receipt of $(Ack(IdSend_m, S_i))$ **by** S_j **from** h_j

1. Send$(Ack(IdSend_m))$ to S_i;
2. $id_{predh_j} := (IdSend_m, S_i)$;
3. $LastRcv_{h_j}.SD_i := LastRcv_{h_j}.SD_i \cup (IdSend_m, IdSend_m)$; {Manage the dependency sequences as in [152]}
4. **for all** $m' \in AtFile_{h_j}$ **do**
5. **if** m' is an *install* message **then**
6. **if** $(\forall k = \overline{1,n} : \forall\ idf_l \in \phi_{m'}[k],\ idf_l \in LastRcv_{h_j}.SD_k)$ **then**
7. Deliver(m') to h_j
8. $AtFile_{h_j} := AtFile_{h_j} - (m', IdSend_{m'}, \phi_{m'})$
9. **end if**
10. **else**
11. {m' is a data message}
12. **if** $(id_{pred_{m'}} = (IdSend_m, S_i))$ **then**
13. Deliver(m') to h_j
14. $AtFile_{h_j} := AtFile_{h_j} - (m', IdSend_{m'}, id_{pred_{m'}})$
15. **end if**
16. **end if**
17. **end for**

30 Procedure Install$(V_x, h_j, (Type, h_i, S_i))$

1. $V_{h_j} := V_x$;
2. **if** $(Type = J)$ **then**
3. $MH_{view_{h_j}} := MH_{view_{h_j}} \cup \{h_i\}$;
4. **if** $(S_i \notin MSS_{view_{h_j}})$ **then**
5. $MSS_{view_{h_j}} := MSS_{view_{h_j}} \cup \{S_i\}$;
6. **end if**
7. **else**
8. **if** $(Type = L)$ **then**
9. $MH_{view_{h_j}} := MH_{view_{h_j}} - \{h_i\}$;
10. **if** (h_i is the last member located in the S_i's cell) **then**
11. $MSS_{view_{h_j}} := MSS_{view_{h_j}} - \{S_i\}$;
12. **end if**
13. **end if**
14. **end if**

If the delivery condition is verified which means that all unstable messages are delivered to this MH, h_j installs the new view V_x (where x represents the number of the installed view) and $id_{Install}$ is added to $LastRcv_{h_j}.SD_i$. Otherwise, the installation of the view for this member will be delayed until the delivery of all unstable messages destined to this member.

Installing the view is expressed by the addition of the identity of h_i to the set MH_{View}. On the other hand, if the delivery condition does not hold, which means that h_j has not received all unstable messages, then the installation of the view for this member will be delayed until the reception of all these unstable messages and the

message $Install()$ will be queued in $AtFile_{h_j}$.

The first message sent by the member h_j after the installation of the new view will be stamped by $\phi_{h_j}[i] := (id_{Install}, MH_{view})$ and $\phi_{h_j}[l] := 0, \forall\, l = \overline{1,n} \mid l \neq i$ in the case of $MMobi\_Causal$ and $(id_{Install}, S_i)$ in the case of $BMobi\_Causal$. This ensures that *future messages*; i.e. sent within the new view, *will not be delivered in an older view.*

When h_i joins the group, if no other MH in MH_{view} is local to S_i's cell, then S_i is added to MSS_{view}.

7.5 Leave procedure

The member h_i, wishing to leave the group, sends a request $Leave(h_i)$ to its MSS S_i. Upon receiving this request, S_i sends a request $Collect(LastRcv_{h_i})$ to all $MSSs$ within the group in order to collect information about unstable messages.

Contrary to the join procedure, we have added the $LastRcv_{h_i}$ set to the collection request in order to update information about intended acknowledgements from h_i: Since h_i will leave the group, than there is no need to await its acknowledgements of messages which has not yet received.

In this case, at the reception of the message $Collect(LastRcv_{h_i})$ by an MSS S_j,it verifies for each couple $(id_m, nb_{ack_m}) \in Send_{M_{S_j}}$

Case *MMobi_Causal:*

If $h_i \in id_m.Dest$ and $id_m \notin LastRcv_{h_i}.SD_j$ holds, then nb_{ack_m} will be decremented by 1. If nb_{ack_m} becomes null, the couple (id_m, nb_{ack_m}) will be deleted from $Send_{M_{S_j}}$ and the tuple (id_m, h_i) will be deleted from the entry $\phi[j]$.

Case *BMobi_Causal:*

If $id_m \notin LastRcv_{h_i}.SD_j$, then nb_{ack_m} is decremented by 1. If the nb_{ack_m} becomes null, the tuple will be removed from $Send_{M_{S_j}}$.

Thereafter, we shall proceed in the same way as in the case of the join procedure.

The installation of the view is expressed by the deletion of the h_i's identity from the set MH_{View}. If h_i is the last member located in the cell, then the MSS of this cell is removed from MSS_{view}.

7.6 Dynamic Mobile Group Specification

Let V_i^j to denote the i^{th} view of the group g installed by an MH h_j. The view V_i^j is defined by two components: $MH_{V_i^j}$ and $MSS_{V_i^j}$ which are respectively the set of MHs that are currently members of g and the set of MSSs where these MHs are located.

7.6.1 View properties

Property 7.1. *If an MH $h_j \in g$ and located in a cell covered by S_j, installs a view $V_i^j(g)$ then $h_j \in MH_{V_i^j}(g)$ and $S_j \in MSS_{V_i^j}(g)$.*

Proof. Modifications to the group membership (by modifying the two sets MH_{View} and MSS_{View}) are carried out only under the condition that both MH and its MSS which installs the new view on behalf of the MH belong to the new view. □

Property 7.2. *If an MH $h_j \in (MH_{V_i^k}(g) - MH_{V_{i-1}^k}(g))$, $i > 1$, then h_j asked to join g.*

Proof. If $h_j \in MH_{V_i^k}(g) - MH_{V_{i-1}^k}(g)$, $i > 1$, is because h_j was added to the group membership when the view $V_i^k(g)$ was installed. By definition, a new view is formed by modifying the two sets MH_{View} and MSS_{View}. At the installation of a new view, the set MH_{View} maintains identities of all MHs of the current view plus identities of those which have explicitly expressed a request to join the group. Therefore, if h_j is in $V_i^k(g)$, then h_j has necessarily expressed a request to join the group. □

Property 7.3. *If an MH $h_j \in (MH_{V_{i-1}^k}(g) - MH_{V_i^k}(g))$, $i > 1$, then h_j asked to leave g.*

Proof. If $h_j \in MH_{V_{i-1}^k}(g) - MH_{V_i^k}(g)$, $i > 1$, this means that h_j was removed from the group membership when the view $V_i^k(g)$ was installed. By definition, a new view is formed by modifying the two sets MH_{View} and MSS_{View}. At the installation of a new view, the set MH_{View} maintains identities of all MHs of the current view except identities of those which have explicitly expressed a request to leave the group. Therefore, if h_j is not in $V_i^k(g)$, then h_j has necessarily expressed a request to leave the group. □

Property 7.4. *If an MSS $S_j \in (MSS_{V_i^k}(g) - MSS_{V_{i-1}^k}(g))$, $i > 1$, then either some MH asked to join g from cell or to migrate to cell covered by S_j.*

Proof. If $S_j \in MSS_{V_i^k}(g) - MSS_{V_{i-1}^k}(g)$, $i > 1$, this means that S_j was added to the group membership when the view $V_i^k(g)$ was installed. By definition, a new view is formed by modifying the two sets MH_{View} and MSS_{View}. The identity of an MSS is added to the set MSS_{View} if there is at least one member of the group (i.e. an MH) located in its cell. At the installation of a new view, the set MSS_{View} maintains identities of all MSSs of the current view plus identities of those from which a new member is added to the group by expressing a join request or an existing member has migrated to their cells. So, if S_j is in the new view $V_i^k(g)$ then this is the fact that a new member has expressed a request to join the group from its cell or an existing member has migrated to its cell. □

Property 7.5. *If an MSS* $S_j \in (MSS_{V_{i-1}^k}(g) - MSS_{V_i^k}(g))$, $i > 1$, *then the last member in the cell covered by* S_j *asked to leave the group or migrated to another cell.*

Proof. If $S_j \in MSS_{V_{i-1}^k}(g) - MSS_{V_i^k}(g)$, $i > 1$, this means that S_j was removed from the group membership when the view $V_i^k(g)$ was installed. By definition, a new view is formed by modifying the two sets MH_{View} and MSS_{View}. The identity of an MSS is deleted from the set MSS_{View} if the latest member located in its cell has leaved the cell. At the installation of a new view, the set MSS_{View} maintains identities of all MSSs of the current view except identities of those where the latest member has leaved their cells. A member leaves a cell either because it has expressed a request to leave the group or because it has migrated from this cell to another cell. So, if S_j is not in $V_i^k(g)$ then this is the fact that the latest member located in its cell has leaved the group or migrated to another cell. □

Property 7.1 states that only the members of the group view install the corresponding view. *Property* 7.2, *Property* 7.3, *Property* 7.4 and *Property* 7.5 state that modifications of the two components of the group view are justified only by the join, leave and migration actions.

7.6.2 Multicast service properties

Lets *Dest* to denote the recipient members of the multicast message m.

Property 7.6 (Sending View Delivery)**.** *If a message* m *is delivered to an MH* $h_j \in$ *Dest in a view* V*, and some MH* h_i *multicasts* m *in a view* V'*, then* $V = V'$*.*

Proof. We suppose that a message m is delivered to $h_j \in Dest$ in a view V and its emission is accomplished in a different view V'. Since the emission of a message m precedes causally its delivery, then V is a new view installed in a previous view V'; which means that m is delivered to h_j after the installation of V. This is possible only if m is not an unstable message (i.e. $m \notin List_{MUnst}$)$\Rightarrow$ m is delivered to all recipient members of V' before the installation of V. If m is delivered to h_j, this means that h_j was one group member when the message was sent (i.e. $h_j \in MH_{V'}$)$\Rightarrow$ m is delivered to h_j in the same view of its emission. □

Property 7.7 (Same View Delivery)**.** *If a message* m *is delivered to both MHs* h_i *and* h_j*,* m *is delivered to them in the same view.*

Proof. The *Property* 7.6 ensures that a message is delivered to an MH in the same view of its emission (1). If a message is delivered to two MHs h_i and h_j, this means that h_i and h_j belong to the group membership when the message is sent (2). From (1) and (2), it follows that m is delivered to both MHs h_i and h_j in the same view. □

7.6.3 Broadcast service properties

Property 7.8 (Sending View Delivery)**.** *If a message m is delivered to an MH h_j in a view V, and some MH h_i broadcasts m in a view V', then $V = V'$.*

Proof. We suppose that a message m is delivered to h_j in a view V and its emission is accomplished in a different view V'. Since the emission of a message m precedes causally its delivery, then V is a new view installed in a previous view V'; which means that h_j has delivered m after the installation of V. This is possible only if m is not an unstable message (i.e. $m \notin List_{MUnst}$)$\Rightarrow$ m is delivered to all members of V' before the installation of V. If m is delivered to h_j, this means that h_j was one group member when the message was sent (i.e. $h_j \in MH_{V'}$)$\Rightarrow$ m is delivered to h_j in the same view of its emission. □

Property 7.9 (Same View Delivery)**.** *If a message m is delivered to both MHs h_i and h_j, m is delivered to them in the same view.*

Proof. The *Property* 7.8 ensures that a message is delivered to an MH in the same view of its emission (1). If a message is delivered to two MHs h_i and h_j, this means that h_i and h_j belong to the group membership when the message is sent (2). From (1) and (2), it follows that m is delivered to both MHs h_i and h_j in the same view. □

Property 7.10 (Virtual Synchrony)**.** *If two MHs h_i and h_j install the same view V in the same previous view V', then any message delivered to h_i in V' is also delivered to h_j in V'.*

Proof. Let h_i and h_j two members of a view V' (i.e. $\{h_i, h_j\} \in MH_{V'}$). We suppose that a message m is delivered to h_i in a view V' and that h_j has installed the new view V without delivering m. This is possible only if m is not an unstable message (i.e. $m \notin List_{MUnst}$)), which means that m has been delivered to all group members (i.e. to all members in $MH_{V'}$) before the installation of $V \Rightarrow m$ is delivered to h_j. Which leads to a contradiction with assumption: h_j has installed the new view V without delivering m. □

7.7 Discussion

The particularity of our group view management procedure resides in the consideration of the join and leave requests as data messages, and consequently will be ordered with them. This makes the installation of a view by a member constrained only by the delivery of all messages preceding the join or leave requests and has no need to a *coordination phase*. Thus, our solution is completely distributed unlike the *LH* [121] protocol which proposed a centralized solution to deal with group membership changes; that is, all join and leave requests are treated by an administrator site. Although a

centralized solution is easy to implement because all management functions are concentrated in the administrator site, this site constitutes the network bottleneck and a single point of failure which may prevent an application to scale to many users.

One important property guaranteed by our view management procedure is the virtual synchrony semantic. This semantic allows group members, that survive the same group changes, to know for certain that they have delivered exactly the same set of messages. By ensuring that processes deliver the same set of messages in each view, this allows them to maintain consistency across membership changes. However, achieving such a guarantee comes at the cost of the overhead of membership management and the number of messages exchanged to complete the view installation. As we have shown above, the installation procedure is composed of three phases: The *Collect Request*, in which the $Collect()$ message is diffused to all MSSs members of the group, the *Collect Reply*, in which information about unstable messages are collected from these MSSs, and the *Install Request*, in which the $Install()$ message is diffused to these MSSs after gathering information about unstable messages. So, the procedure results in the exchange of $O(3 * (|MSS_{view}| - 1))$ messages over the wired channels, if the hardware multicast is not supported. But, as we have assumed the support of hardware multicast capability, the communication complexity can be reduced to $O(2 + (|MSS_{view}| - 1))$.

Notice that the communication overhead for realizing the view installation does not depend on the number of MHs members of the group, but rather on the number of $MSSs$ within the group. Thus, we plausibly argue that our solution is scalable relatively to the number of MHs and thereby suitable for large groups. Moreover, the execution of the installation procedure is completely supported by the $MSSs$ when the unique role of MHs consists in sending the join/leave requests by the MH wanting to join or leave the group. This contributes to minimize the amount of computation performed by MHs and the communication overhead in wireless channels and makes the solution adequate for mobile environment.

In addition to consistency, another advantage of guaranteeing the virtual synchrony semantic is the possibility to purge data structures of the member installing the new view. This makes it possible to deal with the non bounded evolution of $LastRcv$ in the case of $MMobi\_Causal$ protocol. We compensate for this problem by resetting $LastRcv$ each time a new view is installed; upon the installation of a new view, keeping information about the delivery of messages sent in the old view (i.e. stable messages) is no longer required. We keep in this list only information about the delivery of view installation's message (i.e. $Install$ message).

7.8 Conclusion

The idea of ordering the join/leave requests with regular messages guarantees a causal consistence property that is of a great importance of applications do not needing strong

ordering (like total order) requirements for messages. Moreover, it makes the installation of a view fully decentralized and without need to a coordination phase which is a very important advantage, even more in mobile environments.

Chapter 8

Conclusion

Causal ordering concept has proven to be of a considerable interest for the design of distributed systems. This ordering property is attractive also in the context of mobile systems. In a mobile computing environment, causal ordering is especially important for applications that involve human interactions from several locations. Some of the major applications of distributed mobile systems in which causal ordering is useful are teleconferencing, stock trading, collaborative applications, etc. Nevertheless, this environment comes with new challenges: First, distributed processes in mobile hosts have mobility. Second, mobile hosts are inherently weaker in computing, storage and energy capabilities than fixed hosts. Third, wireless communication has less bandwidth and higher error rate compared to wired communication. Therefore, causal message ordering in this environment requires algorithms that take these factors into account.

The objective of this work has been to study the impact of mobile environment characteristics on the design of causal ordering protocols as well as the cost for causal delivery will typically pay off. In this thesis, we have examined the issues surrounding the design of an efficient causal ordering protocol for mobile environments with support of group concept.

8.1 Summary of Contributions

We have first examined the related literature. The main result was the identification of a set of criteria according to which causal ordering protocols can be classified, like the underlying network model, the mechanism used to construct the control information, the consideration of environment assumptions, the communication approach, application domain, support of group communication, support of faults, etc. We provide, to the best of our knowledge, the first taxonomy for causal ordering algorithms. We have also identified a number of key requirements that must be considered when designing causal ordering protocols, especially for cellular mobile environments. These requirements, like reduction of control information size, support for mobility, minimal participation by MHs in terms of computation, and communication over the wireless

link, and scalability serve as the basis for the design of an efficient causal ordering protocol for these environments.

By studying the related work, we have noticed that some proposals offer a better message overhead but incur an unnecessary inhibition delay in the delivery of messages, while some other solutions overcome this problem but by incurring high message overhead which can reaches $O(n_h^2)$. Moreover, most of them do not deal with group communication.

The emerging trend towards multicast applications in which there is a need to communicate with several other hosts simultaneously, drove us to make our first contribution. In this proposal, we tried to get benefit from the important characteristics of our unicast protocol [47, 43], such as elimination of unnecessary inhibition delay, low message overhead and scalability, in order to extend it to allow multicast communication among a static group of MHs and which can fill gaps of existing protocols, especially in terms of control information's size appended to each message. The proposed protocol (*MMobi_Causal*) retains all these interesting characteristics. It provides an optimal communication overhead without causing inhibition effect in the delivery of messages. This optimality is reached using the knowledge of *immediate dependency relationships* between messages and transmitting each message with only the relevant information to compliance with the causal order in the delivery of messages. The protocol is proved to satisfy the safety, the liveness, and the exactly once delivery properties.

Moreover, group-based applications often require that the exchanged messages within the group to be addressed to all processes in the group and not only to a subset of group's members. We refer to this type of communication as *broadcast communication*. So, our second contribution was based on our intuition that developing a protocol dedicated to causal broadcast should be more interesting than using a multicast protocol to ensure this broadcast, especially in terms of *control information*'s size appended to each message. The proposed solution (*BMobi_Causal*) proves the advantage of developing a *dedicated* causal broadcast delivery protocol based on *process behavior's awareness*. The originality of our proposal lies on showing how the rather well known behavior of a process can influence the control information's size. We showed that by viewing the behavior of the execution in the sense that the reaction to occurring events is *instantaneous*, tracking the dependency between messages is possible by only keeping information about the message's *immediate predecessor*. Hence, causal broadcast delivery of messages can be easily achieved in a mobile group with a small message overhead. The proposed protocol keeps all the interesting characteristics of our unicast protocol (*Mobi_Causal*). It outperforms its counterparts with respect to communication overhead (its message overhead is $O(1)$). The protocol is also proved to satisfy the safety, the liveness, and the exactly once delivery properties.

The two proposed protocols (*MMobi_Causal* and *BMobi_Causal*) assume a *static*

group model; that is, a group in which group composition is known at system initialization and does not change in time. However, a static group has practical limitations. In practice, it is often desirable that processes join and leave groups at runtime. Removing a crashed process from the group can also be desirable. This requires the support of *dynamic* group communication. The key component of a dynamic group is the *group membership* service, which is responsible for adding and removing processes during the computation. Thus, our last contribution was devoted to the extension of these protocols to support dynamic groups while guaranteeing causal consistency. The novelty of our contribution lies in considering the leave and join requests as *data messages*. Then, a control information is appended to these messages and they will be ordered with data messages. The idea of ordering the join/leave requests with regular messages guarantees a causal consistence property that is of a great importance of applications do not needing strong ordering (like total order) requirements for messages. Moreover, it makes the installation of a view fully decentralized and without need to a coordination phase which is a very important advantage, even more in mobile environments. One important property guaranteed by our view management procedure is the *virtual synchrony* semantic. This semantic allows group members, that survive the same group changes, to know for certain that they have delivered exactly the same set of messages. By ensuring that processes deliver the same set of messages in each view, this allows them to maintain consistency across membership changes.

8.2 Extensions of the work

Possible extensions of this work are as follows:

- Although the complexity analysis and the theoretical proof of the proposed solutions have been done, a simulation about the quantitative improvements they offer over the related works is desirable to reinforce the formal evaluation. A simulation comparing these solutions with some related works is underway.
- Adapting the proposed protocols to suit to specific application needs. For instance, designing a causal ordering protocol for a real-time system should take into account the constraints imposed by the deadline on the data. Another adaptation consists in building Building other ordering properties (e.g. total ordering) by using the causal ordering protocols.
- Extending the proposed protocols to support other group communication aspects, namely overlapping groups, partition and merge events, proximity groups which involve the location aspect in the formation of groups.
- The proposed solutions assume that hosts and the communication network are reliable. So, the proposed solutions require extension to cope with failures.

- Although infrastructured mobile networks represent a major portion of current mobile systems architectures, it is now clear that a new trend favoring ad hoc mobile networks is rapidly emerging. Moreover, a new wave of sensor networks is just around the corner. Extending the proposed protocols to these environments must deal with the absence of base stations.

Bibliography

[1] Casimiro A. and P. Verissimo. Timing failure detection with a timely computing base. *In Proc. of the 3rd European Research Seminar on Advances in Distributed Systems (ERSADS'99)*, pages 23–28, Apr. 1999.

[2] A. Abouaissa, A. Benslimane, and M. Nami. A group communication model for distributed real-time causal delivery. *In Proc. of the IEEE Intl. Conf. On Computer Communications and Networks (ICCCN 1998)*, pages 918–925, Oct. 1998.

[3] A. Acharya and B. R. Badrinath. A framework for delivering multicast messages in networks with mobile hosts. *ACM/Baltzer Mobile Netw. and Applicat.*, 1(2):199–219, June 1996.

[4] F. Adelstein and M. Singhal. Real-time causal message ordering in multimedia systems. *In the Proc. of the 15th IEEE Intl. Conf. on Distrib. Compt. Syst.*, pages 36–43, Jun. 1995.

[5] N. Adly and M. Nagi. Maintaining causal order in large scale distributed systems using a logical hierarchy. *In Proc. of The 13th IASTED Intl. Conf. on Applied Informatics*, pages 214–219, Jan. 1995.

[6] D. A. Agarwal. Totem: A reliable ordered delivery protocol for interconnected local-area networks. *Ph.D. Thesis, Dept. of Electrical and Computer Engineering, University of California, Santa Barbara*, 1994.

[7] M. Ahamad, P. Hutto, and R. John. Implementing and programming causal distributed memory. *In Proc. of the 11th Intl. Conf. on Distrib. Comput. Syst.*, page 274281, 1991.

[8] M. Ahamad, G. Neiger, J.E. Burns, P. Kohli, and P.W. Hutt. Causal memory: Definitions, implementation, and programming. *Distributed Computing*, 9(1):37–50, 1995.

[9] J. Ahn. New causal message logging protocol with asynchronous checkpointing for distributed systems. *GESTS Intl Trans. Computer Science and Engr.*, 19(1):7384, Oct. 2005.

[10] J. Ahn. On tolerating failures of mobile hosts and mobile support stations. *IJCSNS, Intl. Journal of Computer Science and Network Security*, 7(6):99–106, Jun. 2007.

[11] J. Ahn, S. Min, and C. Hwang. A causal message logging protocol for mobile nodes in mobile computing systems. *Future Generation Computer Systems*, 20:663–686, 2004.

[12] R. Aiello, E. Pagani, and G. P. Rossi. Causal ordering in reliable group communication. *Computer Communication Review*, 23(4):106–115, Oct. 1993.

[13] A. Alagar and S. Venkatesan. An optimal algorithm for distributed snapshots with causal message ordering. *Information Processing Letters*, 50:311–316, 1994.

[14] S. Alagar and S. Venkatesan. Causally ordered message delivery in mobile systems. *In Proc. Workshop on Mobile Computing Syst. and Applicat.*, pages 169–174, Dec. 1994.

[15] L. Alvisi, K. Bhatia, and K. Marzullo. Causality tracking in causal message-logging protocols. *Distrib. Comput.*, 15:1–15, 2002.

[16] L. Alvisi, B. Hoppe, and K. Marzullo. Non-blocking and orphan-free message logging protocols. *In Proc. Of the 23rd Fault-Tolerant Computing Symposium*, pages 145–154, June 1993.

[17] L. Alvisi and K. Marzull. Message logging: Pessimistic, optimistic, and causal. *In Proc. of the 15th IEEE Intl. Conf. on Distributed Computing Systems (ICDCS'95)*, pages 229–236, 1995.

[18] L. Alvisi and K. Marzull. Message logging: Pessimistic, optimistic, causal and optimal. *IEEE Trans. on Software Engineering*, 24(2):149–159, Feb. 1998.

[19] Y. Amir, C. Danilov, and J. Stanton. A low latency, loss tolerant architecture and protocol for wide area group communication. *In Proc. of the IEEE Intl. Conf. on Dependable Systems and Networks*, pages 327–336, Jun. 2000.

[20] Y. Amir, D. Dolev, S. Kramer, and D. Malki. Membership algorithms for multicast communication groups. *In the Proc. of the 6th Intl. Workshop on Distrib. Algo.*, pages 292–312, Nov. 1992.

[21] Y. Amir, D. Dolev, S. Kramer, and D. Malki. Transis: A communication subsystem for high availability. *In the Proc. of the 22nd Annual Intl. Symposium on Fault Tolerant Comput.*, pages 76–84, Jul. 1992.

[22] Y. Amir, L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, and P. Ciarfella. Fast message ordering and membership using a logical token-passing ring. *In the Proc. of the IEEE 13th Intl. Conf. on Distrib. Comput. Syst.*, pages 551–560, May 1993.

[23] Y. Amir, L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, and P. Ciarfella. The totem single-ring ordering and membership protocol. *ACM Trans. on Computer Syst.*, 13(4):311342, Nov. 1995.

[24] Y. Amir, C. Nita-Rotaru, J. Stanton, and G. Tsudik. Secure spread: An integrated architecture for secure group communication. *IEEE Trans. Dependable Secur. Comput.*, 2(3):248261, 2005.

[25] G. Anastasi, A. Bartoli, and F. Spadoni. A reliable multicast protocol for distributed mobile systems: Design and evaluation. *IEEE Trans. Parallel Distrib. Syst.*, 12(10):1009–1022, Oct. 2001.

[26] E. Anceaume, B. Charron-Bost, P. Minet, and S. Toueg. On the formal specification of group membership services. *Tech. Report 95-1534, Cornell Univ., Ithaca, USA*, 1995.

[27] T. Anker, G. V. Chockler, D. Dolev, and I. Keidar. Scalable group membership services for novel applications. *In Proc. of DIMACS Workshop on Networks in Distributed Computing*, 45:23–42, 1998.

[28] H. Attiya and J. Welch. *Distributed Computing.* second ed., Wiley, 2004.

[29] O. Babaoglu, A. Bartoli, and G. Dini. Enriched view synchrony: A programming paradigm for partitionable asynchronous distributed systems. *IEEE Transactions on Computers*, 46(6):642658, 1997.

[30] O. Babaoglu, R. Davoli, L.-A. Giachini, and M.G. Baker. Relacs: A communication infrastructure for constructing reliable applications in large-scale distributed systems. *In Proc. of the 28th Hawaii Intl. Conf. on System Sciences, Maui, Hawaii*, pages 612–621, Jan. 1995.

[31] O. Babaoglu, R. Davoli, and A. Montresor. Group communication in partitionable systems: Specification and algorithms. *IEEE Transactions on Software Engineering*, 27(4):308336, Apr. 2001.

[32] O. Babaoglu and A. Schiper. On group communication in large-scale distributed systems. *ACM SIGOPS Operating Systems Review*, 29(1):62–67, Jan. 1995.

[33] N. Badache. A causal broadcast protocol for mobile computing environments. *Foundations of Computing and Decision Sciences*, 22(4):251–265, 1997.

[34] N. Badache. *Ordre Causal et Tolérance aux Défaillances en Environnement Mobile*. Thèse de Doctorat, Institut d'Informatique, USTHB, 1998.

[35] N. Badache and A. Maddi. Gradual design of a causal broadcast protocol. *Parallel Processing Letters*, 7(3):309–320, 1997.

[36] R. Baldoni. A positive acknowledgement protocol for causal broadcasting. *IEEE Transactions on Computers*, 47(12):1341–1350, 1998.

[37] R. Baldoni, R. Friedman, and R. Van Renesse. The hierarchical daisy architecture for causal delivery. *In Proc. of the 17th Intl. Conf. on Distrib. Compt. Syst. (ICDCS'97)*, page 570, May 1997.

[38] R. Baldoni, M. Malek, A. Milani, and S.Tucci Piergiovanni. Weaklypersistent causal object in dynamic distributed systems. *In Proc. of the 25th IEEE Symposium On Reliable Distributed Systems (SRDS)*, Oct. 2006.

[39] R. Baldoni, A. Mostefaoui, and M. Raynal. Causal delivery of messages with real-time data in unreliable networks. *IJournal of Real-Time Systems*, 10(3):245–262, 1996.

[40] R. Baldoni, R. Prakash, M. Raynal, and M. Singhal. Efficient $\delta-$causal broadcasting. *Intl. Journal of Computer Systems Science and Engineering*, 13(5):263–269, Sep. 1998.

[41] A. Bartoli. Group-based multicast and dynamic membership in wireless networks with incomplete spatial coverage. *ACM/Baltzer Mobile Networking and Applications*, 3(2):175–188, Jun. 1998.

[42] A. Benslimane and A. Abouaissa. Dynamical grouping model for distributed real time causal ordering. *Computer Communications*, 25:288–302, 2002.

[43] C. Benzaid. A new protocol for causal message ordering in mobile computing systems with no inhibition delays. *In Proc. of the 7th Int. Symp. on Programming and Systems (ISPS'05)*, pages 59–68, May 2005.

[44] C. Benzaid and N. Badache. Bmobi_causal: A causal broadcast protocol in mobile dynamic groups. *In Proc. Of the 27th Annual ACM SIGACT-SIGOPS Symp. on Principles of Distrib. Comput. (PODC 2008)*, page 421, Aug. 2008.

[45] C. Benzaid and N. Badache. A causal multicast protocol for dynamic groups in cellular networks. *In Proc. of Euro American Conference on Telematics and Information Systems (EATIS 2008)*, Sep. 2008.

[46] C. Benzaid and N. Badache. An optimal causal broadcast protocol in mobile dynamic groups. *In Proc. Of IEEE Intl. Symp. on Parallel and Distributed Processing with Applications (ISPA'08)*, pages 477–484, Dec. 2008.

[47] Chafika Benzaid and Nadjib Badache. Mobi_causal: a protocol for causal message ordering in mobile computing systems. *SIGMOBILE Mobile Comput. Commun. Rev.*, 9(2):19–28, Apr. 2005.

[48] K. Berket, D. A. Agarwal, and O. Chevassut. A practical to the intergroup protocols. *Future Generation Computer Systems*, 18(5):709–719, Apr. 2002.

[49] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Distributed Database Systems.* Addison-Wesley, 1987.

[50] B. Bhargava and S. R. Lian. Independent checkpointing and concurrent rollback for recoveryan optimistic approach. *In Proc. of the Symposium on Reliable Distributed Systems*, page 312, 1988.

[51] K. Bhatia, K. Marzullo, and L. Alvisi. Scalable causal message logging for wide-area environments. 2002.

[52] K. Birman. A response to cheriton and skeen's criticism of causal and totally ordered communication. *SIGOPS Oper. Syst. Rev.*, 28(1):11–21, Jan. 1994.

[53] K. Birman. Building secure and reliable network applications. *Manning Publishing and Prentice Hall*, 1997.

[54] K. Birman. *Reliable Distributed Systems: Technologies, Web Services, and Applications.* Springer, 2005.

[55] K. Birman, R. Cooper, T. Joseph, K. Marzullo, M. Makpangou, K. Kane, F. Schmuck, and M. Wood. The isis system manual. *Depart. of Computer Science, Cornell University*, Sept. 1990.

[56] K. Birman and T. Joseph. Exploiting virtual synchrony in distributed systems. *In the Proc. of the ACM Symposium on Operating Systems Principles*, pages 123–138, Nov. 1987.

[57] K. Birman and T. Joseph. Reliable communication in the presence of failures. *ACM Trans. on Comput. Syst.*, 5(1):47–76, Feb. 1987.

[58] K. Birman and T. Joseph. Exploiting replication in distributed systems. *Distributed Syst.*, pages 319–367, Jan. 1990.

[59] K. Birman and R. van Renesse. *Reliable Distributed Computing with the ISIS Toolkit.* IEEE Computer Society Press, 1994.

[60] S. Bittner. Ordering in mobile networks using integrated sequencers. *In Proc. of the 3rd Intl. Workshop on Wireless Information Systems (WIS 2004)*, pages 73–80, Apr. 2004.

[61] D. Bottazzi, A. Corradi, and R. Montanari. Agape : a location-aware group membership middleware for pervasive computing environments. *In Proc. of the 8th IEEE Intl. Symp. on Computers and Communications*, pages 1185–1192, 2003.

[62] D. Bottazzi, A. Corradi, and R. Montanari. Context-aware group communication in mobile ad-hoc networks. *In Proc. of the 2nd International Conf. on Wired/Wireless Internet Communications (WWIC'04), LNCS 2957*, pages 38–47, Jan. 2004.

[63] D. Bottazzi, A. Corradi, and R. Montanari. Context-awareness for impromptu collaboration in manets. *In Proc. of the 2nd IEEE Annual Conf. on Wireless On-demand Network Systems and Services (WONS'05)*, pages 16–25, Jan. 2005.

[64] W. Cai, B.S. Lee, and J. Zhou. Causal order delivery in a multicast environment: An improved algorithm. *Journal of Parallel and Distributed Computing*, 62(1):111–131, 2002.

[65] P. Chandra and A. D. Kshemkalyani. Causal multicast in mobile networks. *In Proc. of the the IEEE Comput. Soc.'s 12th Annu. intl. Symp. on Modeling, Anal., and Simulation of Comput. and Telecommun. Syst. (Mascots'04)*, pages 213–220, Oct. 2004.

[66] J. M. Chang and N. Maxemchuck. Reliable broadcast protocols. *ACM Trans. on Computer Systems*, 2(3):251–273, Aug. 1984.

[67] B. Charron-Bost. Concerning the size of logical clocks in distributed systems. *Information Processing Letters*, 39:1116, 1991.

[68] B. Charron-Bost, F. Mattern, and G. Tel. Synchronous, asynchronous, and causally ordered communications. *Distributed Computing*, 9(4):173–191, 1996.

[69] D. Cheriton and V. Zwaenepoe. Distributed process groups in the vkernel. *ACM Trans. on Computer Syst.*, 3(2):77–107, 1985.

[70] D. R. Cheriton and D. Skeen. Understanding the limitations of causally and totally ordered communication. *In Proc. of the 14th ACM Symposium on Operating Systems Principles (SoSP-14)*, page 4457, 1993.

[71] K. Chi, L. Yen, C. Tseng, and T. Huang. A causal multicast protocol for mobile distributed systems. *IEICE TRANS. INF. & SYST.*, E83-D(12):2065–2074, Dec. 2000.

[72] G. V. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: A comprehensive study. *ACM Computing Surveys (CSUR)*, 33(4), Dec. 2001.

[73] Collins Cobuild. *Advanced Learner's English Dictionary.* Collins CoBUILD, 5th edition, 2006.

[74] J. Collins, N. Hall, and L. A. Paul. *Causation and Counterfactuals.* MIT Press, Cambridge, MA, 2004.

[75] R. Cooper. Experience with causally andtotally ordered group communication support-a cautionary tale. *ACM Operating Systems Review*, 28(1):2832, Jan. 1994.

[76] F. Cristian. Probabilistic clock synchronization. *Distributed Computing*, 3(3):146158, 1989.

[77] F. Cristian and F. Schmuck. Agreeing on processor group membership in timed asynchronous distributed systems. *Tech. Rep. CSE95-428, University of California, San Diego*, 1995.

[78] R. De Prisco, A. Fekete, N. Lynch, and A. Shavartsman. A dynamic view-oriented group communication service. *In Proc. of the 17th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 227–236, Jun. 1998.

[79] X. Défago, A. Schiper, and P. Urbán. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Computing Surveys*, 36(4):1–50, Dec. 2004.

[80] E. L. Dominguez, S. E. P. Hernandez, and G. R. Gomez. Mocavi: An efficient causal protocol for cellular networks. *IJCSNS: Intl. Journal of Computer Science and Network Security*, 8(1):136–144, Jan. 2008.

[81] E. L. Dominguez, J. E. Ramirez, J. Fanchon, and S. E. Pomares Hernandez. A fault-tolerant causal broadcast algorithm to be applied to unreliable networks. *In Proc. of the International Conf. on Parallel and Distributed Computing Systems (PDCS'05)*, pages 465–470, Nov. 2005.

[82] R. Drummong and O. Babaoglu. Low-cost clock synchronization. *Distributed Computing*, 6(3):193203, July 1993.

[83] D. M. Eagleman and A. O. Holcombe. Causality and the perception of time. *TICS 6/8*, pages 323–325, 2002.

[84] M. A. El-Gendy, H. Baraka, and A. H. Fahmy. Migrating group communication protocols to networks with mobile hosts. *In Proc. of the IEEE 1998 Midwest Symposium on Systems and Circuits (MWSCAS'98)*, pages 74–77, Aug. 1998.

[85] E. Elnozahy. On the relevance of communication costs of rollback-recovery protocols. *In Proc. of the 15th ACM Symposium on Principles of Distributed Computing*, pages 74–79, 1995.

[86] E.N. Elnozahy and W. Zwaenepoe. Manetho: Transparent rollback-recovery with low overhead limited rollback and fast output commit. *IEEE Trans. On Computers*, 41(5):526–531, May 1992.

[87] A. Fekete and N. L. A. Shvartsman. Specifying and using a partitionable group communication service. *ACM Transactions on Computer Systems*, 19(2):171216, May 2001.

[88] A. Fernández, E. Jimdnéz, and V. Cholvi. On the interconnection of causal memory systems. *In the Proc. of the 19th ACM Symposium on Principles of Distrib. Comput.*, pages 163–170, 2000.

[89] A. Fernández, E. Jimdnéz, and V. Cholvi. On the interconnection of causal memory systems. *Journal of Parallel and Distributed Computing*, 64(4):498506, 2004.

[90] C. Fidge. Timestamps in message-passing systems that preserve the partial ordering. *In Proc. of the 11th Australian Computer Science Conf.*, page 5666, Feb. 1988.

[91] C. Fidge. Logical time in distributed computing systems. *IEEE Computer*, 24(8):28–33, Aug. 1991.

[92] R. Friedman and R. van Renesse. Strong and weak virtual synchrony in horus. *In Proc. of the 15th Symp. on Reliable Distrib. Syst. (SRDS 96)*, page 140, 1996.

[93] A. Gidenstam, B. Koldehofe, M. Papatriantafilou, and P. Tsigas. Lightweight causal cluster consistency. *Lecture Notes in Computer Science*, pages 17–28, 2005.

[94] V. Hadzilacos and S. Toueg. Fault-tolerant broadcasts and related problems. *Distributed systems (2nd Ed.), ISBN 0-201-62427-3*, pages 97–145, 1993.

[95] V. Hadzilacos and S. Toueg. A modular approach to fault tolerant broadcasts and related problems. *Technical Report TR94-1425, Cornell University, Computer Science Department*, May 1994.

[96] M. Hiltunen and R. Schlichting. Properties of membership services. *In Proc. of the 2nd IEEE Intl. Symp. on Autonomous Decentralized Systems (ISADS'95)*, pages 200–207, Apr. 1995.

[97] J. Ioannidis, D. Duchamp, and G. Q. Maguire. Ipbased protocols for mobile internetworking. *In Proc. of ACM SIGCOMM Symposium on Communication, Architectures and Protocols*, page 235245, Sep. 1991.

[98] D. Janaki Ram, M. Uma Mahesh, N. S. K. Chandra Sekhar, and C. Babu. Causal consistency in mobile environment. *Operating Systems Review*, 35(1):3440, 2001.

[99] D.B. Johnson and W. Zwaenepoel. Sender-based message logging. *In: Digest of Papers: 17th Annual Intl. Symp. on Fault-Tolerant Computing*, page 1419, Jun. 1987.

[100] R. José and A. Macêdo. Causal order protocols for group communication. *In the Proc. of the 8th Brazilian Symp. in Computer Networks (SBRC'95)*, pages 265–283, May 1995.

[101] F. M. Kaashoek, A. S. Tanenbaum, and K. Verstoep. Using group communication to implement a fault-tolerant directory service. *In the Proc. of the IEEE 13th Intl. Conf. on Distrib. Comput. Syst.*, pages 130–139, May 1993.

[102] F. M. Kaashoek, R. van Renesse, H. van Staveren, and A. S. Tanenbaum. Flip: an internetwork protocol for supporting distributed systems. *ACM Trans. on Computer Syst.*, 11(1):73–106, Feb. 1993.

[103] M. F. Kaashoek and A. S. Tanenbaum. An evaluation of the amoeba group communication system. *In Proc. of 16th IEEE Intl. Conf. on Distrib. Comput. Systems (ICDCS-16)*, page 436447, Jun. 1996.

[104] M. F. Kaashoek, A. S. Tanenbaum, S. F. Hummel, and H. E. Bal. An efficient reliable broadcast protocol. *Oper. Syst. Rev.*, 23(4):519, 1989.

[105] S. Kawanami, T. Enokido, and M. Takizawa. A group communication protocol for scalable causal ordering. *In Proc. Of the 18th Intl. Conf. on Advanced Information Networking and Application (AINA'04)*, pages 296–302, Mar. 2004.

[106] I. Keidar and R. Khazan. A client-server approach to virtually synchronous group multicast: Specifications and algorithms. *In Proc. of the 20th IEEE Intl. Conf. on Distributed Computing Systems*, pages 344–355, Apr. 2000.

[107] I. Keidar, J. Sussman, K. Marzullo, and D. Dolev. A client-server oriented algorithm for virtually synchronous group memebership in wans. *In Proc. of the 20th IEEE Intl. Conf. on Distributed Computing Systems*, pages 356–365, Apr. 2000.

[108] I. Keidar, J. Sussman, K. Marzullo, and D. Dolev. Moshe: A group membership service for wans. *ACM Transcations on Computer Systems*, 20(3):191–238, 2002.

[109] M.-O. Killijian, R. Cunningham, R. Meier, L. Mazare, and V. Cahill. Towards group communication for mobile participants. *In Proc. of Principles of Mobile Computing (POMC'01). Newport, Rhode Island, USA*, pages 75–82, 2001.

[110] B. Kim and E. Hong. Supporting mobile system in group communication.

[111] B. Kim, D. Lee, and D. Nam. Scalable group membership service for mobile internet. *In Proc. of the 7th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'02)*, pages 295–298, Jan. 2002.

[112] H. J. Kim, D. Lee, and H. Y. Toun. A scalable membership service for group communications in wans. *In Proc. of the IEEE 2000 Pacific Rim International Symp. On Dependable Computing (PRDC 2000)*, pages 59–68, Dec. 2000.

[113] A. D. Kshemkalyani and M. Singhal. Necessary and sufficient conditions on information for causal message ordering and their optimal implementation. *Distrib. Comput.*, 11(2):91–111, Apr. 1998.

[114] S. S. Kulkarni and M. U. Arumuga. Approximate causal observer. *In Proc. of the Intl. Workshop on Networked Sensing Systems (INSS)*, pages 123–128, Jun. 2004.

[115] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Commun. of the ACM*, 21(7):558–565, July 1978.

[116] L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers*, 28(9):690691, 1979.

[117] L. Lamport. On interprocess communication. part i: Basic formalism. *Distributed Computing*, 1(2):7785, 1986.

[118] N. Lavi, I. Cidon, and I. Keidar. Supporting groupware in mobile networks. *In proc. of the 6th IFIP/IEEE International Conf. on Mobile and Wireless Communication Networks (MWCN'04)*, pages 95–106, Oct. 2004.

[119] B. Lee, T. Park, H. Y. Yeom, and Y. Cho. An efficient algorithm for causal message logging. 1999.

[120] N. Lesley, H. Hou, and J. Mitra. Causal errors in distributed systems. *IEEE Distributed Systems Online*, 4(6), 2003.

[121] C. Li and T. Huang. A mobile-support-station-based causal multicast algorithm in mobile computing environment. *Proc. Nat. Sci. Council, ROC(A)*, 23(1):100–110, 1999.

[122] Q. Li and D. Rus. Global clock synchronization in sensor networks. *IEEE Transactions on Computers*, 55(2):214–226, Feb. 2006.

[123] R. Lipton and J. Sandberg. Pram: A scalable shared memory. *Tech. Report CS-TR-180-88, Princeton University*, 1988.

[124] C. H. Lwin, H. Mohanty, and R. K. Ghosh. Causal ordering in event notification service systems for mobile users. *In Proc. of the Intl. Conf. on Information Technology: Coding and Computing (ITCC04) Volume 2*, page 735, 2004.

[125] R. A. Macedo. Fault-tolerant group communication protocols for asynchronous systems. *Ph.D. Thesis, Dept. of Computer Science, University of Newcastle Upon Tyne*, 1994.

[126] R. A. Macedo, P. Ezhilchlvan, and S. K. Shrivastava. Newtop: a total order multicast protocol using causal blocks. *BROADCAST project deliverable report*, I, Oct. 1993.

[127] D. Malki. Multicast communication for high availability. *Ph.D. Thesis, Institute of Computer Science, The Hebrew University of Jerusalem, Israel*, 1994.

[128] F. Mattern. Virtual time and global states of distributed systems. *In M. C. et al., editor, Proc. of the Intl. Workshop on Parallel and Distrib. Algorithms*, page 215226, Dec. 1988.

[129] F. Mattern and S. Funfrocken. A non-blocking lightweight implementation of causal order message delivery. *In: Ken Birman, Friedemann Mattern, Andre Schiper (Eds.): Theory and Practice in Distributed Systems, Springer-Verlag LNCS 938*, pages 197–213, 1995.

[130] R. Meier. Communication paradigms for mobile computing. *Mobile Computing and Communications Review*, 6(4):56–58, Oct. 2002.

[131] P. M. Melliar-Smith, Moser L. E., and V. Agrawala. Broadcast protocols for distributed systems. *IEEE Trans. on Parallel and Distrib. Syst.*, 1(1):17–25, Jan. 1990.

[132] P. M. Melliar-Smith and L. E. Moser. Trans: A reliable broadcast protocol. *IEEE Trans. on Communications*, 140(6):481–493, Dec. 1993.

[133] A. Michotte. *The Perception of Causality.* Basic Books, New York, 1946.

[134] D. Mills. Internet time synchronization: the network time protocol. *IEEE Trans. on Communication*, 39(10):1482–1493, 1991.

[135] D. L. Mills. Internet time synchronization: the network time protocol. *IEEE Transactions on Communications*, 39(10):14821493, Oct. 1991.

[136] S. Mishra, L. L. Peterson, and R. D. Schlichting. A membership protocol based on partial order. *In the Proc. of the Intl. Working Conf. on Dependable Comput. for Critical App.*, pages 309–331, Feb. 1991.

[137] A. Montresor. The jgroup reliable distributed object model. *In Proc. of the 2nd IFIP Intl Working Conf. on Distrib. Applications and Interoperable Systems (DAIS'99). Helsinki, Finland*, pages 389–402, Jun. 1999.

[138] A. Montresor. A reliable registry for the jgroup distributed object model. *In Proc. of the 3rd European Research Seminar on Advances in Distributed Systems (ERSADS'99). Madeira, Portugal*, Apr. 1999.

[139] L. E. Moser, Y. Amir, P. M. MelliarSmith, and D. A. Agarwal. Extended virtual synchrony. *In Proc. of the 14th IEEE Intl. Conf. on Distributed Computing Systems*, page 5665, Jun. 1994.

[140] L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia, and C.A. Lingley-Papadopoulos. Totem: A fault-tolerant multicast group communication system. *Communications of the ACM*, 39(4):5463, 1996.

[141] L. E. Moser, P. M. Melliar-Smith, and V. Agarwala. Processor membership in asynchronous distributed systems. *IEEE Trans. on Parallel and Distrib. Syst.*, 5(5):459–473, May 1994.

[142] L. E. Moser, P. M. Melliar-Smith, and V. Agrawala. Asynchronous fault-tolerant total ordering algorithms. *SIAM Journal of Computing*, 22(4):727–750, Aug. 1993.

[143] A. Mostefaoui and M. Raynal. Causal multicasts in overlapping groups: Towards a low cost approach. *Proc. Of the 4th IEEE Intl. Conf. of Future Trends in Distrib. Compt. Syst.*, pages 136–142, Sep. 1993.

[144] C. Ohori, M. Inoue, T. Masuzawa, and H. Fujiwar. A causal broadcast protocol for distributed mobile systems. *Systems and Computers in Japan*, 32(3):65–75, 2001.

[145] J. Pearl. *Causality: Models, Reasoning, and Inference.* Cambridge University Press, New York, 2000.

[146] L. L. Peterson, N. C. Buchholz, and R. D. Schlichting. Preserving and using context information in interprocess communication. *ACM Trans. on Computer Syst.*, 7(3):217–246, Aug. 1989.

[147] S. Pomares Hernandez, J. Fanchon, K. Drira, and M. Diaz. Causal broadcast protocol for very large group communication systems. *In Proc. of the 5th Intl. Conf. on Principles of Distributed Systems (OPODIS'01)*, pages 175–188, Dec. 2001.

[148] D. Powell. Delta-4 - a generic architecture for dependable distributed computing. *ESPRIT Research Reports, Springer Verlag*, pages 267–294, Nov. 1991.

[149] R. Prakash and R. Baldoni. Architecture for group communication in mobile systems. *In Proc. of the 17th IEEE Symposium on Reliable Distributed Systems. Indiana, USA*, pages 235–242, Oct. 1998.

[150] R Prakash, M Raynal, and M Singhal. An efficient causal ordering algorithm for mobile computing environments. *In Proc. of the 16th intl. Conference on Distributed Computing Syst. (ICDCS '96)*, pages 744–751, May 27-30 1996.

[151] R. Prakash, M. Raynal, and M. Singhal. An adaptive causal ordering algorithm suited to mobile computing environments. *Journal of Parallel Distrib. Comput.*, 41(2):190–204, March 1997.

[152] R Prakash and M Singhal. Dependency sequences and hierarchical clocks: efficient alternatives to vector clocks for mobile computing systems. *Wireless Netw.*, 3(5):349–360, Oct. 1997.

[153] B. Rajagopalan and P. K. McKinle. A token-based protocol for reliable ordered multicast communication. *In the Proc. of the 8th IEEE Symposium on Reliable Distrib. Syst.*, pages 84–93, Oct. 1989.

[154] K. Ravindran and B. Prasad. Communication structures and paradigms for distributed conferencing applications. *In Proc. of the 12th Intl. Conf. on Distributed Computing Systems (ICDCS'92)*, pages 598–605, May 1992.

[155] M. Raynal and M. Ahamad. Exploiting write semantics in implementing partially replicated causal objects. *In Proc. of the 6th EUROMICRO Conf. on Parallel and Distrib. Comput.*, pages 157–163, Feb. 1998.

[156] M. Raynal, A. Schiper, and S. Toueg. The causal ordering abstraction and a simple way to implement it. *Inform. Process. Lett.*, 39(6):343–350, Oct. 1991.

[157] M. Reiter and L. Gong. Securing causal and relationships in distributed systems. *Computer Journal*, 38(8):633642, 1996.

[158] A. Rezgui and N. Badache. A causal multicast protocol based on reserved causality path. *Rapport Technique*, 1995.

[159] L. Rodrigues, R. Baldoni, E. Anceaume, and M. Raynal. Deadline-constrained causal order. *In Proc. of the Third IEEE Intl. Symp. on Object-oriented Real-time distrib. Comput. (ISORC 2000)*, Mar. 2000.

[160] L. Rodrigues and P. Verissimo. xamp: a multi-primitive group communication service. *In the Proc. of the 11th Symposium on Reliable Distrib. Syst.*, pages 112–121, Oct. 1992.

[161] L. Rodrigues, P. Verissimo, and J. Rufin. A low-level processor group membership protocol for lans. *In the Proc. of the 13th Intl. Conf. on Distrib. Comput. Syst.*, pages 541–550, May 1993.

[162] L. Rodrigues and P. Verssimo. Causal separators for large-scale multicast communication. *In Proc. Of 15th IEEE Intl. Conf. on Distrib. Compt. Syst.*, pages 83–91, May 1995.

[163] J. Roger Mitchell and V. K. Garg. A non-blocking recovery algorithm for causal message logging. 1997.

[164] A. Schiper. Dynamic group communication. *Distrib. Comput.*, 18(5):359374, 2006.

[165] A. Schiper, K. Birman, and P. Stephenson. Lightweight causal and atomic group multicast. *ACM Trans. Comput. Syst.*, 9(3):272–314, Aug. 1991.

[166] A. Schiper, J. Eggli, and A. Sandoz. A new algorithm to implement causal ordering. *Proc. of the 3rd Intl. Workshop on Distributed Algorithms, In Lecture Notes in Computer Science*, 392:219–232, Sept. 1989.

[167] R. D. Schlichting and F. B. Schneider. Fail-stop processors: an approach to designing fault-tolerant distributed computing systems. *ACM Transactions on Computer Systems*, 1(3):222–238, 1985.

[168] F. Schneider. Implementing fault-tolerant services unsing the state machine sproach: A tutorial. *ACM Compt. Surveys*, 22(4):299319, Dec. 1990.

[169] R. Schwarz and F. Mattern. Detecting causal relationships in distributed computations: In search of the holy grail. *Distrib. Comput.*, 7(3):149–174, 1994.

[170] K. Shima, H. Higaki, and M. Takizaw. Fault-tolerant causal delivery in group communication. *In Proc. of the Intl. Conf. on Parallel and Distrib. Syst. (ICPADS'96)*, page 302, 1996.

[171] C. Skawratananond, N. Mittal, and V. K. Garg. A lightweight algorithm for causal message ordering in mobile computing systems. *In Proc. of 12th ISCA Intl.*

Conference on Parallel and Distributed Computing Syst. (PDCS), pages 245–250, 1999.

[172] R. B. Strom and S. Yemeni. Optimistic recovery in distributed systems. *ACM Transactions on Computer Systems*, 3(3):204–226, 1985.

[173] J. Sussman, I. Keidar, and K. Marzullo. Optimistic virtual synchrony. *In Proc. of the 19th IEEE Symp. on Reliable Distrib. Syst. (SRDS00)*, page 42, 2000.

[174] F.J. Torres-Rojas and M. Ahamad. Plausible clocks: Constant size logical clocks for distributed systems. *Distributed Computing*, 12(4):179–196, 1999.

[175] S. J. Turner, W. Cai, and J. Chen. A middleware approach to causal order delivery in distributed simulations. *In Proc. of the 2003 European Simulation Interoperability Workshop (Euro-SIW)*, (03E-SIW-085).

[176] R. Van Renesse. Causal controversy at le mont st.-michel. *Oper. Syst. Rev.*, 27(2):4453, Apr. 1993.

[177] R. Van Renesse. Why bother with catocs? *Operating Systems Review*, 28(1):2227, Jan. 1994.

[178] R. van Renesse, K. Birman, R. Friedman, M. Hayden, and D. Karr. A framework for protocol composition in horus. *In Proc. of the 14th Annual ACM Symposium on Principles of Distrib. Comput. (PODC'95)*, pages 80–89, Aug. 1995.

[179] R. van Renesse, K.P. Birman, and S. Maffeis. Horus: A flexible group communication system. *Commun. ACM*, 39(4):7683, 1996.

[180] P. Verissimo. Causal delivery protocols in real-time systems: a generic model. *Real-Time Systems*, 10(1):45–73, Jan. 1996.

[181] P. Verissimo, L. Rodrigues, and A. Casimiro. Cesiumspray: A precise and accurate global time service for large-scale systems. *Real-Time Systems*, 12(3):243–294, May 1997.

[182] P. Watson and I. Watson. An efficient garbage collection scheme for parallel computer. *Parallel Architectures and Languages Europe*, pages 432–443, Jun. 1987.

[183] B. Whetten, T. Montgomery, and S. Kaplan. A high performance totally ordered multicast protocol. *In Selected Papers from the International Workwhop on Theory and Practice in Distributed Systems. Lecture Notes in Computer Science*, 938:33–57, 1994.

[184] R. Yavatkar. Mcp: A protocol for coordination and temporal synchronization in multimedia collaborative applications. *In Proc. of the 12th IEEE Intl. Conf. on Distributed Computing Systems*, pages 606–613, 1992.

[185] L. Yen, T. Huang, and S. Hwang. A protocol for causally ordered message delivery in mobile computing systems. *Mobile Network. Applicat.*, 2(4):365–372, Dec. 1997.

[186] L. H. Yen, K. H. Chi, and T. L. Huang. A scalable scheme for causal message ordering. *In Proc. of the National Computer Symposium (NCS'03)*, Dec. 2003.

[187] S. Zhou, W. Cai, S. J. Turner, and B. S. Lee. Critical causal order of events in distributed virtual environments. *ACM Trans. on Multimedia Computing, Commun. and Applica.*, 3(3), Aug. 2007.