An Efficient Causal Group Communication Protocol for Free Scale Peer-to-Peer Networks

Grigory Evropeytsev, Eduardo López Domínguez, Saúl Eduardo Pomares Hernández, José Perez Cruz

An Efficient Causal Group Communication Protocol for Free Scale Peer-to-Peer Networks

Grigory Evropeytsev [1], Eduardo López Domínguez [1,*], Saul E. Pomares Hernandez [2] and José Roberto Perez Cruz [3]

[1] Laboratorio Nacional de Informática Avanzada (LANIA), Xalapa 91000, México; grigory88@live.com
[2] Computer Science Department, Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), Puebla 72840, México; spomares@inaoep.mx
[3] CONACYT—FIC, Universidad Michoacana de San Nicolás de Hidalgo (UMSNH), Morelia 58040, México; jrperezcr@conacyt.mx
* Correspondence: elopez@lania.mx; Tel.: +52-228-841-6100

Abstract: In peer-to-peer (P2P) overlay networks, a group of n (≥2) peer processes have to cooperate with each other. Each peer sends messages to every peer and receives messages from every peer in a group. In group communications, each message sent by a peer is required to be causally delivered to every peer. Most of the protocols designed to ensure causal message order are designed for networks with a plain architecture. These protocols can be adapted to use in free scale and hierarchical topologies; however, the amount of control information is O(n), where n is the number of peers in the system. Some protocols are designed for a free scale or hierarchical networks, but in general they force the whole system to accomplish the same order viewed by a super peer. In this paper, we present a protocol that is specifically designed to work with a free scale peer-to-peer network. By using the information about the network's architecture and by representing message dependencies on a bit level, the proposed protocol ensures causal message ordering without enforcing super peers order. The designed protocol is simulated and compared with the Immediate Dependency Relation and the Dependency Sequences protocols to show its lower overhead.

Keywords: causal ordering; peer-to-peer; free scale topology

1. Introduction

Multiparty overlay systems, such as peer-to-peer, are technologies that can solve problems in information distribution and processing, in areas including file distribution [1,2], multiplayer interactive games [3] and telecommunication [4]. These systems are characterized by considering a group communication between n (≥2) peers that are distributed geographically. To achieve a consistent exchange of information in this environment, messages from the peers have to be causally ordered [5]. The use of causal ordering in the overlay peer-to-peer systems provides message synchronization and reduces the indeterminism produced by asynchronous execution of the peers and by random and unpredictable delays of the messages in the communication channels. Causal ordering guarantees that the actions (requests or questions) are delivered before corresponding reactions (results or answers).

Some works have proposed protocols to guarantee causal ordering delivery of the messages [6–27]. These protocols can be classified into three categories. The first category includes the protocols designed for plain peer-to-peer networks. We can classify [12,14,24,25] in this category. One disadvantage of these protocols is that the size of the control information in the worst case scenario grows with the number of peers in the network. In addition, the protocols from this category add additional processing to the peers that also depends on the number of peers in the system. The second category includes protocols that are designed for free scale peer-to-peer networks. In the case of this topology, the peers can be separated into two groups: peers and super peers. Super peers are characterized by higher processing powers and/or bandwidth. This network topology was designed to solve some problems of the plain P2P networks, such as bottlenecks caused by the very limited capabilities of some peers. Free scale peer-to-peer networks take advantage of system heterogeneity, assigning greater responsibility to those who are more capable of handling it [28]. The similar topology can be seen in the Virtual Private Network (VPN) technology [29], where Intranet computers can communicate directly between each other but the remote clients communicate through the VPN server. However, up to date there are no protocols to guarantee causal ordering delivery of the messages, specifically designed for such networks. To overcome the lack of dedicated solutions for these networks, some protocols for plain topology can be adapted

by establishing the causal order under each super peer. Nevertheless, besides the problems related to the uncontrolled overhead growth, these approaches may induce unnecessary inhibition on the delivery of messages.

The third category incorporates all the protocols that are designed for the hierarchical networks [7,8,11,13,15–18,26,27], 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 to the 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, serious overloads in super peers' channels be caused in dense networks. Besides this approach, there are some protocols specifically designed for hierarchical networks [13,17,18] which are based on the use of a global time reference to ensure causal ordering of messages. Unfortunately, the characteristics of most hierarchical networks make it difficult to establish a global time reference; this is mainly due to the absence of shared memory and the lack of perfectly synchronized clocks [30]. 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 present the design of a protocol that ensures causal ordering of the messages in the overlay networks with free scale topology without using the global time reference and maintaining the low overhead in the communication channels and in the peers. To achieve this, our causal protocol works on two communication groups according to their connection type. An internal group consists of peers that are connected to a super peer. This group uses a bit vector to represent message dependencies producing overhead on a bit level. Another group is an external group consisting of interconnected peers. This group uses an idea of hierarchical clocks [15] to represent message dependencies. The results obtained via a simulation demonstrate that the overhead in an internal group is 3.5 times lower than the overhead from the Immediate Dependency Relation protocol [14] while the overhead in the external group can be compared to the overhead in the Immediate Dependency Relationship protocol, but it is half the overhead of the Dependency Sequences protocol.

2. Materials and Methods

This section describes the system model as well as the concepts and definitions used in the protocol presented in this work.

2.1. System Model

A free scale peer-to-peer network consists of peers that are connected only to super peers and peers that have a direct connection with other peers. Based on this, a free scale peer-to-peer network can be divided into two groups (see Figure 1):

1. Internal group (internal peers that are connected only a super peer); and
2. External group (interconnected external peers).

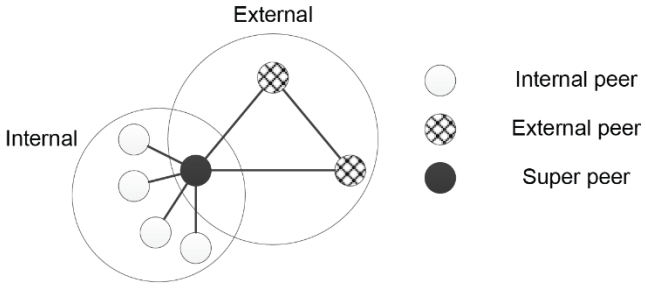


Figure 1. Internal and external groups of a free scale P2P (peer-to-peer) network.

Peers that form an internal group are called internal peers. Peers in an external group are called external peers (see Figure 1). Peers in a free scale network can only belong to one group (internal or external). A super peer is a member of both groups at the same time. An internal group can have only one super peer, while an external group can consist of several.

A super peer is a special node with major processing and bandwidth capacities in comparison to peers. Peers in an internal group are considered to have lower processing power or bandwidths compared to external group peers. Peers in an internal group can be represented by mobile phones and tablets, among others, connected by a 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 in an internal group and as a meta-process representing all of the events in the external group for peers in the internal group. With this representation, 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. In addition, the channels are considered to be non FIFO (First In, First Out) channels. This implies that messages can be reordered by a communication channel.

2.2. 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 [30]. 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 three following conditions:*

If a and b are events in the same process, and a comes before b, then $a \rightarrow b$.

If a is the sending of a message by one process and b is the receipt of the same message by another process, then $a \rightarrow b$

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 $a \nrightarrow b$ and $b \nrightarrow a$. 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.

2.2.1. The Immediate Dependency Relation (IDR)

The Immediate Dependency Relation [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].

2.2.2. Process and Meta-Process

Definition 3. *A single process is defined to be a totally ordered set of events [30]. 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. Thus, if a and b are events in the same meta-process, and a comes before b, this does not mean that $a \rightarrow b$.*

3. Results

3.1. Protocol Composition

3.1.1. 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 document.

Bit Vector

A 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 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 ($\bar{}$) operations that are bitwise, i.e., the operation is applied to bits at Position 1, then bits at Position 2, etc.

Extended Linear Time

Extended Linear Time (*LTx*) is a data type that can contain one of the following: An integer number and a bit vector. Extended Linear Time cannot contain an integer and a bit vector at the same time. In addition, 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.

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 time.

Data Structures in an Internal Peer

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*—an integer representing a sequence number of a message.
- *RV*—bit vector representing received messages.
- *DV*—bit vector representing message IDR.

Data Structures in an External Peer

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*.

Data Structures in a Super Peer

A super peer maintains the following variables:

- $id_{ext}(sp)$—identifier of a super peer in the external group.
- *VTx*(*sp*)—extended vector time. The size of a vector is *G*, where *G* is the number of peers and super peers in an external group.
- *I*—bit vector representing identifiers of translated messages.
- *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. The size of a vector is *G*. Each pair consists of two message identifiers called *in* and *out*.

Internal Group Message Structure

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

$$m_{int} = (id, SN, Last, DV, Data) \quad (1)$$

where

- *Id*—the identifier of a sending process in the internal group.
- *SN*—an integer representing a message sequence number.
- *Last*—an integer identifier of the last message from this peer.
- *DV*—a bit vector representing message dependency.
- *Data*—the user data to be transmitted.

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

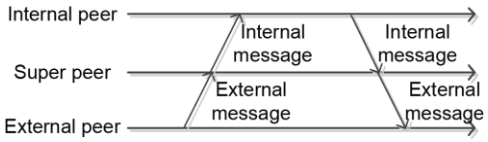


Figure 2. Messages in a free scale peer-to-peer network.

External Group Message Structure

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

$$m_{ext} = (id, SN, CI, I, Data) \quad (2)$$

where

- *Id*—the identifier of a sending process.
- *SN*—an integer representing a message sequence number.
- *CI*—vector of pairs representing message control information. Each pair consists of a process identifier and *LTx*.
- *I*—bit vector with internal message identifiers.
- *Data*—the user data to be transmitted.

This message structure is used by both peers and super peer in an external group (see Figure 2).

3.1.2. Specification of the Causal Protocol

Initially each process performs an initialization process. An internal peer performs this process as follows.

Listing 1. Internal peer initialization.

$SN := 0$
$RV := Ø$
$DV := Ø$

An external peer initializes its variables as follows.

Listing 2. External peer initialization.

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

$CI := Ø$

Furthermore, a super peer performs initialization as follows (Listing 3).

Listing 3. Super peer initialization.

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

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

$TT := (Ø, Ø, \ldots, Ø)$

$I := Ø$

When an internal peer wants to send a message to a super peer, it constructs the message using the following procedure (see Listing 4).

Listing 4. Message sending by an internal peer.

```
1   SN := SN + 1
2   m_int = (id_int, SN, 0, DV, Data)
3   DV := Ø
```

When an internal peer receives a message from a super peer, it verifies its delivery condition. First, it should verify a FIFO condition that a received message has not arrived before a previous message from the same sender. After such verification, a peer needs to check that it has received all of the messages that form the immediate dependency relation with the currently received message (see Listing 5).

Listing 5. Internal peer delivery condition.

```
1   If (RV[m_int.Last] = 1) then    // FIFO condition
2       If (m_int.DV & RV = m_int.DV) then    // Causal condition
3           Deliver(m_int)
4       End if
5   End if
```

If a delivery condition is satisfied, a peer updates its data structures (see Listing 6) and it can deliver a message to a corresponding application. If a delivery condition is not satisfied, a message should be buffered.

An internal peer will receive its own message returned by a super peer. In this case, it 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 (see Listing 6).

Listing 6. Message delivery to internal peer.

```
1   RV[m_int.SN] := 1
2   If (m_int.id ≠ id_int) then
3       DV := DV & ¬m_int.DV
4       DV[m_int.SN] := 1
5       DV[m_int.Last] := 0
6   End if
```

Since a super peer only transmits messages from other peers, it does not have a message emission phase. When a super peer receives a message from an internal peer, it checks the message delivery condition. Since an internal peer can only receive messages from a super peer, it means that only a FIFO dependency needs to be checked. To check the message delivery condition, a super peer uses the following algorithm (see Listing 7).

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

```
1   If (m_int.SN = LR[m_int.id].in + 1) then
```

2 Deliver(m_{int})

3 End if

If a delivery condition is satisfied, a super peer can forward this message to other peers (see Listing 8). If a delivery condition is not satisfied, a message should be buffered.

Listing 8. Internal message delivery to super peer.

1 $m_{int}.Last := LR[m_{int}.id].out$

2 $VTx[id_{ext}] := VTx[id_{ext}] + 1$

3 $LR[m_{int}.id] := <m_{int}.SN, VT[id_{ext}]>$

4 $m_{int}.SN := VTx[id_{ext}]$

5 Send m_{int} to all peers in the internal group

6 Transform and send message m_{int} to all peers in the external group

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 (see Listing 8). A message transformation will be discussed later in this section.

When an external peer wants to send a message, it constructs it using the algorithm (see Listing 9).

Listing 9. Message sending by external peer.

1 $VTx[id_{ext}] := VTx[id_{ext}] + 1$

2 $m_{ext} = (id_{ext}, VTx[id_{ext}], CI, Ø, Data)$

3 $CI := Ø$

When a message is received by an external peer or a super peer, it requires updating its clock with the internal message identifiers that a message is carrying in its field $m_{ext}.I$ (see Listing 10).

Listing 10. Message receiving by peers and super peers in an external group.

1 If $VTx[m_{ext}.id]$ is a bit vector then

2 $VTx[m_{ext}.id] = VTx[m_{ext}.id] \mid m_{ext}.I$

3 End if

After this update, a message delivery condition can be checked. A message delivery condition consists of two parts. If a message arrived from an external peer, a message FIFO order should be checked. If the message FIFO ordering is not violated or a message has arrived from a super peer, then a message causal delivery condition is checked. This condition consists in checking message identifiers that are inside messages control information (see Listing 11). These conditions (FIFO and causal) are checked in both external peers and a super peer when it receives a message from an external group.

Listing 11. Message delivery condition in external group.

1 If ($VTx[m_{ext}.id]$ is a bit vector) or ($m_{ext}.SN = VTx[m_{ext}.id] + 1$) then // FIFO

2 If for each $<i, dep>$ ($i \neq id_{ext}$) in

3 $m_{ext}.CI => \begin{cases} dep \leq VTx[i] & \text{,if } VTx[i] \text{ is an integer} \\ dep \ \& \ VTx[i] = dep & \text{,if } VTx[i] \text{ is a bit vector} \end{cases}$ then // Causal

Deliver(m_{ext})

4 End if

5 End if

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 (see Listing 12). This update consists of two parts. First, it needs to update the clock so that the 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.

Listing 12. Message delivery to an external peer.

```
1   If VTx[m_ext.id] is an integer then
2       VTx[m_ext.id] := m_ext.SN
3   Else
4       VTx[m_ext.id][m_ext.SN] := 1
5   End if
6   If VTx[m_ext.id] is an integer then
7       If exists <i, dep> in CI that i = m_ext.id then
8           CI := CI \ <i, dep>
9       End if
10      CI := CI ∪ <m_ext.id, m_ext.SN>
11  Else
12      If not exists <i, dep> in CI that i = m_ext.id then
13          CI := CI ∪ <m_ext.id, Ø>
14      End if
15      CI[m_ext.id][m_ext.SN]: = 1
16  End if
17  For each <i, dep> in m_ext.CI
18      If VTx[i] is an integer then
19          CI := CI \ <i, dep>
20      Else
21          CI[i] := CI[i] & ¬dep
22      End if
23  End for
```

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 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.

Listing 13. External message delivery to a super peer.

```
1   If VTx[m_ext.id] is an integer then
2       VTx[m_ext.id] := m_ext.SN
3   Else
4       VTx[m_ext.id][m_ext.SN] := 1
5   End if
6   VTx[id_ext] := VTx[id_ext] + 1
7   Transform and send message m_ext to all peers in the internal group
```

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 (see Listing 12 for external peer, and Listing 13 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. Thus, 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. Thus, it is only required to store bits between the first and the last set bits.

To be completely functional, a 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 an 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, and message dependency). This transformation can be achieved by using the algorithm presented below (see Listing 14).

Listing 14. Message transformation from an internal to an external group.

```
1   I_m := I     // Bit vector
2   I := Ø
3   CI := (<id_ext, m_int.DV>)     // Vector of pairs.
4   CI[id_ext][m_int.Last] := 1
5   For each <id, PT> in TT     // PT is a vector of pairs
6       For each <in, out> in PT in reverse order
7           If m_int.DV[out] = 1 then
8               I_m[out] := 1
9               CI[id_ext][out] := 0
10              If VTx[id] is an integer then
11                  CI := CI ∪ <id, in>
12                  Exit For
13              Else
14                  If not exists <i, dep> in CI that i = id then
15                      CI := CI ∪ <id, Ø>
16                  End if
17                  CI[id][in] := 1
18              End if
19          End if
20      End for
21  End for
22  m_ext := (id_ext, m_int.SN, CI, I_m, m_int.Data)
```

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*.

Listing 15. Message transformation from an external to an internal group.

```
1   DV := m_ext.CI[id_ext]     // Bit vector
2   For each <id, dep> in m_ext.CI
3       If (VTx[id] is an integer)
4           If (exists <in, out> in TT[id] where in = dep) then
5               DV[out] := 1
6           End if
7       Else
8           For each 1 in dep in position i
9               If exists <in, out> in TT[id] where in = i then
10                  DV[out] := 1
```

```
11                  End if
12              End for
13          End if
14  End for
15  m_int = (0, VTx[id_ext], 0, DV, m_ext.Data)
16  If TT[m_ext.id] is not empty and VTx[m_ext.id] is an integer then
17          m_int.Last := TT[m_ext.id][last].out
18  End if
19  TT[m_ext.id] := TT[m_ext.id] ∪ <m_ext.SN, VTx[id_ext]>
20  I[VTx[id_ext]] := 1
```

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, and message dependency), but to be interpreted correctly in an internal group, they should be transformed to a bit vector form (see Listing 15). In addition, a super peer can receive messages that contain dependencies on messages that are not yet received. To deal with this, a super peer checks the message delivery condition as described previously to ensure that this message can be delivered by 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. Thus, a message m' that requires m to be delivered before it cannot be delivered to any peer in an internal group.

3.1.3. Correctness Proof

To show that our algorithm ensures the causal delivery (correctness), we provide a correctness proof. In order to do the proof as simple as possible, we focus on the part of the internal group where the message dependencies are represented on a bit level. In this proof we will use the following notations:

- m_k: message with $SN = k$.
- p_j: peer with identifier j.

We show that with this information we ensure the causal order.

Theorem 1. *For each set bit in $m_l.DV[k]$, k identifies a message from the same group or TT contains (in, out) at position id such that (k = out), where (id, in) identifies a message m_k, which has IDR with m_l.*

Main steps of the proof. The proof is composed of three lemmas and a proposition. The lemmas are intermediate results necessary for our proof:

- Lemma 1 shows that if $m_l.DV[k]$ is set, then the message with $SN = k$ causally precedes message m_l.
- Lemma 2 shows that if $m_l.Last = k$, then message with $SN = k$ causally precedes message m_l.
- Lemma 3 indicates that the message m_k has an immediate dependency relation with the other message m_l if and only if bit $m_l.DV[k]$ is set.
- Proposition 1 shows that through the bits structure $h(m)$ attached to messages sent and the causal information at the super peer, we ensure the causal order (Theorem 1).

Lemma 1.

If $m_l.DV[k]$ is set then message $m_k \rightarrow m_l$

Proof: By Line 2 in Listing 4, we have that $m_l.DV[k]$ is set if and only if $DV[k]$ at peer p_j is set when the sending of m_l is carried out by p_j. By using Line 4 in Listing 6, we have that $DV[k]$ is set only after the delivery of m_k. In addition, the super peer does not make any modifications to the DV of a message. This implies that the delivery of m_k precedes the sending of m_l ($delivery(m_k) \rightarrow send(m_l)$). Therefore, $m_k \rightarrow m_l$.

Lemma 2.

If $m_l.Last = k$ then message $m_k \rightarrow m_l$

Proof. By Line 1 of Listing 8, we have that $m_l.Last$ is set to the value of $LR[m_l.id].out$ at the super peer. By Line 3 of Listing 8, $LR[m_l.id].out$ is set to k upon send of m_k which was received by super peer from peer with id $m_l.id$. This means that m_k and m_l were sent by the same peer and therefore $m_k \rightarrow m_l$.

Lemma 3.

$m_k \downarrow m_l$ if and only if $m_l.DV[k]$ is set

The proof to this lemma is divided into two steps: First, we show that if $m_k \downarrow m_l$ then $m_l.DV[k]$ is set and second, we show that if $m_l.DV[k]$ is set, then $m_k \downarrow m_l$.

Step 1:

If $m_k \downarrow m_l$ then $m_l.DV[k]$ is set

The proof is by contrapositive. We proof that if $m_l.DV[k]$ is cleared then m_r exists such that $m_k \rightarrow m_r \rightarrow m_l$; thus, the message m_k does not have an immediate dependency relation with message m_l. We assume that $m_l.DV[k]$ is cleared. Only two events can clear $DV[k]$ before sending m_l, these are:

- By Lines 3, 5 of Listing 6, $DV[k]$ is cleared when the delivery of message m_r is carried out with $m_r.DV[k]$ bit set or $m_r.Last = k$. Lemmas 1 and 2 show that if $m_r.DV[k]$ is set or $m_l.Last = k$ then $m_k \rightarrow m_l$. Moreover, $delivery(m_r) \rightarrow send(m_l)$ implies that $m_r \rightarrow m_l$ and then $m_k \rightarrow m_r \rightarrow m_l$. Therefore, m_k does not directly precede message m_l.
- By Line 3 of Listing 4, the sending of m_r empties DV. In addition, the event of $send(m_r)$ takes place such that $delivery(m_k) \rightarrow send(m_r) \rightarrow send(m_l)$. Therefore, m_k does not directly precede message m_l.

If neither of these two events occur, we have that $DV[k]$ is set when the $send(m_l)$ is carried out and by Line 1 of Listing 4, we have that $m_l.DV[k]$ is set.

Step 2:

If $m_l.DV[k]$ is set, then $m_k \downarrow m_l$.

The proof is by contradiction. By Lemma 1, we know that if $m_l.DV[k]$ is set then $m_k \rightarrow m_l$ with $p_i \neq p_j$. We suppose that there is a message m_r such that $send(m_k)$ by $p_i \rightarrow send(m_r)$ by $p_r \rightarrow \text{send}(m_l)$ by p_j, and in addition that $m_k \downarrow m_r$. The proof considers two cases: $p_r \neq p_j$ and $p_r = p_j$.

- We consider the case where $p_r \neq p_j$ and the delivery m_k causally precedes m_r ($delivery(m_k) \rightarrow delivery(m_r)$) at p_j. By step 1, we know that $m_r.DV[k]$ is set. Hence, on the delivery m_r ($delivery(m_r)$) at p_j, $DV[k]$ is cleared Line 3 of Listing 6. When performing the sending of m_l ($send(m_l)$) and because of $delivery(m_r) \rightarrow send(m_l)$, then $DV[k]$ at peer p_j is cleared and therefore, $m_l.DV[k]$ is cleared, which is a contradiction.
- In the case where $p_r = p_j$, we have that $delivery(m_k) \rightarrow send(m_r) \rightarrow send(m_l)$, because the sending of m_r ($send(m_r)$) takes place, $DV[k]$ is cleared by Line 3 of Listing 4. Therefore, we have that $m_l.DV[k]$ is cleared, which is a contradiction.

Finally, the following proposition shows that through the bits attached to the sent messages and the information stored at the super peer, we ensure the causal order in the overlay peer-to-peer network.

Proposition 1.

If $m_l.DV[k]$ is set and m_k is a message from the external group then TT contains (in, k).

Proof. By Line 19 of Listing 15, we have that (in, k') is inserted into $TT[m_{k'}.id]$ only after the delivery of message $m_{k'}$ at the super peer where $(m_{k'}.id, in)$ identifies a message in the external group. The corresponding internal message has SN equals to k'. In the delivery of $m_{k'}$ at p_j with $SN = k'$, we have (by Line 4 of Listing 6) that $DV[k']$ is set. We know by Lemma 3 that if $m_l.DV[k']$ is set then $m_{k'} \downarrow m_l$. On the reception of message m_l sent by p_j with $m_l.DV[k']$ set at the super peer, by Lines 5–21 of Listing 14, we have that TT contains an element (in', out') where $out' = k'$ at position $m_{k'}.id$ which identifies a message $(m_{k'}.id, in')$ in the IDR protocol of the external group.

3.1.4. Causal Protocol Description

To describe 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 peers with identifiers in an internal group 1 and 2, denoted as $P(i)_1$ and $P(i)_2$.

- A super peer with an identifier in an external group 1, denoted as Sp_1.
- Two external peers with identifiers in an external group 2 and 3, denoted as $P(i)_2$ and $P(i)_3$.

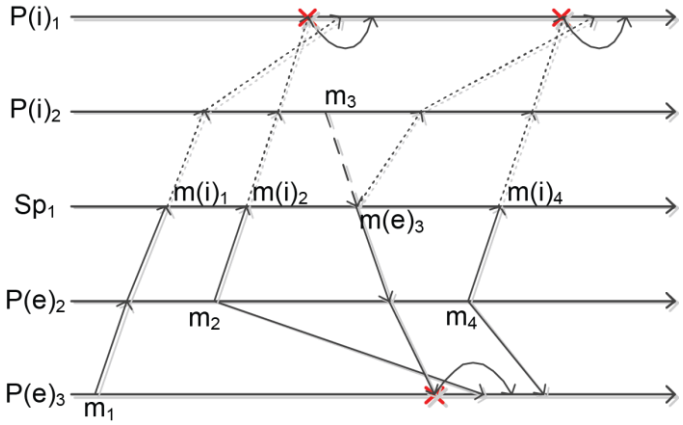


Figure 3. Scenario example.

This scenario contains three 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 dashed 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.

Diffusion of m_3 at $P(i)_2$

- First, an internal peer increments its sequence number to 1 (see Listing 4, Line 1).
- An internal peer generates a message m_3 = (2, 1, 0, 01, *Data*) (see Listing 4, Line 2).
- The *DV* variable is cleared (see Listing 4, Line 3).

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 (see Listing 7, Line 1): 1 = 0 + 1 and message m_3 can be delivered.
- m_3.*Last* field is changed to 0 (*LR*[2].*out* = 0) (see Listing 8, Line 1).
- A clock for this super peer is updated with a new value (see Listing 8, Line 2): *VTx*(*p*) = (3, 1, 1), and *LR* is updated with message identifiers (see Listing 8, Line 3): *LR*[2] = <1, 3>.
- m_3.*SN* is updated to 3, and m_3 is sent to all peers in an internal group.

Transformation of m_3 at Sp_1 to External Group

- I_m = 11 and *I* is set to Ø (see Listing 14, Lines 1–2), and *CI* is created with value <1, 01> and *TT*[2] contains <1, 2> (see Listing 14, Line 7).
- *CI* element <1, 01> has its bit 2 cleared (see Listing 14, Line 9): <1, Ø>, and this dependency is removed. *CI* is updated with message dependency (see Listing 14, Line 11): CI = <2, 1> and, m_3 is transformed to an external group: $m(e)_3$ = (1, 3, <2, 1>, 11, m_3.*Data*).

Reception of $m(e)_3$ at $P(e)_3$

- When a peer $P(e)_3$ receives $m(e)_3$, the process clock is updated *VTx*(*p*) = (11, 0, 1).
- Then the delivery condition is checked.
  - The FIFO delivery condition is not checked because the message arrived from a super peer.
  - $m(e)_3$.*CI* contains <2, 1> and but *VTx*[2] = 0, so this dependency is not satisfied.
  - The message delivery condition is not satisfied; therefore, the message $m(e)_3$ should be buffered.

Reception of m_2 at $P(e)_3$

- When a peer $P(e)_3$ receives m_2, it checks its delivery condition.

- The FIFO delivery condition is satisfied (see Listing 11, Line 1): 1 = 0 + 1 and m_2.*CI* contains <3, 1>, and this dependency is satisfied because id_{ext} = 3. Both conditions are satisfied, and the message m_2 can be delivered.

- A clock component for peer 2 is updated with a new value (see Listing 12, Line 2): *VTx*(*p*) = (11, 1, 1) and *CI* is empty, so a new message dependency is inserted (see Listing 12, Line 10): *CI* = <2, 1>.
- Message buffer contains $m(e)_3$. Its delivery condition should be revalidated.
  - The FIFO delivery condition is not checked because the message arrived from a super peer.
  - $m(e)_3$.*CI* contains <2, 1> and *VTx*[2] is an integer, so the causal condition is satisfied (see Listing 11, Line 2): $1 \leq 1$.
  - Both conditions are satisfied, and the message $m(e)_3$ can be delivered.
- A clock component for super peer (id_{ext} = 1) is updated with a new value (see Listing 12, Line 2): *VTx*(*p*) = (111, 1, 1).
- The *CI* does not contain an element for peer 1, so a new message dependency is inserted (see Listing 12, Line 10): *CI* = <1, 001>, <2, 1>.
- $m(e)_3$.*CI* contains <2, 1>, so this pair is removed from *CI* (see Listing 12, Line 19): *CI* = <1, 001>

Diffusion of m_4 at $P(e)_2$

- First, an external peer increments its clock (see Listing 9, Line 1): *VTx*(*p*) = (111, 2, 1). An external peer generates a message m_4 = (2, 2, <1, 001>, Ø, *Data*) (see Listing 9, Line 2), and the *CI* is cleared (see Listing 9, Line 3).

Reception of m_4 at $P(e)_3$

- When a peer $P(e)_3$ receives m_4, it checks its delivery condition.
  - The FIFO delivery condition is satisfied (see Listing 11, Line 1): 2 = 1 + 1 and m_4.*CI* contains <1, 001> and *VTx*[1] is a bit vector, so this dependency is satisfied (see Listing 11, Line 2): 001 & 111 = 001. Both conditions are satisfied, and the message m_4 can be delivered.
- A clock component for peer 2 is updated with a new value (see Listing 12, Line 2): *VTx*(*p*) = (111, 2, 1).
- The *CI* does not contain an element for peer 2, so a new message dependency is inserted (see Listing 12, Line 10): *CI* = <1, 001>, <2, 2>.
- m_4.*CI* contains <1, 001> so this pair is modified (see Listing 12, Line 21): *CI* = <1, 000>, <2, 2>, and empty dependency <1, 000> is removed, so *CI* = <2, 2>.

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 (see Listing 11, Line 1): 2 = 1 + 1. m_4.*CI* contains <1, 001> and this dependency is satisfied because id_{ext} = 1. Both conditions are satisfied and the message m_4 can be delivered.
- A clock component for peer 2 (see Listing 13, Line 29) and for this super peer (see Listing 13, Line 6) are updated with a new value: *VTx*(*p*) = (4, 2, 1).

Transformation of m_4 at Sp_1 to Internal Group

- As m_4.*CI* contains an element for process 1 (see Listing 15, Line 1) *DV* is initialized to 001. In this case, m_4.*CI* does not contain other dependencies so m_4 is transformed to an internal group: $m(i)_4$ = (0, 4, 0, 001, *Data*) and *TT*[2] contains <1, 2> (see Listing 15, Line 16).
- $m(i)_4$.*Last* is updated to 2: $m(i)_4$ = (0, 4, 2, 001, *Data*) and *TT* is updated with message identifiers before and after transformation (see Listing 15, Line 19): *TT*[2] = <1, 2>, <2, 4>
- A bit for internal message is set in variable *I* (see Listing 15, Line 20): *I* = 0001.

Reception of $m(i)_4$ at $P(i)_2$

- When a peer $P(i)_2$ receives $m(i)_4$, it checks its delivery condition.
  - The FIFO delivery condition is satisfied (see Listing 5, Line 1): *RV*[2] = 1, and the causal delivery condition (see Listing 5, Line 2) is also satisfied: 001 & 111 = 001.
  - Both conditions are satisfied, and the message $m(i)_4$ can be delivered.

- The receive vector is updated (see Listing 6, Line 1): RV = 1111.
- The message dependency vector is updated as well (see Listing 6, Lines 3–5): DV = 0001.

Reception of $m(i)_4$ at $P(i)_1$

- When a peer $P(i)_1$ receives $m(i)_4$, it checks its delivery condition.
  - The FIFO delivery condition is satisfied (see Listing 5, Line 1): $RV[2] = 1$, but the causal delivery condition (see Listing 5, Line 2) is violated: 001 & 11 = Ø ≠ 001. Therefore, the message delivery condition is not satisfied, and message $m(i)_4$ should be buffered.

Reception of $m(i)_3$ at $P(i)_1$

- When a peer $P(i)_1$ receives $m(i)_3$, it checks its delivery condition.
  - The FIFO delivery condition is satisfied (see Listing 5, Line 1): $RV[0] = 1$. The causal delivery condition (see Listing 5, Line 2) is also satisfied: 01 & 11 = 01. Both conditions are satisfied, and the message $m(i)_3$ can be delivered.
- The receive vector is updated (see Listing 6, Line 1): RV = 111.
- The message dependency vector is updated as well (see Listing 6, Lines 3–5): DV = 001.
- The message buffer contains $m(i)_4$. Its delivery condition should be revalidated.
  - The FIFO delivery condition is satisfied (see Listing 5, Line 1): $RV[2] = 1$, and the causal delivery condition (see Listing 5, Line 2) is also satisfied: 001 & 111 = 001. Both conditions are satisfied, and the message $m(i)_4$ can be delivered.
- The receive vector is updated (see Listing 6, Line 1): RV = 1111.
- The message dependency vector is updated as well (see Listing 6, Lines 3–5): DV = 0001.

3.1.5. Overhead Analysis

As the proposed protocol depends on the Immediate Dependency Relation protocol [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 n-1 messages (n is the number of processes in a system), its dependency vector cannot contain more than the n-1 set bits. However, the set bits can be separated by cleared bits. Let m_1 be the message with the lowest sequence number to form an IDR with m. Then a bit vector can have no more bits than the number of messages concurrent to m_1 that exist 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 it is proportional to a number of processes in a system producing an overhead of O(n) bits.

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). As vector I in the external message, its size is limited by the number of messages that can be generated in the external and internal groups producing an overhead of vector I to O(n) (the same as overhead in the internal 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 behavior has been shown in [14].

3.2. Simulations

To analyze our protocol, we carry out different simulations. The scenario used in these simulations consists of one internal group and one external group connected by one super peer. All of the peers in the system are divided equally between these two groups. In the simulation, each peer generates a message every 70–90 milliseconds. The system is simulated with a different number of peers and with different delays in the communication channels. All of the simulation scenarios are listed in Table 1. The simulations are performed in the OMNeT++ discrete event simulator [31].

Table 1. Number of peers and delays.

Message Delays	Number of Peers
0–50 milliseconds	10–900
50–250 milliseconds	10–500
50–550 milliseconds	10–400

All of the delays are distributed normally with the mean being the middle of the interval and 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 of the interval, the value of the nearest interval end is taken instead.

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 by our protocol, it is validated against an IDR to detect causal order violations not detected by our protocol. In addition, the overheads for IDR and Dependency Sequences (DS) [15] were calculated to be compared with an overhead generated by our protocol.

As the IDR protocol is designed for plain networks, it is simulated in the following way. The super peer in the system is considered transparent for the IDR implementation, and each peer (internal and external) performs the full calculations of the IDR protocol as described in [14]. A super peer only retransmits the messages between groups.

As for DS protocol simulations, only the external group is considered. Each external peer is simulated as a super peer with exactly one peer in its group.

The results of simulations are analyzed in the following way. Two overheads are mainly analyzed: an overhead of communication (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 the Dependency Sequences [15] stores information only at super peers level, the storage overhead for this protocol is not analyzed.

By considering channel delays from 0 to 50 milliseconds (see Figure 4), the simulation results show that an overhead in an internal group is lower than an overhead in an external group, and an overhead in an external group is lower than an overhead produced by the Immediate Dependency Relation and the Dependency Sequences protocols. For 900 peers, each internal peer requires storing on average 128 bytes of information, and the average overhead in an internal group is around 90 bytes. In the external group, each peer stores on average 2700 bytes and produces an overhead on average of 950 bytes. The IDR protocol requires storing on average 4900 bytes on each peer and to send 1650 bytes, while the DS protocol sends on average 3950 bytes.

For these delays the results show that the overhead in an internal group is 18.3 times lower than the overhead of IDR and require storing 38 times less information. Overhead in an external group is about 1.7 times lower than the overhead of IDR and 4.1 times lower than the one of DS protocol, and require storing 1.7 times less information.

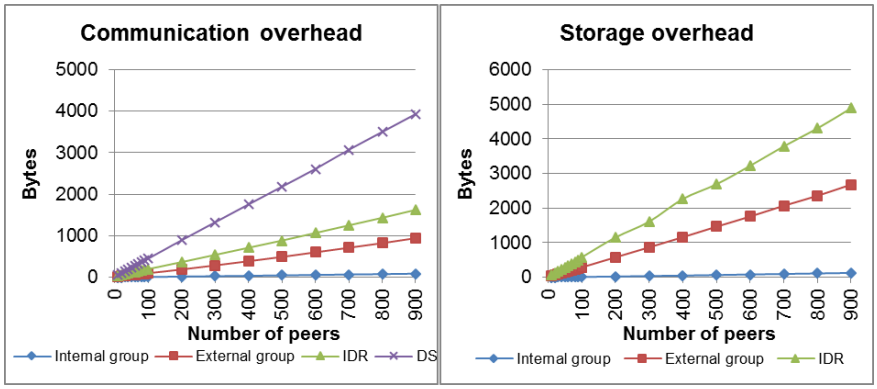
As all of the overheads have a liner type growth we calculate the slope using the least squares method. For communication overheads, the slopes are: 0.098 for internal group, 1.018 for external group, 1.760 for IDR and 4.356 for DS protocol. For storage overheads, the slopes are: 0.143 in internal group, 2.952 in external group and 5.388 for IDR protocol.

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 a 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 that $m_1 \downarrow m_2$ if and only if it has already delivered m_1. If a peer received a message m_3 that $m_1 \downarrow m_3$ and has already delivered m_2 (which implies delivery of m_1), then it can deliver m_3 without any delay. 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.

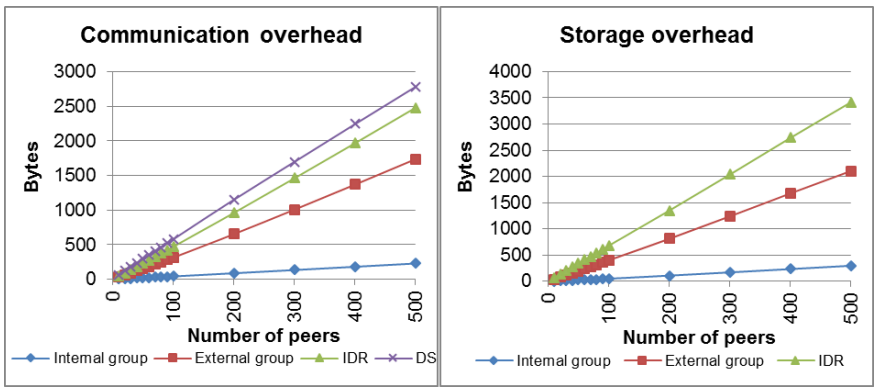
Therefore, if all peers have delivered m_2 such that $m_1 \downarrow m_2$, then the delivery of messages depending on m_1 will not be affected in the internal group in any way, and a super peer will not receive any message depending on m_1 from the internal group. Thus, the information about m_1 can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



(**a**) (**b**)

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

By considering channel delays from 50 to 250 milliseconds (see Figure 5), the results also show that an overhead in an internal group is lower than an overhead in an external group, and the overhead in an external group is lower than an overhead produced by the Immediate Dependency Relation and the Dependency Sequences protocols. For 500 peers, each internal peer requires storing on average 300 bytes of information, and the average overhead in an internal group is around 230 bytes. In the external group, each peer stores on average 2100 bytes and produces an overhead on average of 1700 bytes. The IDR protocol requires storing on average of 3400 bytes on each peer and sending 2400 bytes, while the DS have a communication overhead of 2800 bytes.



(**a**) (**b**)

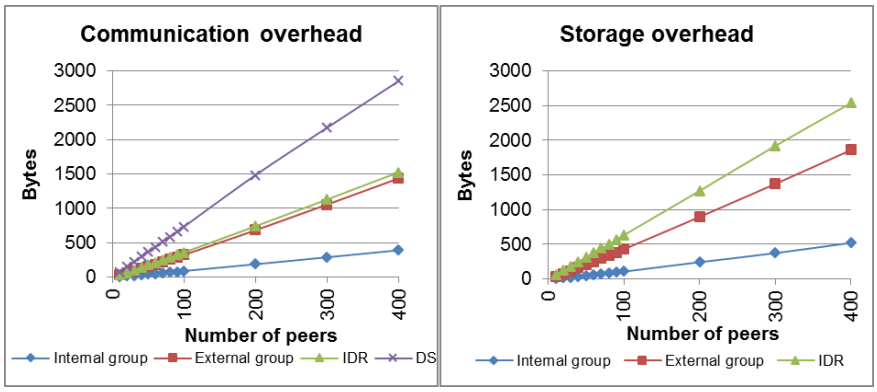
Figure 5. Communication (**a**) and storage (**b**) overheads in bytes for a system with delays from 50 to 250 milliseconds.

The results for delays from 50 to 250 milliseconds show that the overhead in an internal group is ten times lower than the overhead of IDR and requires storing eleven times less information. The overhead in an external group is about 1.4 times lower than the overhead of IDR and 1.64 times lower than the one of the DS, and requires storing 1.6 times less information.

For communication overheads, the slopes are: 0.461 for internal group, 3.480 for external group, 4.965 for IDR and 5.561 for DS protocol. For storage overheads, the slopes are: 0.603 in internal group, 4.225 in external group and 6.838 for IDR protocol.

In scenario with delays from 50 to 550 milliseconds (see Figure 6) and for 400 peers, each internal peer requires storing on average 550 bytes of information and the average overhead in an internal group is around 400 bytes. In the external group, each peer stores on average 1860 bytes and produces an overhead on average

of 1400 bytes. The IDR protocol requires storing on average 2500 bytes on each peer and sending 1500 bytes while the DS requires sending 2850 bytes.



(**a**) (**b**)

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

These results show that the overhead in an internal group is 3.5 times lower than the overhead of the IDR and requires storing 4.5 times less information. The overhead in an external group is similar to the overhead of the IDR, but requires storing 1.3 times less information. As for the DS protocol, the proposed solution has two times lower communication overhead.

For communication overheads, the slopes are: 0.990 for internal group, 3.617 for external group, 3.830 for IDR and 7.162 for DS protocol. For storage overheads, the slopes are: 1.325 in internal group, 4.680 in external group and 6.378 for IDR protocol.

Scalability Analysis

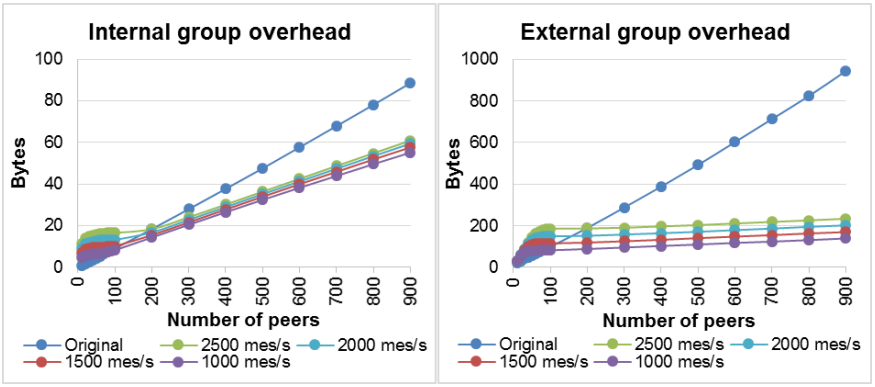
To analyze the scalability of the system, we evaluate the control information management of the protocol against the message density, which is defined as the number of messages sent per process per time unit. For this last analysis, we conducted the following experiment. The system model from the previous experiment was modified in the following way: each peer, instead of generating the message every 80 milliseconds was modified to generate messages so that the total number of messages remain fixed for each number of peers in the system. For example, in a system with a generation rate of 1000 messages per second, with 50 peers, each peer generates 20 messages per second; in a system with 100 peers, each peer generates 10 messages per second; and in a system with 500 peers, each peer generates two messages per second.

The system is analyzed for the following message generation rates: 2500, 2000, 1500 and 1000 messages per second. The obtained results are compared with the results from the experiment described above. In the experiment above, the number of messages generated in the system is 12.5 per second per peer.

By considering channel delays from 0 to 50 milliseconds (see Figure 7), the simulation results show that more messages are generated in the system per second; more control information is required to ensure the causal ordering of messages. In addition, the overhead growth has two different stages: a slow growth for a low number of peers (less than 100), and a linear growth part when the number of peers is high (more that 100–200 in this experiment).

Analyzing the growth of the overhead (linear growth part) in the internal group, the slopes are 0.0612 for 2500 messages/second, 0.0611 for 2000 messages per second, 0.0603 for 1500 messages/second and 0.0583 for 1000 messages/second. In the external group, the slopes are 0.0685 for 2500 messages/second, 0.0728 for 2000 messages/second, 0.0735 for 1500 messages/second and 0.0719 for 1000 messages/second. In the original experiment, the slopes are 0.0979 for the internal group and 1.0177 for the external group.

In this scenario, the difference between slopes with the message limit is less than five percent. However, the difference with the original experiment is 1.60 times for the internal group and 13.9 times for the external group.

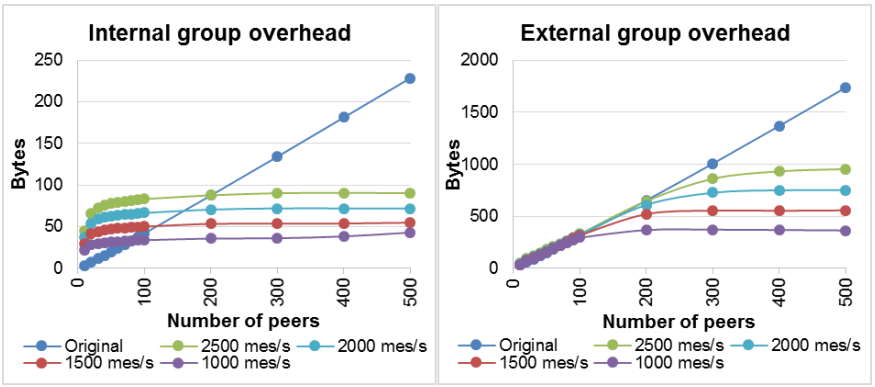


(**a**) (**b**)

Figure 7. Internal (**a**) and external (**b**) group overheads in bytes for a system with delays from 0 to 50 milliseconds.

By considering channel delays from 50 to 250 milliseconds (see Figure 8), the results also show that more messages are generated in the system per second; more overhead is required to ensure the causal ordering. The overhead has the same growth stages as for the system with communication channel delays from 0 to 50 milliseconds.

Analyzing the growth of the overhead (linear growth part) in the internal group, the slopes are 0.0083 for 2500 messages/second, 0.0041 for 2000 messages per second, 0.0045 for 1500 messages/second and 0.0241 for 1000 messages/second. In the external group, the slopes are 0.4632 for 2500 messages/second, 0.1215 for 2000 messages/second, 0.0147 for 1500 messages/second and −0.0374 for 1000 messages/second. In the original experiment, the slopes are 0.4606 for the internal group and 3.4804 for the external group.



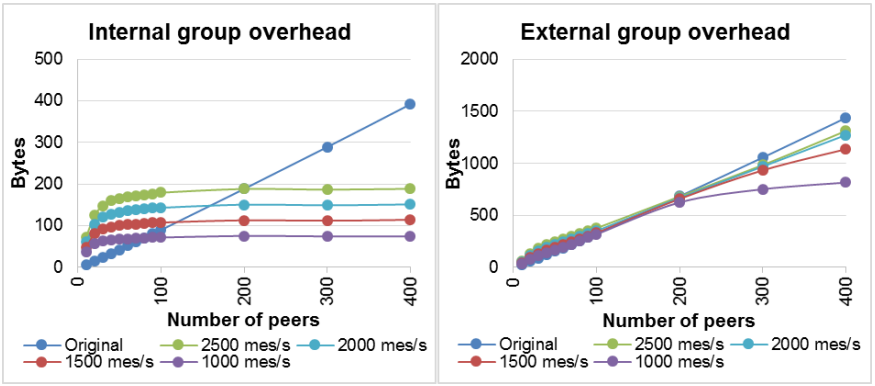
(**a**) (**b**)

Figure 8. Internal (**a**) and external (**b**) group overheads in bytes for a system with delays from 50 to 250 milliseconds.

In this scenario in both groups the slopes have a tendency toward zero, which means that the overhead directly depends on the density of the messages exchanged, and does not depend on the number of peers in the system. The slopes from this experiment are 45 times lower for the internal group and 25 times lower for the external group.

In a scenario with delays from 50 to 550 milliseconds (see Figure 9), the results show that more messages are generated in the system per second, more overhead is required to ensure the causal ordering. The overhead has the same growth stages as for the system with communication channel delays from 0 to 50 milliseconds and from 50 to 250 milliseconds.

Analyzing the growth of the overhead in the internal group, the slopes are 0.002177379 for 2500 messages per second, 0.00272099 for 2000 messages per second, 0.005812997 for 1500 messages per second and −0.006493243 for 1000 messages per second while in the original experiment the slope is 0.990. As for external group the slopes could not be calculated because the linear growth part was not reached but the same tendency as for other network delays can be observed.



(**a**) (**b**)

Figure 9. Internal (**a**) and external (**b**) group overheads in bytes for a system with delays from 50 to 550 milliseconds.

In this scenario, the slopes have a tendency toward zero which means that the overhead does not directly depends on the number of peers in the system, but it depends on the message density in the system.

The results obtained from this experiment show that when the number of messages generated in the system per second remains constant, the overhead growth slope has a tendency toward zero. This means that there is no direct dependency between the number of peers in the system and the amount of control information required for causal message ordering. Thus, the number of peers in the system can be relatively high while the message generation rate of the system remains constant.

In addition, it should be noted that the above experiments consider that all of the peers in the system equally participate in the communication. All of the peers that are passive receivers do not increase system overhead. For example, in a system with 300 peers where 200 peers are passive receivers and do not generate any messages, the overhead will be the same as in a system containing only 100 participating peers. We note that in real world P2P applications, such as video call/meeting, current solutions only support up to 25 peers participating equally in the communication. This means that with our protocol, applications could allow up to 100 peers exchange streams in an equitable participation, keeping a controlled overhead growth.

This leads to the following. If several communication groups exist over the same peer-to-peer network, the communication and storage overhead will only depend on the number of peers participating in a group communication, not on the total number of peers in the system. For example, the overhead will depend on the number of people participating in a video conference, and not on a total number of people using the software.

4. Discussion

Some protocols have been proposed to implement causal message ordering for peer-to-peer overlay networks. These protocols can be divided into three groups based on the network architecture they are designed for. In addition, each group can be further divided into two sub-groups, based on whether a protocol uses a global time reference or if it is based on logical references between messages. These protocols are described as follows.4.1. Protocols Designed for Plain Networks

A plain peer-to-peer overlay network consists of a number of interconnected peers. Each peer communicates directly with other peers. In a broadcast group, each peer sends messages to every other peer in a group.

4.1.1. Protocols that Use A Global Time Reference

All protocols from this category [9,19–21] are based on the same idea 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 time with a time server and have knowledge about the minimum and maximum delays of messages in communication channels and maximum clock drift. To determine the order of two messages m_1 and m_2, these protocols use the physical time. If the difference in physical time when the messages were sent is greater than the maximum delay, then the messages can be ordered based on physical time. In addition, they can determine that a message m_1 cannot be received when a message m_2 was sent (difference in physical time is smaller than

the minimum delay). If these protocols cannot determine a message order based on these rules, a logical clock is used to order messages.

4.1.2. Protocols that Use Logical References

This section contains the protocols that are designed for plain peer-to-peer networks and do not use the global time reference. To ensure causal ordering, these protocols use the "happened before" relation defined by Lamport [30].

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 with a message, but in the worst case scenario, it produces n-1 copies (where n is the number of peers in the system) of the same message saturating 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 and not completely removing them. To achieve this reduction, this protocol resends the last received message with the new message, thus doubling the average message size. However, the protocol does not require any control information to be sent alongside the message to ensure message causal ordering.

An Efficient Algorithm for Causal Message Ordering [12] extends the idea of vector clock [32] for the cases where each message is addressed to a different subset of processes. This modification requires more information to be stored than in the Vector clock protocol. In the worst case scenario, this protocol requires storing an entry for each process in a group, thus producing an overhead of O(n) (where n is the number of peers in the system) and when sending a message, all of these entries must be included.

Probabilistic Analysis of Causal Message Ordering [22] analyzes the probability of violation of causal order of messages. This protocol implies that if the message send is delayed after the last send or receive event, the probability decreases and can decrease down to 0. 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. This produces the reduced average overhead compared 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 processes in the system (O(n), where n is the number of peers in the system). In regard to storage requirements, this protocol stores vector clock and message control information, thus producing storage requirements twice as high as the number of peers in the system.

An optimal algorithm for enforcing the causal messages ordering is proposed in [24]. The algorithm achieves the optimality by using four propagation constraints to reduce the propagation of redundant information and by using an encoding scheme to represent and attach in messages only the necessary causal dependency information. A performance analysis of the communication overhead of the optimal algorithm under a wide range of system conditions using simulations is presented in [25]. The results indicate that the optimal algorithm performs significantly better than the algorithm proposed in [33]. However, the communication overhead in the worst case is O(n), where n is the number of peers in the system (broadcast case).

4.2. Protocols Designed for Free Scale Networks

A free scale network consists of a number of peers and super peers (a super peer is a peer with major processing power and bandwidth). The peers are divided into two groups based on their connection types. Some of the peers are only connected to a super peer. Other peers are interconnected between them and super peers [28]. While we surveyed the state of the art, no protocols were found specifically designed for this type of networks. However, 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 network topology and treat peers in different groups in the same manner. Thus, the overhead grows with the number of peers in the system.

4.3. Protocols Designed for Hierarchical Networks

A hierarchical network is a network consisting of peers and super peers. A peer is connected only to a super peer while super peers communicate between them [28]. In this network architecture, if a peer sends a

message to a peer in another group, it sends a message to its super peers. A super peer forwards this message to a super peer connected to a destination peer, and then the message is forwarded to its destination.

4.3.1. Protocols that Use a Global Time Reference

Protocols in this subgroup [13,17,18] are based on the same idea that is used for protocols with a global clock designed for a plain network. They use a combination of logical and physical clocks to ensure causal message order.

4.3.2. Protocols that Use Logical References

This section contains the protocols that are designed for hierarchical peer-to-peer networks that use the "happed before" relation defined by Lamport [30] or other mechanisms that do not imply global time reference.

Distributed Floor Control Protocols [7] and A Reliable Multicast [6] protocol use the synchronization mechanisms by coordinators to ensure causal ordering of events. As a result, these protocols do not require any control information to be sent, but require an infrastructure to support communication between peers. In addition, these protocols are a type of server–client protocol, where a server order is enforced to all clients.

The Two-Layered protocol for a Large-Scale Group of Processes [16] and the Domain-Based Causal Ordering Group Communication protocol [11] use two vector clocks: one for the subgroup, and another one for the global group. As this reduces the size of control information and storage requirements, it implies that the messages that are concurrent in one group become causally ordered in another group. Both protocols require sending g integers in a subgroup, where g is the number of peers in a subgroup and l integer in a global group, where l is the number of groups.

An Optimal Causal Broadcast Protocol [8] sends only the identifier of a last received or send message. As a result of this decision, the overhead is constant, but this can produce cases where the messages will not be correctly ordered.

Dependency Sequences [15] represent message causal relations 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 will have an overhead that is no less than the overhead of a previous message. Without an external mechanism to remove unnecessary dependencies, an overhead will grow indefinitely.

Hierarchical Clocks [15] use two different clocks to represent a message causal relationship. One clock is used to represent the message dependency on external events (events from other groups) and the other clock to represent message dependency on local events (events from the same group). A disadvantage of these protocols is that they do not contain mechanisms to reduce sequence sizes, and thus the size of the control information will grow indefinitely. They require external mechanisms like checkpoints to accomplish this task.

In the Adaptive Causal Ordering Algorithm [26], a message carries control information only about its direct predecessors; however, in the worst case, the communication overhead is O(n) integers per message, where n is the number of peers in the system (broadcast case).

For the Causal Multicast in Mobile Networks [27], the communication overhead is dynamic having a worst case of size O(n), where n is the number of peers in the system (broadcast case). We note that in this protocol, the updating process of the control structures considers the acknowledgement by the peers for each message received to its local super peer, which increases the delay in the communication.

5. Conclusions

The protocol presented in this paper ensures causal message ordering in a free scale peer-to-peer overlay network. The protocol uses a combination of the Immediate Dependency Relation protocol and an idea of hierarchical clocks to reduce message overhead. Our protocol is efficient in the overhead attached per message at the communication channels. The overhead sent per message is characterized by being dynamically adapted according to the behavior of the concurrent messages. In addition, the simulations show that the protocol proposed in this work produces less overhead than the IDR and DS protocols, and also requires less storage overhead. As a result, this protocol can be used to ensure causal message ordering, while producing a lower overhead that the IDR and DS protocols. The lower overhead will allow for devices with lower computational capacities to be included in the system. The protocol proposed in this paper can be further extended by reducing an overhead dependency on the number of internal groups, allowing for more super peers and corresponding

internal groups to be included in a system. This will allow a further extension of the protocol to be used in hierarchical peer-to-peer networks.

Acknowledgments: This research was partially supported by the National Council on Science and Technology (CONACyT, México), Project # C394/2016/271602 and grant number 290817 assigned to Grigory Evropeytsev.

References

1. Tarkoma, S. *Overlay Networks: Toward Information Networking*; Auerbach Publications: Boston, MA, USA, 2010.
2. Cohen, B. The BitTorrent Protocol Specification, Version 11031 (10 January 2008). Available online: http://www.bittorrent.org/beps/bep_0003.html (accessed on 7 July 2015).
3. Knutsson, B.; Lu, H.; Xu, W.; Hopkins, B. Peer-to-peer Support for Massively Multiplayer Games. In Proceedings of the Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies (INFOCOM 2004), Hong Kong, China, 7–11 March 2004.
4. Baset, S.A.; Schulzrinne, H.G. An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. In Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM 2006), Barcelona, Spain, 23–29 April 2006; pp. 1–11.
5. Schwarz, R.; Mattern, F. Detecting Causal Relationships in Distributed Computations: In Search of the Holy Grail. *Distrib. Comput.* **1994**, *7*, 149–174.
6. Anastasi, G.; Bartoli, A. A Reliable Multicast Protocol for Distributed Mobile Systems: Designs and Evaluation. *IEEE Trans. Parallel Distrib. Syst.* **2001**, *12*, 1009–1022.
7. Banik, S.M.; Radhakrishnan, S.; Zheng, T.; Sekharan, C.N. Distributed Floor Control Protocols for Computer Collaborative Applications on Overlay Networks. In Proceedings of the International Conference on Collaborative Computing: Networking, Applications and Worksharing, San Jose, CA, USA, 19–22 December 2005.
8. Benzaid, C.; Badache, N. An Optimal Causal Broadcast Protocol in Mobile Dynamic Groups. In Proceedings of the International Symposium on Parallel and Distributed Processing with Applications, Sydney, Australia, 10–12 December 2008; pp. 477–484.
9. Doulikun, D.; Aikebaier, A.; Enokido, T.; Takizawa, M. Experimentation of Group Communication Protocols. In Proceedings of the 16th International Conference on Network-Based Information Systems (NBiS), Gwangju, Korea, 4–6 September 2013; pp. 476–481.
10. Friedman, R.; Manor, S. *Causal Ordering in Deterministic Overlay Networks*; Israel Institute of Technology: Haifa, Israel, 2004.
11. Hsiao, C.; Liao, Y. Domain-Based Causal Ordering Group Communication in Wireless Hybrid Networks. In Proceedings of the 5th International Conference on Ubiquitous Information Management and Communication, Seoul, Korea, 21–23 February 2011; pp. 131:1–131:6.
12. Maddi, A.; Dahamni, F. An Efficient Algorithm for Causal Messages Ordering. In Proceedings of the 2001 ACM Symposium on Applied Computing, Las Vegas, NV, USA, 2001; pp. 499–503.
13. Nishimura, T.; Hayashibara, N.; Enokido, T.; Takizawa, M. Causally Ordered Delivery with Global Clock in Hierarchical Group. In Proceedings of 11th International Conference on Parallel and Distributed Systems, Fukuoka, Japan, 20–22 July 2005; pp. 56–564.
14. Pomares Hernández, S.E. The Minimal Dependency Relation for Causal Event Ordering in Distributed Computing, *Appl. Math. Inf. Sci.* **2015**, *9*, 57–61.
15. Prakash, R.; Singhal, M. Dependency Sequences and Hierarchical Clocks: Efficient Alternatives to Vector Clocks for Mobile Computing Systems. *Wirel. Netw.* **1997**, *3*, 349–360.
16. Taguchi, K.; Takizawa, M. Two-Layered Protocol for a Large-Scale Group of Processes. In Proceedings of the Ninth International Conference on Parallel and Distributed Systems, Zhongli, Taiwan, 17–20 December 2002; pp. 171–176.
17. Tsuneizumi, I.; Aikebaier, A.; Enokido, T.; Takizawa, M. A Flexible Group Communication Protocol with Hybrid Clocks. In Proceedings of the 7th International Conference on Advances in Mobile Computing and Multimedia, Kuala Lumpur, Malaysia, 14–16 December 2009; pp. 469–474.
18. Tsuneizumi, I.; Aikebaier, A.; Ikeda, M.; Enokido, T.; Takizawa, M. A Scalable Hybrid Time Protocol for a Heterogeneous Group. In Proceedings of the International Conference on Broadband, Wireless Computing, Communication and Applications, Fukuoka, Japan, 4–6 November 2010; pp. 214–221.
19. Tsuneizumi, I.; Aikebaier, A.; Enokido, T.; Takizawa, M. A Scalable Peer-to-Peer Group Communication Protocol. In Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications, Perth, WA, USA, 20–23 April 2010; pp. 268–275.

20. Tsuneizumi, I.; Aikebaier, A.; Ikeda, M.; Enokido, T.; Takizawa, M.; Deen, S.M. Hybrid Clock-Based Synchronization in a Scalable Heterogeneous Group. In Proceedings of the 13th International Conference on Network-Based Information Systems, Takayama, Japan, 14–16 September 2010; pp. 246–253.
21. Tsuneizumi, I.; Aikebaier, A.; Enokido, T.; Takizawa, M. Reduction of Messages Unnecessarily Ordered in Scalable Group Communication. In Proceedings of the International Conference on Complex, Intelligent and Software Intensive Systems, Krakow, Poland, 15–18 February 2010; pp. 299–306.
22. Yen, L. Probabilistic Analysis of Causal Message Ordering. In Proceedings of the Seventh International Conference on Real-Time Computing Systems and Applications, Cheju Island, Korea, 12–14 December 2000; pp. 409–413.
23. Zhou, S.; Cai, W.; Turner, S.J.; Lee, B.; Wei, J. Critical Causal Order of Events in Distributed Virtual Environments. *ACM Trans. Multimed. Comput. Commun. Appl.* **2007**, *3*, doi:10.1145/1236471.1236474.
24. Kshemkalyani, A.D.; Singhal, M. Necessary and Sufficient Conditions on Information for Causal Message Ordering and Their Optimal Implementation. *Distrib. Comput.* **1998**, *11*, 91–111.
25. Chandra, P.; Gambhire, P.; Kshemkalyani, A.D. Performance of the Optimal Causal Multicast Algorithm: A Statistical Analysis. *IEEE Trans. Parallel Distrib. Syst.* **2004**, *15*, 40–52.
26. Prakash, R.; Raynal, M.; Singhal, M. An Adaptive Causal Ordering Algorithm Suited to Mobile Computing Environments. *J. Parallel Distrib. Comput.* **1997**, *41*, 190–204.
27. Chandra, P.; Kshemkalyani, A.D. Causal Multicast in Mobile Networks. In Proceedings of the IEEE Computer Society's 12th Annual International Symposium on Date of Conference:Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, Volendam, The Netherlands, 4–8 October 2004; pp. 213–220.
28. Yang, B.; Garcia-molina, H. Designing a Super-peer Network. In Proceedings of the IEEE International Conference on Data Engineering, Bangalore, India, 5–8 March 2003; pp. 49–60.
29. Virtual Private Networking: An Overview. Available online: https://technet.microsoft.com/en-us/library/bb742566.aspx (accessed on 1 December 2015).
30. Lamport, L. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM* **1978**, *21*, 558–565.
31. OMNeT++ Discrete Event Simulator. Available online: https://omnetpp.org/ (accessed on 23 July 2015).
32. Mattern, F. Virtual Time and Global States of Distributed Systems. In *Proceedings of Parallel and Distributed Algorithms*; North-Holland Publishing: Amsterdam, The Netherlands, 1988; pp. 215–226.
33. Raynal, M.; Schiper, A.; Toueg, S. The causal ordering abstraction and a simple way to implement it. *Inf. Process. Lett.* **1991**, *39*, 343–350.