An efficient causal group communication protocol for P2P hierarchical overlay networks

Grigory Evropeytsev, Eduardo López Domínguez, Saúl Eduardo Pomares Hernández, Marco Antonio López Trinidad, Jose Roberto Perez Cruz

An Efficient Causal Group Communication Protocol for P2P Hierarchical Overlay Networks

Grigory Evropeytsev

(National Laboratory of Advanced Informatics, Xalapa, Veracruz, Mexico

grigory88@live.com)

Eduardo López Domínguez

(National Laboratory of Advanced Informatics, Xalapa, Veracruz, Mexico

elopez@lania.mx)

Saul E. Pomares Hernandez

(Computer Science Department, Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), Tonantzintla, Puebla, Mexico

CNRS, LAAS, 7 avenue du Colonel Roche, F-31400 Toulouse, France

Univ de Toulouse, LAAS, F-31400 Toulouse, France

spomares@inaoep.mx)

Marco Antonio López Trinidad

(National Laboratory of Advanced Informatics, Xalapa, Veracruz, Mexico

mlopez@lania.mx)

José Roberto Perez Cruz

(CONACYT - Faculty of Civil Engineering, Universidad Michoacana de San Nicolas de Hidalgo (UMSNH), Morelia, Michoacan, Mexico

jrperezcr@conacyt.mx)

Abstract: Peer-to-peer applications such as multiplayer online games are characterized by considering group communication among geographically distributed peers. In such environments, causal ordering is an essential property for consistent exchange of information among peers. Although several works are oriented to ensure message causal order, most of them are not suitable for hierarchical overlay networks. In this paper, we propose an efficient causal protocol oriented to be used in a hierarchical overlay network. In our protocol the overhead timestamp per message is based on the number of messages with immediate dependency relation. By using the information about network architecture and representing message dependencies on a bit level, proposed protocol ensures causal message ordering without enforcing super peers order to all of the peers in a group. The protocol has been simulated and the results show that it presents lower overhead than currently existing causal protocols.

Keywords: Causal ordering, peer-to-peer, hierarchical topology

1 Introduction

Multiparty overlay systems like peer-to-peer have been proposed to solve problems related to distribution and processing of information in many applications, such as, file distribution [1, 2], multiplayer interactive games [3] and telecommunications [4]. These systems are characterized by considering a group communication among n (≥ 2) peers that are geographically distributed. To achieve a consistent exchange of information in theses environments, the messages from the peers have to be causally ordered [5]. The usage of causal ordering in the overlay peer-to-peer systems provides message synchronization and reduces the indeterminism produced by asynchronous execution of the peers, or by random and unpredictable delays of the messages within the communication channels.

In the literature several protocols have been proposed to guarantee causal ordering delivery of the messages (CODM), for a great variety of distributed systems [6-23]. According to the operational network topology for which they were designed, these protocols can be classified into three categories: plain peer-to-peer, free scale peer-to-peer and hierarchical networks.

Plain peer-to-peer networks refer to the CODM protocols designed for the theoretical simplest distributed systems [12, 14]. The main disadvantage of these protocols is that they require an excessive communication overhead, since they do not consider the network topology to manage the exchanged control information. In the worst case the size of the control information grows linearly with the number of peers in the network, causing that the computational effort of each peer increases according to the growth of the neighbourhoods.

Free scale peer-to-peer networks, consider systems in which the peers can be separated in two kinds: peers and super peers. Super peers are nodes with the higher bandwidth or processing power, while the peers are nodes with fewer resources and dependent of super peers. This topology was designed to distribute the effort load by reducing the resources requirements in peers. However, up to date there are no CODM protocols specifically designed for the generic scheme of such networks. To overcome the lack of dedicated solutions for this networks, some protocols for plain topology can be adapted by establishing the CODM under each super peer. Nevertheless, besides the problems related to the uncontrolled overhead growth, these approaches may induce unnecessary inhibition on messages delivery.

Hierarchical networks include CODM protocols where the network topology is established to disable the direct communication among peers, allowing only the connections among super peers and between a super peer and each of its dependent peers. Similar than the generic case of free-scale networks, some protocols proposed for other topologies can be adapted to hierarchical networks. This is the case of the protocol proposed by Friedman and Manor [10], which has good characteristics in hierarchical sparse networks where the peers have a low message sending rate. However, since this protocol has a quadratic traffic cost, in dense networks may cause serious overloads in super peers' channels.

Besides this approach, there are some protocols specifically designed for hierarchical networks [13, 17, 18] which are based on the employment of a global time reference to ensure causal ordering of messages. Unfortunately, the characteristics of most hierarchical networks make difficult to establish a global time reference, this mainly due to the absence of shared memory and the lack of perfectly synchronized clocks [24]. In addition to this disadvantage, these protocols require knowing the maximum delays of the messages in the communication channels, which is not always feasible [13].

In this paper we propose a CODM protocol, to ensure causal ordering of the messages in the hierarchical overlay networks, that does not require the establishment of a global time reference and maintains a low overhead in the communication channels and in the peers. To achieve this, the proposed protocol defines two communication groups, according to the connection type. The first group, called the *internal group*, is a collection of peers connected to a super peer. In the internal group, the protocol messages use a bit vector to represent the causal dependencies, resulting in a bit level overhead. The second group, called the *external group*, consists of interconnected super peers. In this group, the concept of hierarchical clocks [15] is applied to represent message dependencies.

In the proposed protocol, the message control information, transmitted, stored and processed by each peer node, is adapted according to the underlying communication channel network (i.e. wired or wireless) and its capacity (i.e. memory and processing). Through simulations we demonstrate that in both, internal and external groups, the protocol message overhead is reduced up to two to three times than the reported by the Immediate Dependency Relation [14] and Dependency Sequences [15] protocols.

This paper is organized as follows. Section 2 contains an overview of the related work about CODM protocols. Section 3 includes explanations about preliminary concepts. The proposed protocol is presented in section 4. Section 5 contains the simulation results. Finally, conclusions and future work are presented in section 6.

2 Related work

Some protocols have been proposed to implement CODM for peer-to-peer overlay networks. These protocols can be classified in three main categories according to the network architecture they are designed for: plain peer-to-peer, free scale peer-to-peer and hierarchical networks. In addition, CODM protocols also can be classified based on whether they use a global time reference or employ logical references between messages. A general description of these protocols follows:

2.1 Protocols designed for plain networks

A plain peer-to-peer overlay network consists of several interconnected peers, where each one can communicate directly with other peers. In a broadcast group each peer sends messages to every one of the other peers in a group.

2.1.1 Plain network protocols based on global time references

All protocols from this category [9, 19, 20, 21] are based on the concept of using a combination of logical and physical clocks to ensure causal message order. To achieve this, protocols are required to be able to synchronize its clocks with a time server and have knowledge about minimum and maximum delays of messages in communication channels and maximum clock drift. These protocols use temporal records, based on a physical clock, to determine the order of two sent messages m_1 and m_2. In this way, if the difference between temporal records is greater than the maximum transmission delay, the messages are temporally ordered. If the difference between temporal records is smaller than the minimum delay, the messages are considered concurrent. If the temporal order of the messages cannot be determined, a logical clock is used to perform a causal ordering.

2.1.2 Plain network protocols based on logical references

To ensure causal ordering these protocols use the "happened before" relation defined by Lamport [24].

The Causal Ordering in Deterministic Overlay Network [10] protocol uses the idea to forward each received message. Thus, this protocol does not require any control information to be sent within a message, however, in the worst case scenario it produces n-1 copies (where n is the number of peers in the system) of the same message, which can result in the saturation of the communication channels.

The Critical Causal Order of Events in Distributed Virtual Environment protocol [23] is based on the idea of reducing causal order violations without completely removing them. To achieve this reduction, the protocol resends the last received message with the new message, by doubling the average message size. However, the protocol does not require any control information to be sent alongside the message to ensure causal ordering of messages.

An Efficient Algorithm for Causal Message Ordering [12] extends the idea of vector clock [25] to tackle those cases where each message is addressed to a different subset of processes. This modification requires more information storage in comparison to the original Vector clock protocol. In the worst case scenario, this protocol requires to store one entry for each process in a group, producing an overhead of O(n) (where n is the number of peers in the system). Furthermore, when a message is sent, all of the stored entries must be included along with the control information.

Probabilistic Analysis of Causal Message Ordering [22] analyses the probability of violation in causal ordering of messages. This protocol implies that by introducing the delay before message is send the probability of the causal order violation is decreased. This method does not require any control information at all, but reduces the concurrency of the system.

The Immediate Dependency Relation protocol [14] is based on the idea of sending only the identifiers of immediate predecessors of a message. In this way, the average overhead is reduced in comparison to the vector clock protocol. However, in the worst case scenario, the overhead in the communication channels can be as high as the number of peers in the system (O(n), where n is the number of peers in the system). Regarding with the storage requirements, this protocol stores the vector clock and the message control information, thus the resulting storage overhead is twice the number of peers in the system.

2.2 Protocols designed for free scale networks

A free scale network consists of a set of peers and super peers (a super peer is a peer node that features higher processing power and manages wider communication bandwidth). The peers are divided into two groups, based on their connection types. In this case, some of the peers are only connected to a super peer, while some others are connected with other peers and super-peers [26].

At the time when we surveyed the state of the art, no protocols were found specifically designed for this type of networks. Although, the protocols designed for plain networks can be adapted to be used in a free scale network architecture, however, these protocols do not use information about the network topology and treat peers in different groups in the same manner. Therefore, the overhead grows in the same proportion with the number of peers in the system.

2.3 Protocols designed for hierarchical networks

A hierarchical network is a network consisting of peers and super peers where there are only two kind of connections. A peer is only connected to a super peer while super peers communicate among them [26]. In this kind of network, the communication between different group members must be performed via the super peers. In this way, when a peer sends a message to another peer, that is located in a different group, the message transmission is done through three transactions. Firstly, the source peer sends the message to its local super peer. Then, the local super peer forwards the message to the corresponding super peer. Finally, the last super peer delivers the message to the respective receiver.

2.3.1 Hierarchical networks protocols based on global time references

Protocols in this category [13, 17, 18] use a combination of logical and physical clocks to ensure causal message order, similarly as some protocols designed for plain networks. The main characteristic of those protocols is that the group is divided into subgroups to reduce the storage and the computational overhead that each peer manages.

2.3.2 Hierarchical networks protocols based on logical references

In this category we describe the protocols that are designed for hierarchical peer-to-peer networks based on the "happened before" relation [24], or mechanisms that do not involve a global time reference.

Reliable Multicast [6] and Distributed Floor Control protocols [7] use the synchronization mechanisms, provided by some coordinators, to ensure the causal ordering of events. In this case the protocols do not require any control information to be exchanged, nonetheless require some kind of infrastructure to support the communication between peers. These protocols are a special type of server-client protocol where the server ordering is issued to all clients or peers.

The Domain-Based Causal Ordering Group Communication protocol [11] and the Two-Layered protocol for a Large-Scale Group of Processes [16] are based on the using of two vector clocks: one to record the causal dependencies within a subgroup and a second one for the causal dependencies of the entire group. In this manner, the size of the control information and the storage requirements are reduced, implying that concurrent messages in one group become causally ordered in another group. In both protocols, the sending of g integers in a subgroup is necessary where g is the number of peers in the corresponding subgroup and of an l integer in the global group where l is the number of groups.

An Optimal Causal Broadcast Protocol [8] exchanges only the identifier of a last received or sent message. The result of this decision is a constant overhead, but can produce cases when the messages are not correctly ordered.

In Dependency Sequences [15] message causal relations are represented as a sequence of message identifiers in the form of intervals. However, the proposed protocol does not contain any mechanisms to remove unnecessary dependencies. As a result of this fact, each new message has an overhead that is bigger than the previous message. Therefore, the protocol overhead grows indefinitely.

In Hierarchical Clocks [15] two different clocks are used to represent causal message relationships. One clock is used to represent the message dependency on external events (events from other groups) and the second clock represents message dependency on local events (events from the same group). The problem with this protocol is that it does not contain mechanisms to reduce clock sizes, and thus the size of the control information will grow indefinitely. For this fact, similar than Dependency Sequences, Hierarchical Clocks requires additional mechanisms like checkpointing techniques to control the growth of the overhead. Also the Hierarchical Clocks approach requires an extensive calculations to be performed to determine the causal order of two events.

3 Preliminaries

This section describes the system model along with the concepts and definitions that are used in the presented work.

3.1 System model

A hierarchical peer-to-peer network consists of peers that are connected only to super peers and interconnected super peers. In this manner, in a hierarchical peer-to-peer network, two kind of groups can be distinguished [see Figure 1]:

1. Internal groups (peers that are connected only to a super peer)
2. External group (interconnected super peers)

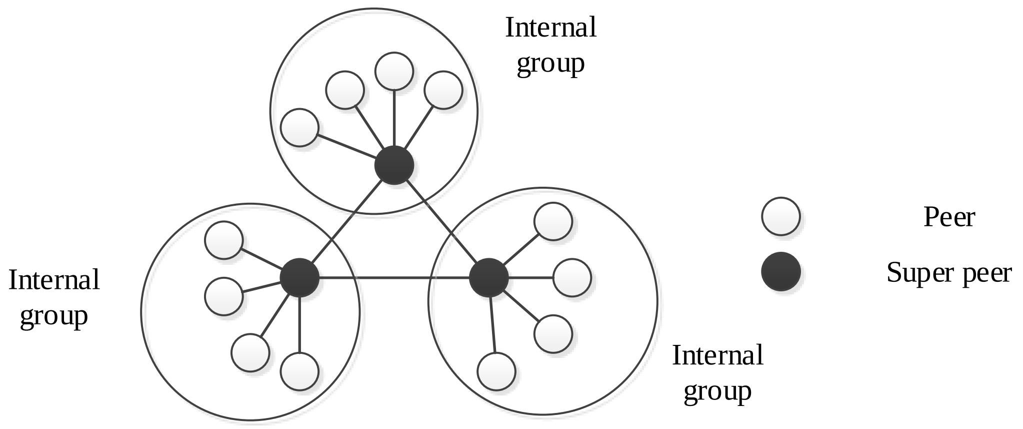


Figure 1. Internal and external groups of a hierarchical P2P network.

Peer in an internal group are also called internal peers. In the proposed protocol, some peers are allowed to belong to an external group and are called external peers.

In a hierarchical network peers can only belong to one group (internal or external). A super peer is a member that belongs to both groups at the same time. An internal group can have only one super peer while an external group consists of several super peers.

A super peer is a special node with higher computer processing power and wider bandwidth capacity compared to peers. In an internal group peers are considered to have lower processing power or bandwidths compared to external group peers. In an internal group peers generally can be represented by mobile devices such as: smart phones and tablets, connected via wireless cellular network to the Internet.

A super peer in an external group can be seen as a meta-process [15] representing all of the events of an internal group. On the other hand, the super peer can be seen as a meta-process representing all of the events in the external group for peers in the internal group. Under this concept, peers in the internal group do not require an extensive knowledge about peers in an external group and vice versa.

The communication channels are considered to be reliable with random but finite delays. Thus, every message will eventually arrive to its destination process. The channels are also considered to be non FIFO channels, implicating the messages can be reordered by the communication channel.

3.2 Background and Definitions

Causal ordering was developed to remove inconsistencies in message delivery, which is produced by an unpredictable delay in the communication channels. Causal order is based on the “happened before” relation defined by Lamport [24]. This relation is denoted by “→” as follows.

Definition 1**.** The relation “→” on the set of events of a system is the smallest relation satisfying the following three conditions:

1. If a and b are events in the same process, and a comes before b, then $a \rightarrow b$.
2. If a is the sending of a message by one process and b is the receipt of the same message by another process, then $a \rightarrow b$
3. If $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$.

Two distinct events a and b are said to be concurrent $a \parallel b$ if neither $a \nrightarrow b$ nor $b \nrightarrow a$. This relation can be extended to messages in the following form: message $m \rightarrow$ message m' if and only if $send(m) \rightarrow send(m')$ where *send* is the message sending event.

3.2.1 The Immediate Dependency Relation (IDR)

The IDR [14] is the propagation threshold of the control information, regarding the messages sent in the causal past which must be transmitted to ensure a causal delivery. IDR is denoted as “↓” and its formal definition is as follows.

Definition 2**.** Two messages m and m' form an IDR $m \downarrow m'$ if and only if $m \rightarrow m'$ and m'' does not exist, such that $m \rightarrow m''$ and $m'' \rightarrow m'$.

Thus, a message m directly precedes a message m', if and only if no other message m'' exists in a system, such that m'' belongs at the same time to the causal future of m, and to the causal past of m'.

This relation is important since if the delivery of messages respects the order of their diffusion for all pairs of messages in IDR, then the delivery will respect the causal delivery for all messages.

Causal information that includes the messages immediately preceding a given message is sufficient to ensure a causal delivery of such message [14].

3.2.2 Process and meta-process

Definition 3**.** A single process is defined to be a totally ordered set of events [24].

In other words a process can be defined as a set of events and for each two events from this set it is possible to determine which of these events happened before.

Definition 4**.** A meta-process is defined to be a partially ordered set of events [15]. It can be used to represent a group of processes.

A meta-process allows for some events to be concurrent, thus condition 1 from *Definition* 1 cannot be applied to a meta-process. So if *a* and *b* are events in the same meta-process, and *a* comes before *b*, this does not mean that $a \rightarrow b$.

4 Protocol composition

4.1 Data structure

In order to define data structures to ensure message causal ordering we need to define additional data types and structures that will be used throughout this work.

4.1.1 Bit vector

Bit vector is an array of variable size. Each element can take only two values: set (represented by 1) and cleared (represented by 0). Each bit vector can be extended with zeros to a required size and the trailing zeros can be trimmed. An empty bit vector is denoted as Ø.

Bits in the bit vector are numbers starting from 1. *V*[*x*] represents a bit at position *x* in vector *V*. A bit at position 0 is assumed to be always set. Bit vectors support AND (&), OR (|) and NOT ($\overline{}$) operations that are bitwise i.e. the operation is applied to bits at position 1, and then bits at position 2, etc.

4.1.2 Extended linear time

Extended Linear Time (*LTx*) is a data type that can contain only one of the following:

- An integer number.
- A bit vector.

Extended Linear Time cannot contain both an integer and a bit vector. Additionally, it is possible to determine at any given time whether a given linear time contains an integer or a bit vector.

If this data type contains an integer, it represents a process, and if it contains a bit vector, it represents a meta-process.

4.1.3 Extended vector time

Extended Vector Time (*VTx*) is a vector of *LTx*. Each element does not depend on others. Thus, a vector can have one element that is an integer and another element that is a bit vector at the same.

4.1.4 Internal peer data structures

Each internal peer maintains the following data:

- id_{int} – identifier of a peer in the internal group. This identifier must be unique in a group.
- *SN*(*p*) – an integer representing a sequence number of a message.
- *RVint*, *RVext* – bit vectors representing received messages.
- *DVint*, *DVext* – bit vectors representing message IDR.

4.1.5 External peer data structures

In an external group, each peer maintains the following variables:

- $id_{ext}(p)$ – identifier of a peer in the external group. This identifier must be unique in a group.
- *VTx*(*p*) – extended vector time. The size of a vector is *G*, where *G* is the number of peers and super peers in an external group.
- *CI* – vector of pairs representing message control information. Each pair consists of a process identifier and *LTx*. *CI*[*x*] is a pair where the process identifier is *x*.

4.1.6 Super peer data structure

Super peer maintains the following variables:

- $id_{ext}(sp)$ – identifier of a super peer in the external group.
- *VTx*(*sp*) – extended vector time. Size of a vector is *G*, where *G* is the number of peers and super peers in an external group.
- *SN*(*sp*) – an integer representing a sequence number of a message from external group.
- *LR* – vector of pairs of size *L*, where *L* is the number of peers in an internal group of this super peer. Each pair contains two sequence number names, *in* and *out*.

- *TT* – vector of vectors of pairs. Size of a vector is *G*. Each pair consists of two message identifiers called *in* and *out*.

4.1.7 Internal group message structure

Messages in an internal group are denoted by m_{int} and have the following structure:

$$m_{int} = (id, SN, Last, DVint, DVext, Data)$$

- *id* – is the identifier of a sending process in the internal group.
- *SN* – an integer representing a message sequence number.
- *Last* – an integer identifier of a last message from this peer.
- *DVint*, *DVext* – a bit vector representing message dependency.
- *Data* – application data to be transmitted.

This message structure is used by both peers and super peers in an internal group [see Figure 2].

4.1.8 External group message structure

In an external group messages are denoted by m_{ext} and have the following structure:

$$m_{ext} = (id, SN, Last, CI, Data)$$

- *id* – is the identifier of a sending process.
- *SN* – an integer representing a message sequence number.
- *Last* – an integer identifier of a last message from this peer.
- *CI* – vector of pairs representing message control information. Each pair consists of a process identifier and *LTx*.
- *Data* – the user data to be transmitted.

This message structure is used by both peers and super peers in an external group [see Figure 2].

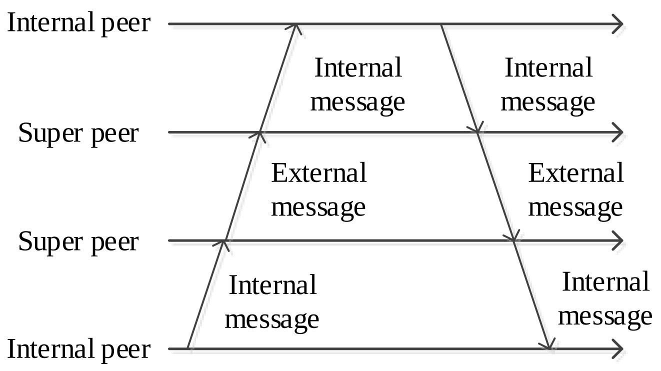


Figure 2. Messages in a hierarchical peer-to-peer network.

4.2 Specification of the causal protocol

A message transaction in an internal group begins with a peer running an initialization process setting the fields of the data structure with the following values:

```
$SN(p) := 0$

$RVint := RVext := \emptyset$

$DVint := DVext := \emptyset$
```

Listing 1. Internal peer initialization.

On the other hand, an external peer runs an initialization process to set the fields of the data structure with the following values:

```
$VTx(p)[Z] := \begin{cases} 0 & \text{,if } Z \text{ is an identifier of a peer} \\ \emptyset & \text{,if } Z \text{ is an identifier of a super peer} \end{cases}$

$CI := \emptyset$
```

Listing 2. External peer initialization.

An initialization process in a super peer sets the fields of the data structure with the following values:

```
$VTx(sp)[Z] := \begin{cases} 0 & \text{,if } Z \text{ is an identifier of a peer or } Z = id_{ext} \\ \emptyset & \text{,if } Z \text{ is an identifier of a super peer} \end{cases}$

$SN(sp) := 0$

$LR := (<0, 0>, <0, 0>, \ldots, <0, 0>)$

$TT := (\emptyset, \emptyset, \ldots, \emptyset)$
```

Listing 3. Super peer initialization.

Each time an internal peer requires to send a message to a super peer, the peer constructs the message using the following procedure [Listing 4]. (The detailed example of the protocol functions are present in the section 4.3).

```
$SN(p) := SN(p) + 1$

$m_i = (id_{int}, SN(p), 0, DVint, DVext, Data)$
```

```
$DVint := DVext := Ø$
```

Listing 4. Message sending by internal peer

Each time an internal peer receives a message from a super peer, it verifies the message delivery condition. First, a FIFO condition is verified to check that a received message has not arrived before a previous message from the same sender. Then after such verification, the peer checks that it has received all of the messages that form the immediate dependency relation with the currently received message [see Listing 5].

If the delivery condition is satisfied, the peer updates its data structures [Listing 6] and then delivers the message to the corresponding application. If the delivery condition is not satisfied, the message should be buffered.

```
If ($m_i.id \neq 0$ and $RVint[m_i.Last] = 1$) or ($m_i.id = 0$ and $RVext[m_i.Last] = 1$) then // FIFO condition
    If (($m_i.DVint$ & $RVint = m_i.DVint$) and ($m_i.DVext$ & $RVext = m_i.DVext$)) then // Causal condition
        Deliver($m_i$)
    End if
End if
```

Listing 5. Internal peer delivery condition.

An internal peer will always receive its own message returned by a super peer. In this case the peer should only update its receive vector *RV*. After a message is delivered, a process should check its buffer. If a message in a buffer satisfies a delivery condition, it should be delivered using the same algorithm [Listing 6].

```
If ($m_i.id \neq 0$) then
    $RVint[m_i.SN] := 1$
Else
    $RVext[m_i.SN] := 1$
End if
If ($m_i.id \neq id_{int}$) then
    $DVint := DVint$ & $\overline{m_i.DVint}$
    $DVext := DVext$ & $\overline{m_i.DVext}$
    If ($m_i.id \neq 0$) then
```

```
            $DVint[m_i.SN]$ := 1
            $DVint[m_i.Last]$ := 0
        Else
            $DVext[m_i.SN]$ := 1
            $DVext[m_i.Last]$ := 0
        End if
End if
```

Listing 6. Message delivery to internal peer.

Since a super peer only can transmit messages from other peers, it does not have a message emission phase. When a super peer receives a message from an internal peer, the super peer checks for the message delivery condition. Since an internal peer can only receive messages from a super peer, which means that only a FIFO dependency requires to be checked. To check the message delivery condition, a super peer executes the following algorithm [Listing 7].

```
If ($m_i.SN = LR[m_i.id].in + 1$) then
        Deliver($m_i$)
End if
```

Listing 7. Super peer delivery condition for messages from an internal group.

When a delivery condition is satisfied, the super peer forwards the message to other peers in an internal group. [Listing 8]. Otherwise, the message should be buffered.

After a message is delivered, the process should check its buffer. If a message in a buffer satisfies a delivery condition, it should be delivered using the same algorithm [Listing 8]. A message transformation will be discussed later in this section.

```
$m_i.Last$ := $LR[m_i.id].out$
$VTx[id_{ext}]$ := $VTx[id_{ext}] + 1$
$LR[m_i.id]$ := <$m_i.SN$, $VT[id_{ext}]$>
$m_i.SN$ := $VTx[id_{ext}]$
Send $m_i$ to all peers in the internal group
```

```
Transform and send message $m_i$ to all peers in the external group
```

Listing 8. Internal message delivery to super peer.

It is noted that the number of bit vector in the internal group is constant and equal to two. It does not depend on the number of peers nor the number of super peers in the system. Thus, peers joining or leaving does not affect other peers in the system. The only modifications in data structures are required at super peer's level. Also for peers joining the group the initialization process is required so it can receive the messages generated after it have joined the group. This process consists of setting the first $VTx[id_{ext}]$ bits in the *RVint* and first $SN(sp)$ bits in the *RVext* vector.

When an external peer wants to send a message, it constructs it using the algorithm [Listing 9].

```
$VTx[id_{ext}] := VTx[id_{ext}] + 1$
$m_e = (id_{ext}, VTx[id_{ext}], VTx[id_{ext}] - 1, CI, Data)$
$CI := \emptyset$
```

Listing 9. Message sending by external peer.

When a message is received by an external peer or a super peer, it should check its delivery condition which consists of checking FIFO and causal conditions.

If the message FIFO ordering is not violated, then a message causal delivery condition is checked. This condition consists on checking message identifiers that are inside messages control information [Listing 10]. These conditions (FIFO and causal) are checked in both external peers and a super peer when it receives a message from an external group.

```
If $\begin{cases} m_e.Last \leq VTx[m_e.id] & \text{,if } VTx[m_e.id] \text{ is an integer} \\ VTx[m_e.id][m_e.Last] = 1 & \text{,if } VTx[m_e.id] \text{ is a bit vector} \end{cases}$ then    // FIFO
        // Causal
        If for each $<i, dep>$ $(i \neq id_{ext})$ in $m_e.CI$ => $\begin{cases} dep \leq VTx[i] & \text{,if } VTx[i] \text{ is an integer} \\ dep \ \& \ VTx[i] = dep & \text{,if } VTx[i] \text{ is a bit vector} \end{cases}$ then
                Deliver($m_e$)
        End if
End if
```

Listing 10. Message delivery condition in external group.

If a delivery condition is satisfied in an external peer, it can deliver a message to an application. To do this a peer is required to update its data structures to ensure that following messages will be correctly ordered [Listing 11]. This update consists of two parts. First, it needs to update the clock so that messages that depend on this one can be delivered. The second part is to update control information so that a message sent from this peer will be correctly ordered by other peers.

```
If $VTx[m_e.id]$ is an integer then
    $VTx[m_e.id] := m_e.SN$
Else
    $VTx[m_e.id][m_e.SN] := 1$
End if
If $VTx[m_e.id]$ is an integer then
    If exists $<i, dep>$ in $CI$ that $i = m_e.id$ then
        $CI := CI \setminus <i, dep>$
    End if
    $CI := CI \cup <m_e.id, m_e.SN>$
Else
    If not exists $<i, dep>$ in $CI$ that $i = m_e.id$ then
        $CI := CI \cup <m_e.id, \emptyset>$
    End if
    $CI[m_e.id][m_e.SN]: = 1$
End if
For each $<i, dep>$ in $m_e.CI$
    If $VTx[i]$ is an integer then
        $CI := CI \setminus <i, dep>$
    Else
        $CI[i] := CI[i] \ \& \ \overline{dep}$
    End if
End for
```

Listing 11. Message delivery to external peer.

If for any pair in *CI* a bit vector is empty this pair can be removed from *CI*. If a delivery condition is satisfied in a super peer it can forward this message to an internal group. To do this a super peer is required to update its data structures to ensure the delivery of messages that depend on this one. This requires an update of a clock so that messages that depend on this one can be delivered.

```
If $VTx[m_e.id]$ is an integer then
        $VTx[m_e.id] := m_e.SN$
Else
        $VTx[m_e.id][m_e.SN] := 1$
End if
$SN(sp) := SN(sp) + 1$
Transform and send message $m_e$ to all peers in the internal group
```

Listing 12. External message delivery to super peer.

After a message is delivered to a peer or a super peer, a process should check its buffer. If a message in a buffer satisfies a delivery condition, it should be delivered using the same algorithm [Listing 11 for peer, Listing 12 for super peer].

As bit vectors *RV*, *DV* and *VTx* grow in size during the execution of a protocol with each message, it is necessary to use mechanisms to reduce bit vector sizes. Communication channels are considered to be reliable; thus, every message sent by a super peer will be delivered to an internal group peer. This means that an *RV* and *VTx* will have bits for each message set. Since a super peer numbers messages with consecutive integers after some execution time, an *RV* and *VTx* will start with consecutive set bits. Considering that after a bit is set, it is not changed to a cleared state at any time. So it is required to store bits between the first cleared bit and the last set bit.

A vector *DV* is based on the immediate dependency relation. Each bit is set only once and then it is cleared when a message is sent or a dependent message is received. So it is only required to store bits between the first and the last set bits.

To be completely functional our protocol requires a mechanism to transform messages from an internal group to an external group and vice versa. This transformation is performed by a super peer because it participates in both groups at the same time.

A message that originated from an internal group generally carries dependencies on other messages from the internal group and dependencies on messages from an external group. The dependencies on messages from an internal group are represented in a form of bit vector but to be interpreted correctly in an external group they should be transformed into a vector of pairs (process identifier, message dependency). This transformation can be achieved by using the algorithm presented below [Listing 13].

```
CI := (<id_ext, m_i.DVint>)   // Vector of pairs.
For each <id, PT> in TT   // PT is a vector of pairs
    For each <in, out> in PT in reverse order
        If m_i.DVext[out] = 1 then
            If VTx[id] is an integer then
                CI := CI ∪ <id, in>
                Exit For
            Else
                If not exists <i, dep> in CI that i = id then
                    CI := CI ∪ <id, Ø>
                End if
                CI[id][in] := 1
            End if
        End if
    End for
End for
m_e := (id_ext, m_i.SN, m_i.Last , CI, m_i.Data)
Send m_e to external group
```

Listing 13. Message transformation from internal to external group.

If a message carries only dependencies on external messages a *CI*[id_{ext}] will be an empty vector. In this case, this dependency can be removed from *CI*.

A message received by a super peer that originated from an external group generally carries dependencies on other messages from an external group as well as dependencies on messages from an internal group. The dependencies on messages from an external group are represented in a form of pairs (process identifier, message dependency) but to be interpreted correctly in an internal group they should be transformed to a bit vector form [Listing 14].

```
$DVint$ := $m_e.CI[id_{ext}]$ or Ø if $m_e.CI$ does not contain element for $id_{ext}$ // Bit vector
$DVext$ := Ø
For each <$id$, $dep$> in $m_e.CI$ where $id \neq id_{ext}$
    If ($VTx[id]$ is an integer)
        If (exists <$in$, $out$> in $TT[id]$ where $in = dep$) then
            $DVext[out]$ := 1
        End if
    Else
        For each 1 in $dep$ in position $i$
            If exists <$in$, $out$> in $TT[id]$ where $in = i$ then
                $DVext[out]$ := 1
            End if
        End for
    End if
End for
$Last$ := $TT[m_e.id][in = m_e.Last]$ or 0 if not exists
$m_i = (0, SN(sp), Last, DVint, DVext, m_e.Data)$
$TT[m_e.id]$ := $TT[m_e.id] \cup$ <$m_e.SN$, $VTx[id_{ext}]$>
Send $m_i$ to internal group
```

Listing 14. Message transformation from external to internal group.

Also a super peer can receive messages that contain dependencies on messages that are not yet received. To deal with this, super peer checks the message delivery condition as previously described to ensure that this message can be delivered to a super peer and only then transforms it to ensure that all of the message dependencies are resolved. This does not affect the message order in any way. If a super peer does not receive message *m*, none of the peers in an internal group have received this message *m*. So a message *m'* that requires *m* to be delivered before cannot be delivered to any peer in an internal group.

4.3 Causal protocol description

To demonstrate how our proposed protocol detects causal order violations, we use a scenario [see Figure 3] composed of a network that consists of the following:

- Two internal groups containing of two peers each. $P(i_1)_1$ and $P(i_1)_2$ are the peers in the internal group 1 and $P(i_2)_1$ and $P(i_2)_2$ are the peers in the internal group 2.
- Two super peer with identifiers 1 and 2. Super peer 1 forms the internal group 1 and super peer 2 forms the internal group 2.

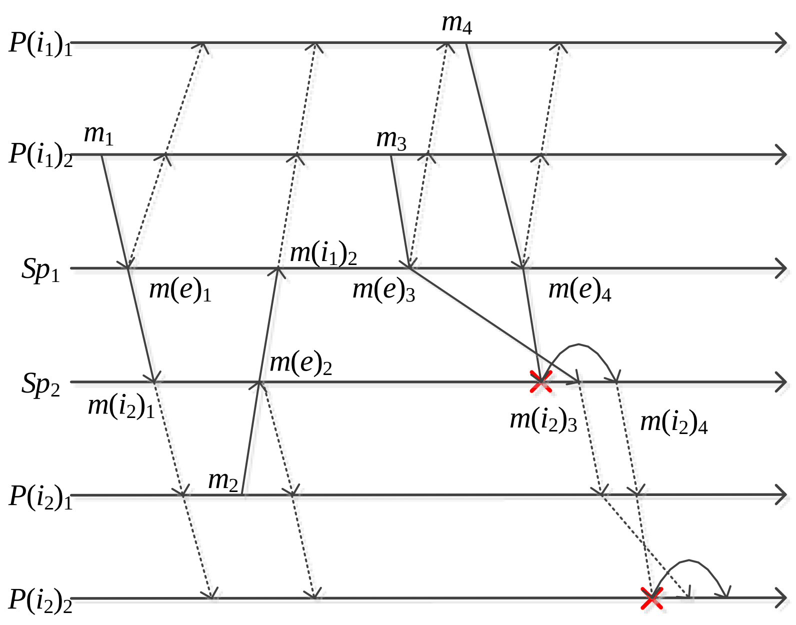


Figure 3. Scenario example.

This scenario contains 3 types of messages:

- Messages in an external group represented as a solid line.
- Messages in an internal group from peer to super peer represented by a solid line.
- Messages in an internal group from super peer to peers represented by a dotted line.

We mark with the X a message delivery that violates a causal order.

4.3.1 Diffusion of m_3 at $P(i_1)_2$.

- First, an internal peer increments its sequence number [Listing 4, Line 1]: $SN = 2$.
- An internal peer generates a message $m_3 = (2, 2, 0, Ø, 1, Data)$ [Listing 4, Line 2].
- The *DVint* and *DVext* variables are cleared [Listing 4, Line 3].

4.3.2 Reception of m_3 at Sp_1.

- When a super peer Sp_1 receives m_3 it checks its delivery condition. The FIFO delivery condition is satisfied [Listing 7, Line 1]: $2 = 1 + 1$ and the message m_3 can be delivered.

- m_3.*Last* field is changed to 1 (*LR*[2].*out* = 1) [Listing 8, Line 1].
- A clock for this super peer is updated with the new value [Listing 8, Line 2]: *VTx*(*p*) = (2, 1) and *LR* is updated with the message identifier [Listing 8, Line 3]: *LR*[2] = <2, 2>.
- m_3.*SN* is updated to 2 and m_3 is sent to all peers in an internal group.

4.3.3 Transformation of m_3 at Sp_1 to external group.

- *CI* is created with value <1, Ø> [Listing 13, Line 1].
- *TT*[2] contains <1, 1> and bit m_3.*DVext*[1] is set [Listing 13, Lines 2-4].
- The element <2, 1> is added to *CI* [Listing 13, Lines 9-12].
- The element <1, Ø> is removed from *CI* and the message m_3 is transformed to an external group: $m(e)_3$ = (1, 2, 1, <2, 1>, m_3.*Data*) [Listing 13, Line 17].

4.3.4 Reception of m_3 at $P(i_1)_1$.

- When a peer $P(i_1)_1$ receives m_3 it checks its delivery condition.
  - The FIFO delivery condition is satisfied [Listing 5, Line 1]: *RVint*[1] = 1
  - The causal delivery condition [Listing 5, Line 2] is also satisfied: 1 & 1 = 1.
  - Both conditions satisfied and the message m_3 can be delivered.
- The receive vector is updated [Listing 6, Line 2]: *RVext* = 11.
- The message dependency vectors are updated as well [Listing 6, Lines 7-14]: *DVint* = 01, *DVext* = Ø.

4.3.5 Diffusion of m_4 at $P(i_1)_1$.

- First, an internal peer increments its sequence number [Listing 4, Line 1]: *SN* = 1.
- An internal peer generates a message m_4 = (1, 1, 0, 01, Ø, *Data*) [Listing 4, Line 2].
- The *DVint* and *DVext* variables are cleared [Listing 4, Line 3].

4.3.6 Reception of m_4 at Sp_1.

- When a super peer Sp_1 receives m_4 it checks its delivery condition. The FIFO delivery condition is satisfied [Listing 7, Line 1]: 1 = 0 + 1 and the message m_4 can be delivered.
- m_4.*Last* field is changed to 0 (*LR*[1].*out* = 0) [Listing 8, Line 1].
- A clock for this super peer is updated with the new value [Listing 8, Line 2]: *VTx*(*p*) = (3, 1) and *LR* is updated with the message identifier [Listing 8, Line 3]: *LR*[1] = <1, 3>.
- m_4.*SN* is updated to 3 and m_4 is sent to all peers in an internal group.

4.3.7 Transformation of m_4 at Sp_1 to external group.

- *CI* is created with value <1, 01> [Listing 13, Line 1].
- *TT*[2] contains <1, 1> and bit m_4.*DVext*[1] is cleared [Listing 13, Lines 2-3].
- *CI* is not modified [Listing 13, Lines 4].
- The message m_4 is transformed to an external group: $m(e)_4$ = (1, 3, 0, <1, 01>, m_4.*Data*) [Listing 13, Line 17].

4.3.8 Reception of $m(e)_4$ at Sp_2.

- When a super peer Sp_2 receives $m(e)_4$ it checks its delivery condition.
  - The FIFO delivery condition is satisfied [Listing 10, Line 1]: *VTx*[1][0] is set.
  - $m(e)_4$.*CI* contains <1, 01> and this dependency is not satisfied because 01 & 10 = 00 $\neq$ 01.
  - The message $m(e)_4$ should be buffered.

4.3.9 Reception of $m(e)_3$ at Sp_2.

- When a super peer Sp_2 receives $m(e)_3$ it checks its delivery condition.
  - The FIFO delivery condition is satisfied [Listing 10, Line 1]: *VTx*[1][1] is set.
  - $m(e)_3$.*CI* contains <2, 1> and this dependency is satisfied because id_{ext} = 2.
  - Both conditions are satisfied and the message $m(e)_3$ can be delivered.
- A clock component for super peer 1 [Listing 12, Line 4] is updated *VTx*(*sp*) = (11, 1).
- *SN*(*sp*) is incremented to 2 [Listing 12, Line 6].

4.3.10 Transformation of $m(e)_3$ at Sp$_2$ to internal group.

- As $m(e)_3$.*CI* contains an element for process 2 [Listing 14, Line 1]. *DVint* is initialized to 1.
- In this case, $m(e)_3$.*CI* does not contains other dependencies so $m(e)_3$ is transformed to an internal group: $m(i_2)_3$ = (0, 2, 1, 1, Ø, $m(e)_3$.*Data*). *TT*[1] contain <1, 1> so *Last* is set to 1.
- <2, 2> is added to *TT*[1] [Listing 14, Line 17].: *TT*[1] contains <1, 1>, <2, 2>.

4.3.11 Delivery of $m(e)_4$ at Sp_2.

- Message buffer contains $m(e)_4$ it delivery condition should be revalidated.
  - The FIFO delivery condition is satisfied [Listing 10, Line 1]: *VTx*[1][0] is set.
  - $m(e)_4$.*CI* contains <1, 01> and this dependency is satisfied because 01 & 11 = 01.
  - Both conditions are satisfied and the message $m(e)_4$ can be delivered.
- A clock component for super peer 1 [Listing 12, Line 4] is updated *VTx*(*sp*) = (111, 1).

- *SN*(*sp*) is incremented to 3 [Listing 12, Line 6].

4.3.12 Transformation of $m(e)_4$ at Sp_2 to internal group.

- As $m(e)_4.CI$ does not contains an element for process 2 [Listing 14, Line 1]. *DVint* is initialized to Ø.
- $m(e)_4.CI$ contains the element <1, 01> and *TT*[1] contains <2, 2>. *DVext* is updated to 01.
- $m(e)_4$ is transformed to an internal group: $m(i_2)_4$ = (0, 3, 0, Ø, 01, $m(e)_3$.*Data*).
- <3, 3> is added to *TT*[1] [Listing 14, Line 17].: *TT*[1] contains <1, 1>, <2, 2>, <3, 3>.

4.3.13 Reception of $m(i_2)_4$ at $P(i_2)_2$.

- When a peer $P(i_2)_2$ receives $m(i_2)_4$ it checks its delivery condition.
  - The FIFO delivery condition is satisfied [Listing 5, Line 1]: *RVext*[0] = 1.
  - The causal delivery condition [Listing 5, Line 2] is not satisfied: 01 & 10 = 00 ≠ 01.
  - The message $m(i_2)_4$ should be buffered.

4.3.14 Reception of $m(i_2)_3$ at $P(i_2)_2$.

- When a peer $P(i_2)_2$ receives $m(i_2)_3$ it checks its delivery condition.
  - The FIFO delivery condition is satisfied [Listing 5, Line 1]: *RVext*[1] = 1.
  - The causal delivery condition [Listing 5, Line 2] is also satisfied: 1 & 1 = 1.
  - Both conditions are satisfied and the message $m(i_2)_3$ can be delivered.
- The receive vector is updated [Listing 6, Line 2]: *RVext* = 11.
- The message dependency vectors are updated as well [Listing 6, Lines 7-11]: *DVint* = Ø, *DVext* = 01.
- Message buffer contains $m(i_2)_4$ it delivery condition should be revalidated.
  - The FIFO delivery condition is satisfied [Listing 5, Line 1]: *RVext*[0] = 1.
  - The causal delivery condition [Listing 5, Line 2] is also satisfied: 01 & 11 = 01.
  - Both conditions are satisfied and the message $m(i_2)_4$ can be delivered.
- The receive vector is updated [Listing 6, Line 2]: *RVext* = 111.
- The message dependency vectors are updated as well [Listing 6, Lines 7-11]: *DVint* = 001, *DVext* = Ø.

4.4 Overhead Analysis

As the proposed protocol depends on the Immediate Dependency Relation [14], the size of the control information of message *m* depends on the number of concurrent messages that form an IDR with *m*.

In the internal group all of the messages are sequentially numbered. As a message *m* cannot form an IDR with more than *g* internal messages (*g* is the number of processes in an internal group) its internal dependency vector cannot contain more than *g*-1 set bits. Also a message *m* cannot form an IDR with more than *n*-*g* external messages (*n* is the number of processes in the system) its external dependency vector cannot contain more than *n*-*g*-1 set bits. But the set bits can be separated by cleared bits. Let m_1 be the message with lowest sequence number to form an IDR with *m*. Then a bit vector can have no more bits than the number of message concurrent to m_1 that exists in a system. As message delay is finite, then each process can generate a finite number of messages concurrent to m_1. Thus, a total number of messages concurrent to m_1 is also finite and is proportional to a number of processes in a system producing an overhead of O(*n*) bits (O(*g*) for internal group + O(*n*-*g*) for external group).

In the external group, message dependencies are represented as a combination of dependencies on external messages and dependencies on internal messages. The number of elements that represent dependencies on external messages are limited by the number of processes in an external group (peers and super peers), thus limiting a number of pairs that represent message dependency to O(*l*) (*l* is the number of peers and super peers in the external group).

We notice that in our protocol, as for the minimal causal algorithm in [14], the likelihood that the worst case will occur approaches zero as the number of participants in the group grows. This is because the likelihood that *k* concurrent messages occur decreases inversely proportional to the size of the communication group. This behaviour has been shown in [14].

5 Simulations

To analyse our protocol we carried out different simulations. The scenario used in these simulations consists of four internal groups and one external group connected by four super peers. All of the peers in the system were distributed equally among these four internal groups. Within the simulation, each peer generates a message every 70 – 90 milliseconds. The system was simulated with a different number of peers and with different delays in the communication channels. The simulations were performed with the OMNeT++ discrete event simulator [27]. All of the simulation scenarios are listed in Table 1.

All the delays are normally distributed with the mean being the middle of the interval and the variance equal to one fourth of the interval (for example, message generation time is distributed like N(80, 5) milliseconds). If a random value is generated outside the interval, the value of the nearest interval end is taken instead. The message delay is applied individually at each channel (peer – super peer, super peer – super peer, super peer – peer).

Message channel delays	Number of peers
0 – 50 milliseconds	10 – 900
50 – 250 milliseconds	10 – 800
50 – 550 milliseconds	10 – 500

Table 1. Number of peers and delays.

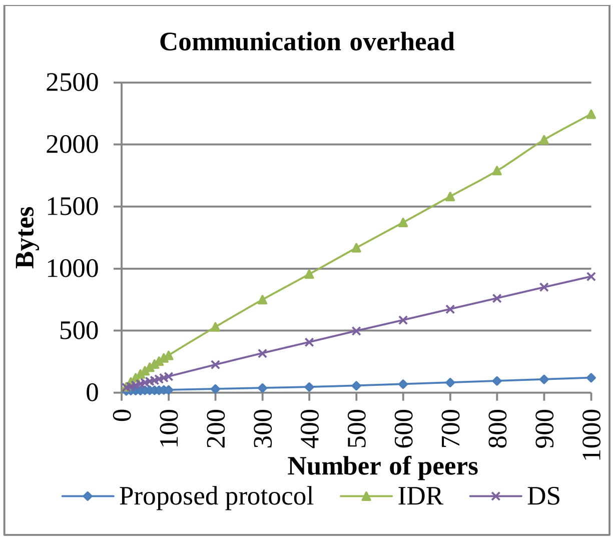
The simulation program uses the Immediate Dependency Protocol [14] to compare and validate the protocol presented in this paper. When a message can be delivered to an application, following our protocol, it is validated against an IDR to identify the causal order violations which our protocol failed to detect. In addition, the overheads for IDR and Dependency Sequences (DS) [15] were calculated to be compared with the overhead generated by our protocol.

The results of simulations are analysed in the following way. Two overheads are mainly analysed: the communication overhead (amount of control information required to be sent with a message) and storage overhead (amount of information required to be stored in each peer). As our protocol generates different overheads in internal and external groups, the maximum overhead in both groups is compared with the generated by IDR and DS protocols.

As Dependency Sequences [15] stores information only at super peers level, the storage overhead for this protocol is not analysed.

By considering channel delays from 0 to 50 milliseconds [see Figure 4], the simulation results show that the overhead of our protocol is lower than the overhead produced by the IDS and DS protocols. For 1000 peers, each internal peer requires to store on average 140 bytes of information, and the average communication overhead is around 120 bytes. IDR protocol requires to store on average 5850 bytes on each peer and to send 2250 bytes, while DS protocol sends on average 950 bytes in the external group.

For these delays the results show that the overhead of our protocol is 18.75 times lower than the overhead of the IDR and 7.9 times lower than the one of DS protocol, and require storing 41.75 times less information.



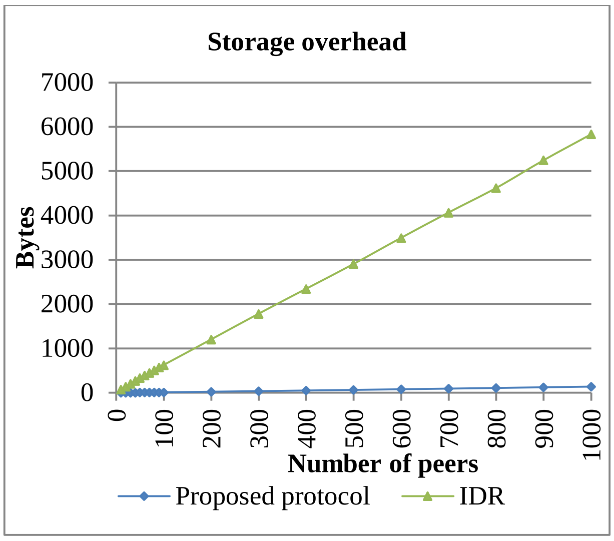


Figure 4. Communication and storage overheads in bytes for a system with delays from 0 to 50 milliseconds.

In our work in order to reduce the size of the variable *TT*, in a super peer, we use the fact that a peer in the internal group can only receive a message that has been already received and delivered by the corresponding super peer. If m_1 and m_2 are messages that have an immediate dependency relationship $m_1{\downarrow}m_2$, and all the peers in the internal group have received and delivered m_2, then the information about m_1 can be removed from a super peer. This can be seen from two different aspects: message reception and message sending:

Message reception: a peer can deliver m_2, such that $m_1{\downarrow}m_2$, if and only if m_1 has already delivered. If a peer received a message m_3, such that $m_1{\downarrow}m_3$, and m_2 has already delivered (which implies delivery of m_1), then it can deliver m_3 without any delay. In this way the information about dependency on m_1 does not affect the m_3 delivery in any way.

Message sending: if a peer delivered message m_2 then no message originated from this peer can carry any dependency on m_1 and the information about m_1 in super peer is no longer required.

Therefore, if all peers have delivered m_2, such that $m_1{\downarrow}m_2$, then the delivery of messages depending on m_1 will not be affected in the internal group in any way, and a super peer will not receive any message depending on m_1 from the internal group. Therefore, the information about m_1 can be deleted from variable *TT* in a super peer.

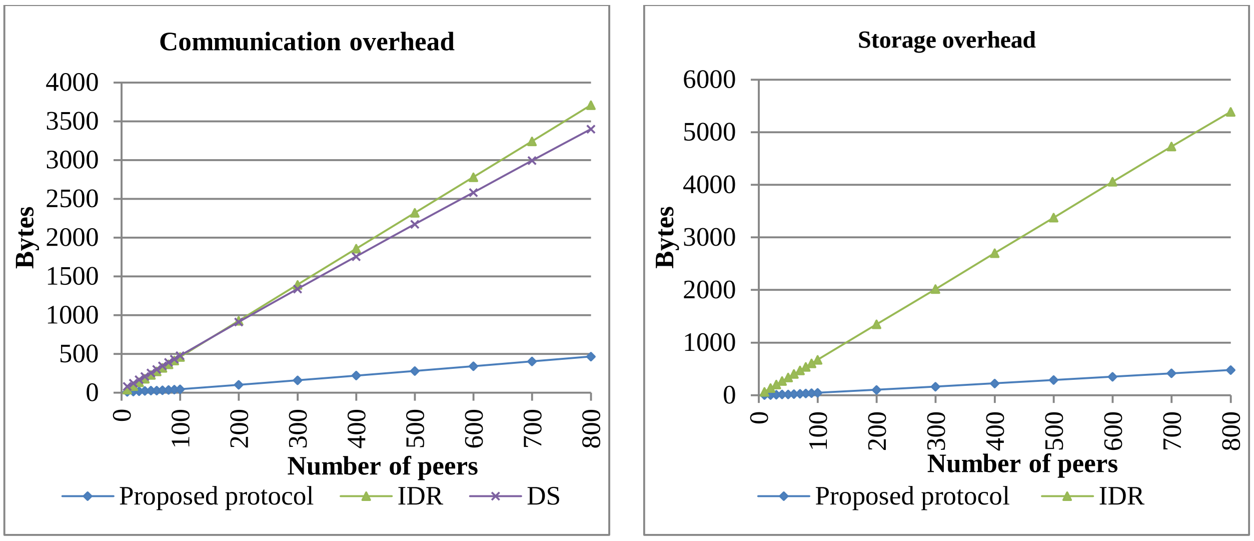


Figure 5. Communication and storage overheads in bytes for a system with delays from 50 to 250 milliseconds.

By considering channel delays from 50 to 250 milliseconds [see Figure 5] the results also show that the overhead of the proposed solution is lower than the overhead produced by the Immediate Dependency Relation and Dependency Sequences protocols. For 800 peers, each internal peer requires to store on average 480 bytes of information and the average overhead is around 470 bytes. The IDR protocol requires to store on average 5400 bytes on each peer and to send 3700 bytes while DS have a communication overhead of 3400 bytes.

The results for delays from 50 to 250 milliseconds show that overhead of our protocol is 7.8 times lower than the overhead of the IDR, 7.2 times lower that the overhead of DS and require storing 11.25 times less information for IDR.

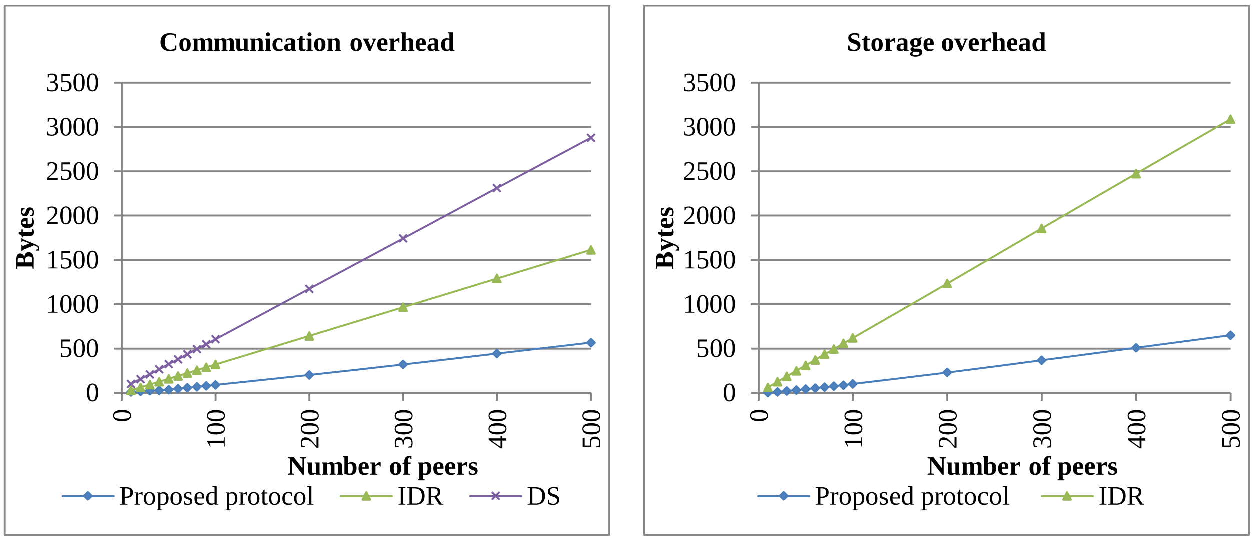


Figure 6. Communication and storage overheads in bytes for a system with delays from 50 to 550 milliseconds.

Moreover, with channel delays from 50 to 550 milliseconds [see Figure 6] and for 500 peers, each internal peer requires storing on average 650 bytes of information and the average overhead is around 560 bytes. The IDR protocol requires to store on average 3100 bytes on each peer and to send 1600 bytes while DS requires to send around 2900 bytes of overhead.

These results shows that the overhead of the proposed protocol is 2.9 times lower than the overhead of the IDR, 5.2 times lower than one of the DS, and require storing 4.7 times less information.

As our protocol adds an extra delay for the messages that arrive out of order, during each simulation, the delivery time (from sending until delivery) and the induced delay (the time message spend in the buffer) are calculated. In all simulations, the delivery delay (including the time that message spend in buffers) does not exceed the maximum delay that can be produced by the communication channels [see Figure 7]. This fact show that the presented protocol does not introduce any excessive delays for message delivery.

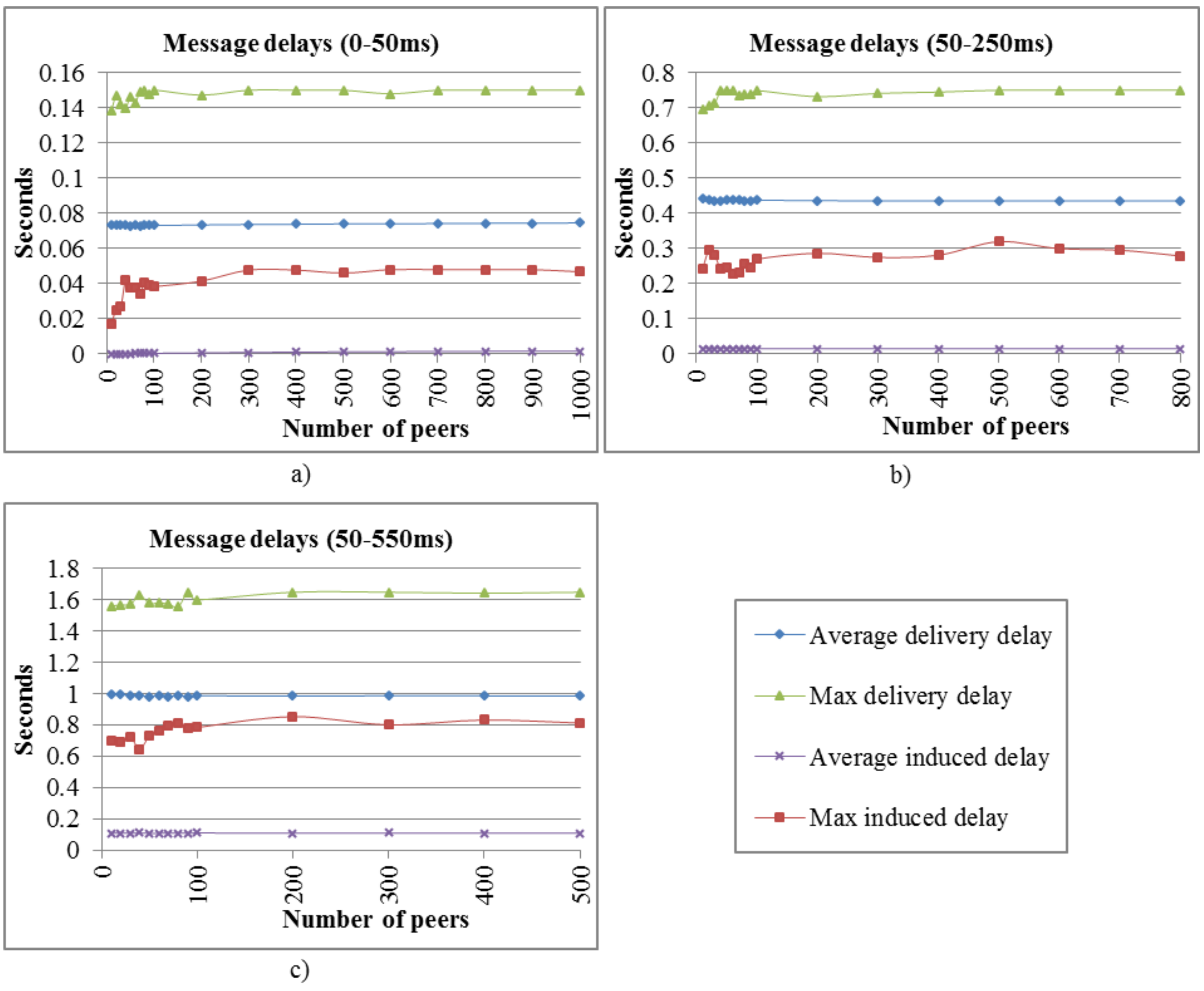


Figure 7. Delivery and induced delays for a system with delays from 0 to 50 milliseconds (a), for a system with delays from 50 to 250 milliseconds (b), for a system with delays from 50 to 550 milliseconds (c).

We can note that the maximum delay induced by the protocol (the time message spend in the buffer) is lower than the average delivery delay. When the number of peers is low (fewer messages in the system), the probability of the maximum delay during message transmission is also low, producing different results for maximum delivery and induced delays. However, when the number of peers grows (more messages in the system), the probability of the maximum delay during transmission grows producing the similar results (as at least one message have the maximum transmission delay).

6 Conclusions and future work

6.1 Conclusions

An Efficient Causal Group Communication Protocol has been presented. The proposed protocol ensures causal message ordering in a peer-to-peer hierarchical overlay network. The protocol uses a combination of the Immediate Dependency Relation and the concept of hierarchical clocks to reduce the message overhead. Thus, our protocol is efficient in terms of the overhead, piggybacked on transmitted messages. The overhead sent per message is characterized by being dynamically adapted according to the behaviour of the concurrent messages. In addition, with the use of the information about network architecture, and representing message dependencies on a bit level, our protocol ensures causal message ordering without enforcing the super peers order to all of the peers in a group. On the other hand, the presented protocol satisfies the hierarchical peer-to-peer overlay network requirements by demanding a low computational effort at the peers' side. This last is achieved performing only binary operations and simple sums. Moreover, low memory buffer is used since only a structure of bits is stored. The simulations show that our protocol produces less communication overhead than IDR and DS protocols, and also requires less storage overhead in peers. Therefore, the low overhead allows a system to include devices with limited computational capacities.

6.2 Future work

On the other hand, we note that further work is needed in order to consider different network conditions, such as loss of messages. In our protocol the IDR identifies the necessary and sufficient control information to be piggybacked on each message, to ensure the causal order in a reliable network. To support the loss of messages, some Forward Error Correction methods [28] can be applied, such as the redundancy on the transmitted control information. The purpose of adding redundancy is to increase the probability that causal order delivery will be obtained, even in the presence of lost messages and significant network delays.

Acknowledgements

Authors would like to thank the National Council of Science and Technology (Consejo Nacional de Ciencia y Tecnología, CONACYT) for its financial support throughout this research work.

References

[1] S. Tarkoma, Overlay Networks: Toward Information Networking, Auerbach, Boston, 2010.

[2] B. Cohen, The BitTorrent Protocol Specification, Version 11031 (10 January 2008), electronic version available at http://www.bittorrent.org/beps/bep_0003.html (access date: 07 July 2015).

[3] B. Knutsson, H. Lu, W. Xu, B. Hopkins, Peer-to-peer Support for Massively Multiplayer Games, Proceedings INFOCOM 2004, IEEE Publishing, 2004.

[4] S.A. Baset, H.G. Schulzrinne, An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol, Proceedings INFOCOM 2006 (25th IEEE International Conference on Computer Communications), IEEE Publishing, Barcelona, 2006, 1-11.

[5] R. Schwarz, F. Mattern, Detecting Causal Relationships in Distributed Computations: In Search of the Holy Grail, Distributed Computing, 7, 3 (1994) 149-174.

[6] G. Anastasi, A. Bartoli, A Reliable Multicast Protocol for Distributed Mobile Systems: Designs and Evaluation, IEEE Transactions on Parallel and Distributed Systems, 12, 10 (2001) 1009-1022.

[7] S.M. Banik, S. Radhakrishnan, T. Zheng, C.N. Sekharan, Distributed Floor Control Protocols for Computer Collaborative Applications on Overlay Networks, Proceedings International Conference on Collaborative Computing: Networking, Applications and Worksharing, IEEE Publishing, San Jose, 2005.

[8] C. Benzaid, N. Badache, An Optimal Causal Broadcast Protocol in Mobile Dynamic Groups, Proceedings International Symposium on Parallel and Distributed Processing with Applications, IEEE Publishing, Sydney, 2008, 477-484.

[9] D. Doulikun, A. Aikebaier, T. Enokido, M. Takizawa, Experimentation of Group Communication Protocols, Proceedings 16th International Conference on Network-Based Information Systems (NBiS), IEEE Publishing, Gwangju, 2013, 476-481.

[10] R. Friedman, S. Manor, Causal Ordering in Deterministic Overlay Networks, Technical Report, Israel Institute of Technology, Haifa, Israel, 2004.

[11] C. Hsiao, Y. Liao, Domain-Based Causal Ordering Group Communication in Wireless Hybrid Networks, Proceedings The 5th International Conference on Ubiquitous Information Management and Communication, ACM Publishing, New York, 2011, 131:1-131:6.

[12] A. Maddi, F. Dahamni, An Efficient Algorithm for Causal Messages Ordering, Proceedings The 2001 ACM Symposium on Applied Computing, ACM Publishing, Las Vegas, 2001, 499-503.

[13] T. Nishimura, N. Hayashibara, T. Enokido, M. Takizawa, Causally Ordered Delivery with Global Clock, IEEE, 2005, 560-564.

[14] S. E. Pomares Hernández, The Minimal Dependency Relation for Causal Event Ordering in Distributed Computing, Applied Mathematics & Information Sciences, 3, 5 (2015), 57-61.

[15] R. Prakash, M. Singhal, Dependency Sequences and Hierarchical Clocks: Efficient Alternatives to Vector Clocks for Mobile Computing Systems, Wireless Networks, 3, 5 (1997), 349-360.

[16] K. Taguchi, M. Takizawa, Two-Layered Protocol for a Large-Scale Group of Processes, Proceedings Ninth International Conference on Parallel and Distributed Systems, IEEE Publishing, 2002, 171-176.

[17] I. Tsuneizumi, A. Aikebaier, T. Enokido, M. Takizawa, A Flexible Group Communication Protocol with Hybrid Clocks, Proceedings The 7th International Conference on Advances in Mobile Computing and Multimedia, ACM Publishing, New York, 2009, 469-474.

[18] I. Tsuneizumi, A. Aikebaier, M. Ikeda, T. Enokido, M. Takizawa, A Scalable Hybrid Time Protocol for a Heterogeneous Group, Proceedings International Conference on Broadband, Wireless Computing, Communication and Applications, Fukuoka, 2010, 214-221.

[19] I. Tsuneizumi, A. Aikebaier, T. Enokido, M. Takizawa, A Scalable Peer-to-Peer Group Communication Protocol, Proceedings 24th IEEE International Conference on Advanced Information Networking and Applications, IEEE Publishing, Perth, 2010, 268-275.

[20] I. Tsuneizumi, A. Aikebaier, M. Ikeda, T. Enokido, M. Takizawa, S.M. Deen, Hybrid Clock-Based Synchronization in a Scalable Heterogeneous Group, Proceedings 13th International Conference on Network-Based Information Systems, IEEE Publishing, Takayama, 2010, 246-253.

[21] I. Tsuneizumi, A. Aikebaier, T. Enokido, M. Takizawa, Reduction of Messages Unnecessarily Ordered in Scalable Group Communication, Proceedings International Conference on Complex, Intelligent and Software Intensive Systems, IEEE Publishing, Krakow, 2010, 299-306.

[22] L. Yen, Probabilistic Analysis of Causal Message Ordering, Proceedings Seventh International Conference on Real-Time Computing Systems and Applications, IEEE Publishing, Cheju Island, 2000, 409-413.

[23] S. Zhou, W. Cai, S.J. Turner, B. Lee, J. Wei, Critical Causal Order of Events in Distributed Virtual Environments, ACM Transactions on Multimedia Computing, Communications, and Applications, 3, 3 (2007).

[24] Lamport, L.: "Time, Clocks, and the Ordering of Events in a Distributed System", Communication of the ACM, 21, 7 (1978), 558-565.

[25] F. Mattern, Virtual Time and Global States of Distributed Systems, Proceedings Parallel and Distributed Algorithms, North-Holland Publishing, 1988, 215-226.

[26] B. Yang, H. Garcia-m, Designing a Super-peer Network, Proceedings IEEE International Conference on Data Engineering, 2003.

[27] OMNeT++ Discrete Event Simulator, https://omnetpp.org/. Access date: July 23, 2015.

[28] E. Lopez Dominguez, S. E. Pomares, G. Rodriguez, Maria A. Medina, An Efficient Causal Protocol with Forward Error Correction for Mobile Distributed Systems, Journal of Computer Science, 6, 7 (2010), 756-768.