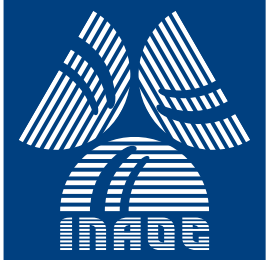


Mecanismo escalable para la sincronización de datos multimedia en tiempo real para la comunicación en grupo

por

José Ricardo Pérez Cruz

Tesis sometida como requisito parcial para obtener el grado de

MAESTRO EN CIENCIAS EN EL ÁREA DE CIENCIAS COMPUTACIONALES

Instituto Nacional de Astrofísica, Óptica y Electrónica
Febrero 2014
Tonantzintla, Puebla

Supervisada por:

Dr. Gustavo Rodríguez Gómez, INAOE
Dr. Saúl Eduardo Pomares Hernández, INAOE
Dr. Eduardo López Domínguez, LANIA

Resumen

La necesidad de interacción entre varios usuarios ha motivado el desarrollo de nuevas aplicaciones en Sistemas Multimedia (SM), mediante estas nuevas aplicaciones se espera maximizar la experiencia del usuario y cumplir con la demanda de comunicación. Estos nuevos retos motivan el diseño y desarrollo de mecanismos para la sincronización de datos multimedia en Sistemas de Comunicación en Grupo (SCG). El principal problema al que se afrontan estos nuevos mecanismos es la escalabilidad, ya que deben de manejar de manera eficiente el incremento tanto en el número de flujos como de participantes en la comunicación.

La principal contribución de esta investigación es la propuesta de un mecanismo escalable para la sincronización de flujos multimedia basado en un esquema descentralizado. El esquema propone la división del grupo de comunicación en subgrupos de participantes y una jerarquía en función de su distancia temporal. Mediante el uso de un esquema maestro/esclavo se logra jerarquizar los flujos; además se logra reducir el número de información de control a enviar y brindar el soporte al incremento de las entidades a partir de la división de subgrupos.

Abstract

The need of interaction between multiple users has motivated the development of new applications in multimedia systems, through these new applications are expected maximize the user experience and meet the communication demands. These new challenges motivates the design and development of mechanisms for multimedia data synchronization in Group Communication Systems. The main issue facing these new mechanisms is scalability, as they must efficiently handle the increase in both the number of flows and participants in communication.

The main contribution of this work is the proposal of a scalable mechanism for the multimedia streams synchronization based on decentralized scheme. These scheme propose the split of group of communication in subgroups of participants and a hierarchy according to their temporal distance. By the use of a Master/Slave scheme the hierarchy of flows is achieved; also will achieves reduce the number of control information to send and provide support for the increase of entities from the division in subgroups.

Índice general

Índice de figuras

Índice de tablas

Lista de acrónimos

3DTI 3D Tele-inmersión

GSM-LM Group Synchronization Mechanism for Live Media

HA-MMD Hierarchical Architecture for Multimedia Distribution

ITU International Telecommunication Union

LAN Local Area Network

Lip-sync Lip synchronization

M/S Master/Slave

MMS Multimedia streams

ML Mapeo Lógico

MLJ Mapeo Lógico Jerárquico

MU Media Unit

NTP Network Time Protocol

PAN Personal Area Network

QoS Quality of Service

RFGSA RTP-based Feedback Global Synchronisation Approach

RTP Real-time Transport Protocol

SCG Sistemas de Comunicación en Grupo

SM Sistemas Multimedia

SMCG Sincronización multimedia en la comunicación en grupo

TS-GCS Topological Structure for the Group Communication System

UTC Universal Time Coordinated

WAN Wide Area Network

Capítulo 1

Introducción

1.1. Motivación

La evolución del Internet ha motivado el desarrollo de nuevas aplicaciones, en Sistemas Multimedia (SM), que maximicen la experiencia del usuario. Algunos ejemplos de estas nuevas aplicaciones son la 3D Tele-inmersión (3DTI) [HWN+10, AHNA12], tele-orquesta (tele-orchestra) [MIF+11], juegos en tiempo real [YK13] y videoconferencias multipartitas (entre 2 o más personas) [LZKM10]. Mediante estas aplicaciones se pretende que un grupo de usuarios, conectados desde cualquier parte del mundo, puedan interactuar en tiempo real de una manera más cercana a lo natural mediante un entorno virtual.

Sistemas como 3DTI deben de enfrentarse a problemas de sincronización, al retraso y a la pérdida de los datos enviados de una o más fuentes presentes en el sistema, ya que al ser desarrollados sobre Internet están comunicándose mediante un canal asíncrono y no fiable.

1.2. Planteamiento del problema

Tras el desarrollo de nuevas aplicaciones en SM que maximicen la experiencia del usuario, como 3DTI, surgen nuevos problemas en la sincronización de flujos multimedia (Multimedia streams (MMS)). Los principales problemas en la sin-

cronización de MMS sobre Internet son: la comunicación en grupo, el manejo de datos multimedia y el manejo de la escalabilidad.

El paradigma de comunicación en grupo ha generado la necesidad de desarrollar mecanismos que brinden el soporte a una comunicación de *muchos-a-muchos*. El principal problema al que se enfrentan estos mecanismos es la arquitectura de Internet, ya que los protocolos que dan soporte a éste están orientados a una comunicación *uno-a-uno*.

Otro de los retos que presenta el diseño de Internet es el manejo de datos multimedia, los cuales son utilizados por los SM, ya que el Internet no fue diseñado para el manejo de este tipo de datos. La transmisión y sincronización de datos multimedia presenta varios retos, ya que deben de cumplirse restricciones de calidad de servicio (Quality of Service (QoS)) para el manejo de tiempo real, dándole importancia al desarrollo de mecanismos para el control y sincronización de datos multimedia.

Existen mecanismos [BLG09, BGL08, IT97, IT95, PMER08] que atacan los problemas de sincronización de datos en tiempo real, la mayoría de ellos hacen uso de referencias globales (tal es el caso de RTP [Per03]) y algunos otros mediante referencias lógicas. La mayoría de estos mecanismos se centran sólo en la sincronización de MMS dejando aún como un problema abierto el manejo de la escalabilidad.

Un mecanismo se dice que es escalable si éste continua funcionando correctamente cuando los objetos que lo componen (procesos, mensajes, etc.) son incrementados [Bon00]. Lo anterior es de suma importancia para los SM en Sistemas de Comunicación en Grupo (SCG), ya que en los SCG pueden ser generados distintos números de flujos provenientes de distintos participantes y se deben de preservar

las referencias temporales entre los flujos para su correcta reproducción en los receptores.

1.3. Propuesta de solución

Tomando en cuenta los requerimientos de los recientes SM en SCG, el presente trabajo de investigación se centra en el desarrollo de un mecanismo capaz de brindar el soporte a la escalabilidad de las entidades del sistema, la heterogeneidad y la sincronización de MMS. Para lograrlo, nuestra investigación parte de la siguiente hipótesis.

En redes de área amplia (Wide Area Network (WAN)), es posible diseñar mecanismos escalables para la sincronización de datos continuos y discretos en tiempo real en ambientes de comunicación multimedia en grupo, con base en el agrupamiento dinámico de las entidades del sistema (procesos y flujos) y la identificación de dependencias causales entre los datos.

Para demostrar nuestra hipótesis en esta tesis se propone, como contribución principal, el desarrollo de un mecanismo escalable para la sincronización de flujos multimedia basado en una arquitectura descentralizada. Esta arquitectura se caracteriza por dividir y jerarquizar a los participantes en subgrupos, explotando la distancia temporal entre participantes y reduciendo el trabajo para la sincronización multimedia.

En este trabajo también se propone el uso de un esquema de sincronización maestro/esclavo, por medio del cual se jerarquizan los flujos y consecuentemente se reducen las dependencias temporales entre ellos. Haciendo uso tanto del esquema de sincronización como de la arquitectura descentralizada, el mecanismo propues-

to considera reducir el *overhead* en la comunicación y la tolerancia al incremento de las entidades del sistema.

1.4. Metodología

Para cumplir con los objetivos planteados para el trabajo de tesis, se planteó la siguiente metodología:

1. **Diseñar un modelo jerárquico organizacional.** Se refiere a diseñar un modelo para el agrupamiento de las entidades, el trabajo se distribuye en subgrupos y de esta manera se reduce la información de control a enviar. Dado que las entidades pueden estar distribuidas en cualquier parte del mundo, se deberá hacer uso de políticas para el agrupamiento y selección de las entidades de manera lógica.

2. **Diseñar un esquema jerárquico para la sincronización multimedia.** El diseño del esquema de sincronización se basa en el esquema organizacional previamente diseñado. Dado el agrupamiento de las entidades, el esquema de sincronización especificará como los participantes sincronizarán los flujos multimedia transmitidos dentro de cada subgrupo. El esquema de sincronización aprovechará la jerarquización de las entidades para reducir el *overhead* en la comunicación.

3. **Desarrollar un mecanismo escalable para la sincronización de datos multimedia.** Se refiere al diseño y desarrollo de algoritmos que hagan uso del esquema organizacional y del esquema de sincronización. El principal problema de los métodos existentes es que no soportan el incremento tanto de participantes como de MMS, por lo que aquí se presenta una primera propuesta para resolver ambos problemas.

4. **Simulación del mecanismo desarrollado.** Una vez que se haya desarrollado el mecanismo jerárquico, éste se evaluará por medio de simulaciones. Mediante las simulaciones se comprobará que el mecanismo cumple con la restricciones de tiempo real además de lograr la reducción del *overhead* en la comunicación.

1.5. Organización de la tesis

El documento está organizado de la siguiente manera: en el capítulo dos se presentan los conceptos para explicar los tipos de comunicación en grupo y el manejo de referencias temporales para la sincronización. En el capítulo tres se muestra un resumen de los trabajos existentes. Para la organizacíon de este capítulo se propone una taxonomía para la organización del trabajo relacionado. En el capítulo cuatro se explica el mecanismo de sincronización desarrollado. Como primer punto se detallan algunas definiciones y posteriormente se presentan los esquemas para la organización y sincronización desarrollados. Por último, se presenta el algoritmo desarrollado así como la explicación en lo general. En el capítulo cinco se explica como se realizaron las simulaciones y se da un análisis de los resultados obtenidos. En el capítulo seis se da un breve resumen del trabajo realizado, se presentan las conclusiones y se explica el trabajo futuro. En el anexo A, se presenta una ejecución realizada del mecanismo desarrollado mediante de un escenario de prueba para mostrar su funcionamiento.

Capítulo 2

Fundamentos

En este capítulo se fundamentan las características principales de la sincronización multimedia y el uso de referencias lógicas. Como primer punto, se explica como se realiza la sincronización de datos multimedia, seguido del tipo de esquemas usados para realizar la sincronización. Por último, se explica la importancia del uso de referencias lógicas para la sincronización de datos multimedia.

2.1. Sincronización de datos multimedia

Para realizar la sincronización de los flujos multimedia generados contamos con dos clases generales de sincronización: la sincronización vista desde la fuente y la sincronización vista desde los datos.

Dentro de la sincronización vista desde la fuente hay dos tipos de sincronización: la sincronización intra-proceso y la sincronización inter-proceso. La sincronización intra-proceso consiste en preservar las referencias temporales entre los MMS que están siendo generados a partir de un proceso. La sincronización inter-proceso consiste en preservar las referencias temporales entre los MMS provenientes de diferentes procesos [SN04].

Dentro de la sincronización vista desde los datos existen dos tipos de sincronización de flujos multimedia [BLG09, UB12, Hwa09, BS96]: intra-flujo e inter-flujo.

La sincronización *intra-flujo* consiste en preservar las dependencias temporales físicas que existen entre los mensajes de un mismo flujo continuo. Lo anterior se refiere a mantener la secuencia de imágenes o *frames*, que componen a un flujo de video, para ser reproducidos de manera coherente en los receptores.

La sincronización *inter-flujo* consiste en preservar las dependencias temporales físicas o lógicas entre varios flujos, para poder reproducirlos tal como fueron generados. Por ejemplo, una teleconferencia para educación a distancia donde se están transmitiendo tanto flujos de audio y video así como una serie de filminas (datos discretos). Todos estos flujos deben reproducirse dentro del receptor en el momento preciso para no perder la coherencia de la ponencia.

Uno de los problemas en la sincronización de MMS en SCG, consiste en preservar las referencias existentes entre flujos, necesario para reproducirlos coherentemente en el receptor. En los ambientes distribuidos, al no existir referencias globales, comúnmente se hace uso de dos técnicas para sincronizar los flujos: el uso de relojes físicos o el uso de relojes lógicos.

La sincronización mediante relojes físicos se realiza etiquetando a cada mensaje con una marca de tiempo, mediante la cual se indica en que instante de tiempo debe ser reproducido. El problema de usar una marca de tiempo físico radica en que cada participante en la comunicación cuenta con un tiempo distinto. Comúnmente, para sincronizar a los participantes con un mismo tiempo físico, se hace uso del protocolo NTP[Mil91, BLG09].

NTP es usado para sincronizar los relojes físicos de los participantes en la comunicación, por lo general se hace uso del tiempo UTC como una referencia global. El problema en este tipo de enfoque radica en tener que sincronizar relojes con

un mismo tiempo, ya que genera más información de control a enviar y se tiene que realizar periódicamente.

Para evitar el problema de sincronización de relojes físicos Lamport [Lam78] propone el uso de relojes lógicos usados para indicar la precedencia de un mensaje, sin importar cuando éstos fueron generados. En el siguiente apartado se explicará en detalle el uso de referencias lógicas.

2.1.1. Sincronización multimedia en la comunicación en grupo

En la sincronización de MMS es muy importante el uso de referencias temporales, pero también es importante cómo estas son obtenidas. En los SCG comúnmente se hace uso de dos enfoques, el primero hace uso de esquemas centralizados y el segundo, hace uso de esquemas distribuidos [MBSv12]. Los esquemas centralizados se enfocan en usar a uno de los participantes del sistema como un coordinador global o agente sincronizador. Este coordinador tiene la función de sincronizar los MMS y corregir los errores que reporten cada uno de los participantes. Los esquemas distribuidos se enfocan a que todos los participantes en la comunicación intercambien referencias entre ellos. Esto quiere decir que todos sincronizarán sus flujos con la información proveniente de los flujos que estén trasmitiendo los demás.

El problema principal de los enfoques centralizados radica en que todo el trabajo se centra en un solo participante, pudiéndose generar cuellos de botella. Los enfoques distribuidos presentan el problema de la generación excesiva de información de control, dado que cada participante envía información duplicada. Los problemas anteriores evitan la escalabilidad del sistema, dado que el uso de estos enfoques no brinda el soporte al incremento de información ni al incremento de los participantes.

Tras el problema que presentan los enfoques distribuidos se adopta el modelo Maestro/Esclavo para los sistemas distribuidos; este modelo es principalmente usado para paralelizar el procesamiento en sistemas de control. El modelo Maestro/Esclavo consiste en que uno de los participantes es seleccionado como proceso maestro el cual se encarga de dividir las tareas en el sistema en subtareas, posteriormente el maestro distribuye las subtareas entre los procesos que fungen como esclavos [Cri09, Var09]. Al aplicar este modelo en la sincronización de MMS presenta problemas al no soportar el incremento de participantes.

Un enfoque de los sistemas distribuidos poco explotado en el área multimedia es el uso de esquemas descentralizados. Los esquemas descentralizados se enfocan en dividir y jerarquizar a los participantes mediante el uso de una arquitectura de subgrupos.

Baldoni et al. en [BBFV99] proponen una arquitectura jerárquica causal la cual tiene una estructura lógica en forma de *margarita (daisy)*. Los autores proponen que la arquitectura se vaya formando cada vez que la aplicación detecte el incremento de mensajes entre procesos, dejándole la responsabilidad de unir y separar grupos en cualquier momento.

La formación de los grupos se realiza de la siguiente manera: primero los procesos en el sistema forman un solo grupo que puede estar compuesto por un solo proceso o unos cuantos procesos. Cuando la cantidad de información o de procesos se incrementa se procede a separar los procesos en *grupos locales*. Uno de los procesos dentro del grupo local será seleccionado como *servidor causal*, el cual se encargará de comunicar a los procesos de su grupo local con los demás grupos locales. Al mismo tiempo, los servidores causales formarán parte de un grupo denominado *grupo de servidores causales*. Los grupos se siguen formando de esta

manera hasta obtener la arquitectura de la figura 2.1 a).

Cuando la cantidad de mensajes dentro de una *margarita* se incrementa de mane-

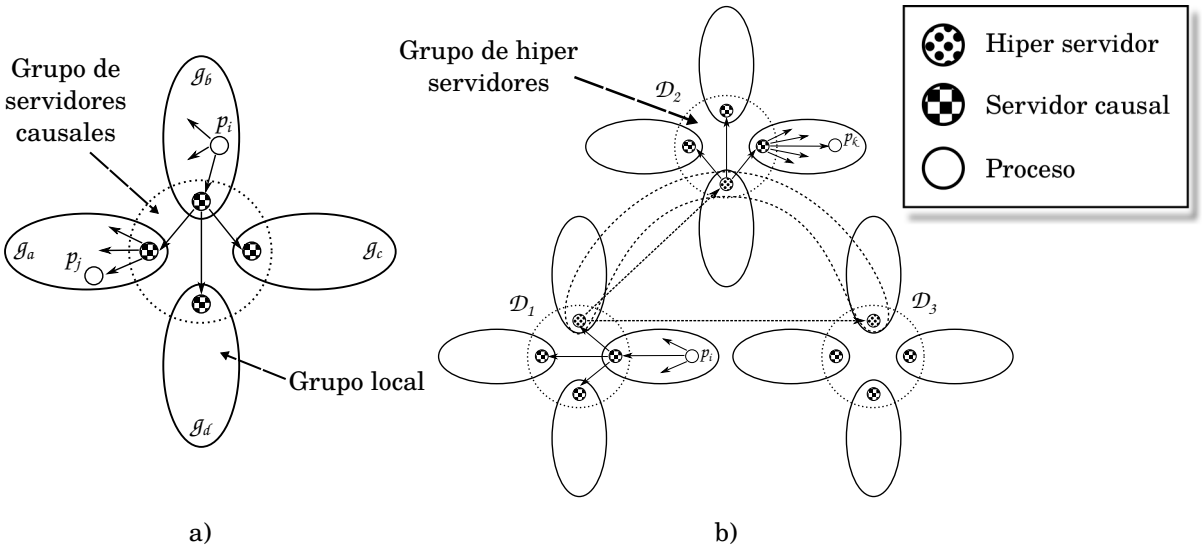


Figura 2.1: Arquitectura jerárquica propuesta en [BBFV99].

ra significativa, se deben separar y crear nuevas *margaritas*. Para comunicar una *margarita* con otra, uno de los servidores causales dentro del grupo de servidores será elegido como *hiper servidor*. Cada *hiper servidor* seleccionado dentro de cada *margarita* formarán un nuevo grupo denominado *grupo de hiper servidores*, los cuales se encargarán de transmitir los mensajes de una *margarita* a otra (ver figura 2.1 b).

2.2. Referencias lógicas

El problema del uso de relojes físicos en un ambiente distribuido se origina en que no todos los participantes pueden tener exactamente el mismo tiempo. Aunque existen técnicas para la sincronización de este tipo de relojes, resulta muy costoso emplearlas en ambientes donde los usuarios comparten constantemente información.

Debido al problema que implica la sincronización de relojes físicos, Lamport [Lam78] propuso el uso de referencias lógicas para la sincronización de eventos en los sistemas distribuidos. La sincronización de eventos es realizada por una relación de orden parcial llamada *Happened-Before* la cual nos dice que un evento a precede a un evento b siempre y cuando a haya sido generado antes que b.

La relación *Happened-Before* se denota por "$\rightarrow$" y está definida por las siguientes reglas [Lam78].

Definición 2.1 *La relación "$\rightarrow$", en un conjunto de eventos de un sistema, es la relación más pequeña que satisface las siguientes condiciones:*

1) *Si "a" y "b" son eventos del mismo proceso y "a" es generado antes que "b", entonces "$a \rightarrow b$".*

2) *Si "a" es el evento de envío de un mensaje de un proceso y "b" el evento de recepción del mismo mensaje en otro proceso, entonces "$a \rightarrow b$".*

3) *Si "$a \rightarrow b$" y "$b \rightarrow c$" entonces "$a \rightarrow c$". Dos eventos disintos "a" y "b" se dice que son* ***concurrentes*** *si "$a \nrightarrow b$" y "$b \nrightarrow a$"*

Ya que es necesario preservar estas referencias, Lamport propone el uso de **relojes lógicos**. Una forma general para ver a estos relojes es mediante una función, la cual asigna un número a cada evento en cada proceso. Esto puede realizarse mediante el uso de contadores dentro de cada proceso y a la vez, marcando a cada evento con el valor correspondiente.

Mattern [Mat89], argumentando que el uso de una estructura lineal (como los contadores propuestos por Lamport) para el ordenamiento de eventos no es la adecuada para un sistema distribuido, propone el uso de **vectores de tiempo**. El problema que señala Mattern consiste en que una simple estructura no podría

preservar de manera correcta la precedencia de los eventos, ya que en algunos casos existirían inconsistencias sobre todo en aquellos eventos que resulten concurrentes.

El objetivo principal de los vectores de tiempo propuestos por Mattern es el de evitar inconsistencias en el sistema, ya que cada elemento en el vector corresponde al tiempo lógico de cada participante perteneciente al sistema. Mediante el uso de esta estructura cada participante puede tener una vista parcial del estado global del sistema y así, preservar las referencias temporales entre ellos.

Capítulo 3

Trabajo relacionado: sincronización de datos multimedia

En este capítulo se presenta el análisis realizado a los trabajos más cercanos a nuestro trabajo de investigación. Como primer punto, se propone una nueva taxonomía para la clasificación de los trabajos tomando en cuenta como se organiza a los participantes en el sistema. Posteriormente, a partir de nuestra taxonomía, se describen los trabajos que tratan la sincronización de flujos multimedia para la comunicación en grupo.

3.1. Taxonomía del trabajo relacionado

A continuación se presenta una propuesta para la clasificación de los trabajos que tratan el problema de la Sincronización multimedia en la comunicación en grupo (SMCG). Los trabajos son clasificados tomando como base el tipo de enfoque usado (centralizado, descentralizado o distribuido) y subclasificándolos con base en el tipo de referencias utilizadas para la sincronización de MMS; ver figura 3.1.

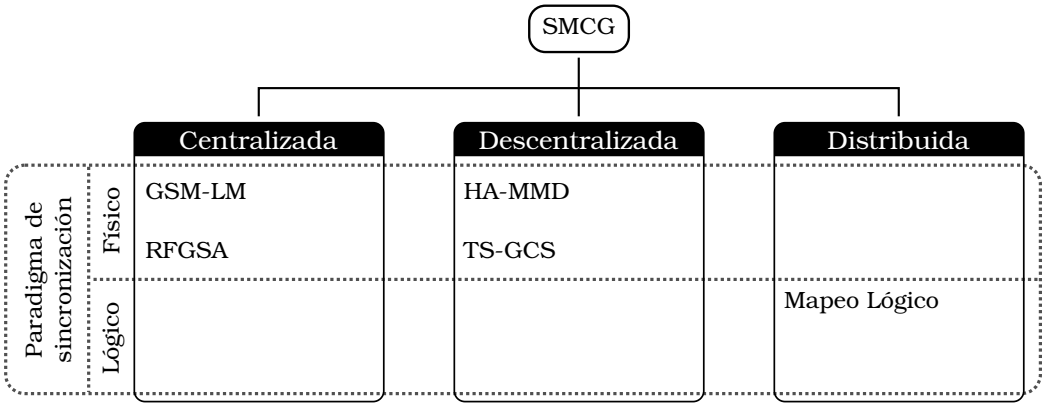


Figura 3.1: Taxonomía para los mecanismos para la sincronización multimedia en grupo.

3.1.1. SMCG centralizado

3.1.1.1. *Group Synchronization Mechanism for Live Media (GSM-LM)*

Yutaka Ishibashi y Shuji Tasaka proponen en [IT97] un mecanismo para la sincronización de MMS en tiempo real enfocado a la comunicación en grupo. Por medio de él se selecciona a uno de los participantes en la comunicación como un gestor o administrador, el cual será el encargado de mantener la sincronización de los datos en los demás participantes (figura 3.2).

El mecanismo está basado en la sincronización intra e inter flujo propuesta en [IT95] en donde se propone el uso de un esquema de sincronización maestro/esclavo a nivel flujos; esto significa que un flujo esclavo deberá sincronizarse con las referencias de un flujo maestro. Para realizar la sincronización de los flujos multimedia se hace uso de los relojes físicos de los participantes, por lo que todos los relojes deberán estar sincronizados, por medio del protocolo NTP, con una referencia de tiempo global.

Los autores, también, definen el objecto *Media Unit (MU)*, que es una serie de datos que componen a un flujo; por ejemplo, los *frames* que configuran a un flujo de

video. Para realizar la sincronización intra e inter flujo cada MU será estampillado con una marca de tiempo (indica el instante de tiempo en que fué generado). Además, los flujos esclavos llevarán un número de secuencia correspondiente al flujo maestro.

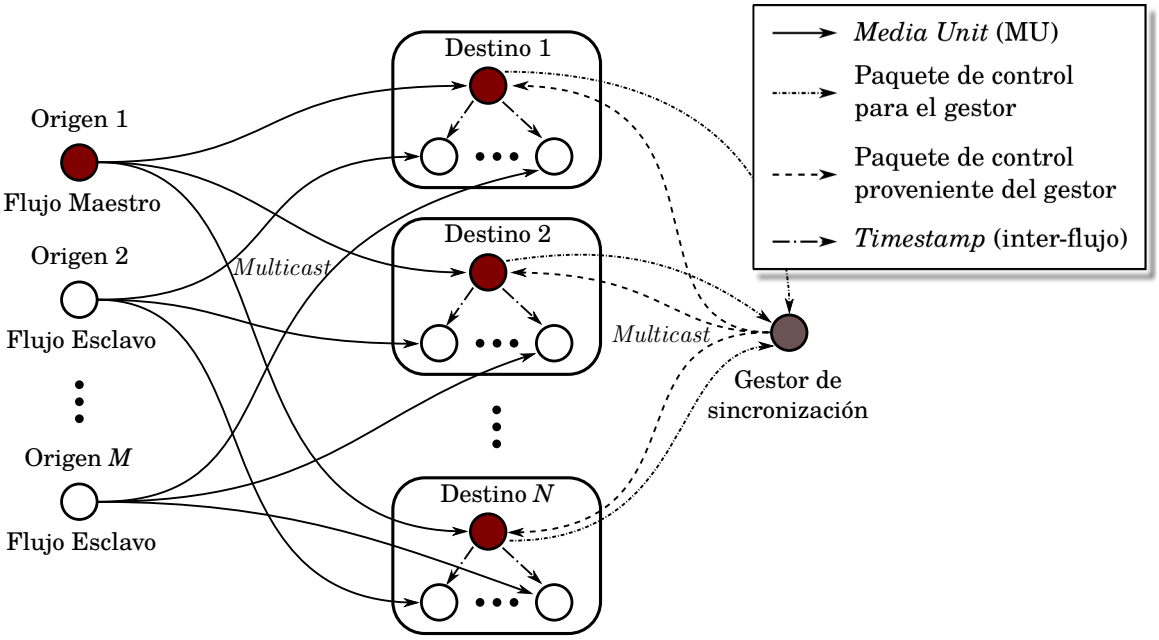


Figura 3.2: Modelo de sincronización de MMS propuesto en [IT97].

Al inicio de la comunicación se supone que los primeros MUs generados llevan la misma marca de tiempo, pero tomando en cuenta que están siendo transmitidos a través de Internet, estos paquetes podrían llegar retrasados y *romper* con la sincronización. Para evitarlo, los destinatarios deben estar calculando periódicamente el tiempo de retraso en la red, para así seleccionar el *tiempo de sincronización* de los MMS y evitar el deterioro en la calidad de salida de los flujos.

Cuando el tiempo estimado para la sincronización es significativamente diferente entre los destinos, se recurre al *gestor* de sincronización para evitar la pérdida de la sincronización. El gestor de sincronización obtiene los tiempos estimados de cada uno de los destinos y ajusta el tiempo de sincronización que envía a todos los destinos mediante paquetes de control.

La principal ventaja que presenta el mecanismo GSM-LM es la reducción de la información de control a enviar por cada participante. La reducción de información de control se logra dado a que cada participante sólo envía las referencias temporales del maestro. Un punto importante a analizar en este mecanismo es la función del gestor de sincronización, éste se encarga de corregir los errores en la sincronización en los destinos. Lo anterior tiene como principal desventaja la generación de cuellos de botella, debido que al incrementar el número de participantes el gestor pueda no atender a estos y no cumplir con restricciones de tiempo real.

3.1.1.2. *RTP-based Feedback Global Synchronisation Approach (RFGSA)*

En la actualidad el protocolo utilizado para la sincronización de datos multimedia es el protocolo RTP. Sin embargo, ya que RTP no brinda soporte a la sincronización en grupo han surgido propuestas para mejorarlo. Boronat et al. [BGL08] para manejar la sincronización de datos multimedia en grupo proponen un enfoque de sincronización global basado en realimentación RTP (RFGSA) por medio del cual se realizan las modificaciones de los paquetes existentes y se agregan nuevos mensajes de control al protocolo.

Mediante la modificación del protocolo RTP, se busca garantizar el mismo tiempo de inicio de salida y obtener un mínimo incremento en la información de control. Para efectuar lo anterior, los autores plantean el uso de dos esquemas maestro/esclavo, el primero para hacer la sincronización en grupo y el segundo para realizar la sincronización inter-flujo (figura 3.3).

La sincronización en grupo se hace seleccionando, primero a un flujo como *flujo maestro*, y su *emisor* será considerado como *fuente de sincronización*. Una vez hecho lo anterior se procede a seleccionar a uno de los receptores como el *receptor*

maestro, éste servirá como referencia para el proceso de sincronización grupal, el punto de reproducción del flujo maestro se toma como referencia. Seleccionados los maestros, se obtiene el tiempo de reproducción inicial del flujo maestro entre los receptores, para satisfacer que éste sea reproducido en el mismo instante de tiempo.

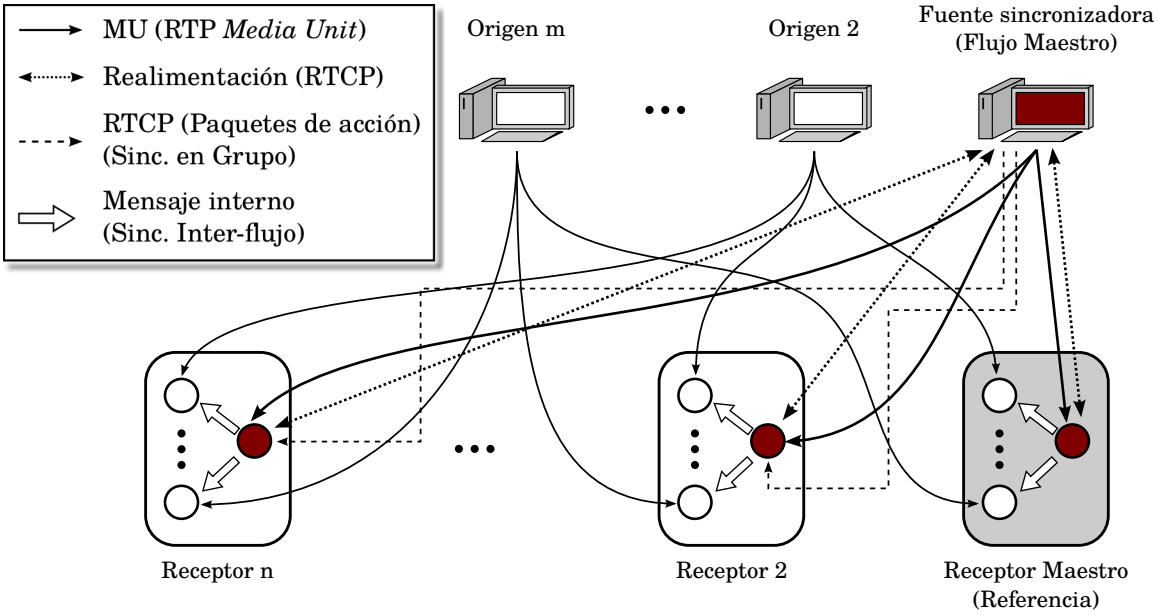


Figura 3.3: Modelo de sincronización en grupo propuesto en [BGL08].

Cuando se usa una marca de tiempo físico, hay que tener en cuenta el retraso existente en la red. Cuando se detecta una asincronía respecto al umbral de reproducción, la fuente de sincronización procede a informar a los receptores *mediante mensajes de acción*, para que ajusten su tiempo de reproducción ya sea *saltando* o *pausando* los MUs recibidos.

Paralelamente a la sincronización en grupo, internamente o localmente en cada uno de los receptores se debe realizar la sincronización inter-flujo. Para efectuar la sincronización, los autores proponen el uso de un *canal* de comunición inter-proceso y es hecha por medio de la información de los puntos de reproducción del

flujo maestro. Si se presenta alguna asincronía, se hace uso de mensajes de acción internos para *saltar* o pausar los MUs.

El mecanismo RFGSA al estar basado en el mecanismo GSM-LM comparte características similares y las mismas desventajas. Otra desventaja que presenta este mecanismo es la adicción de nuevos paquetes de control, esto incrementa el flujo de mensajes en la red por cada flujo y enfocándonos en una comunicación grupal, no brinda soporte a la escalabilidad.

3.1.2. SMCG descentralizado

3.1.2.1. *Hierarchical Architecture for Multimedia Distribution (HA-MMD)*

Yen y Akyildiz [YF96] proponen una arquitectura jerárquica para minimizar el tamaño del *buffer* utilizado para mantener las relaciones temporales entre MMS durante el transcurso de la comunicación. Esta arquitectura consiste en agrupar a los *usuarios* en la comunicación en subgrupos dependiendo de su vecindad. Un usuario va a ser vecino de otros siempre y cuando el límite del buffer de éstos sea menor al de éste.

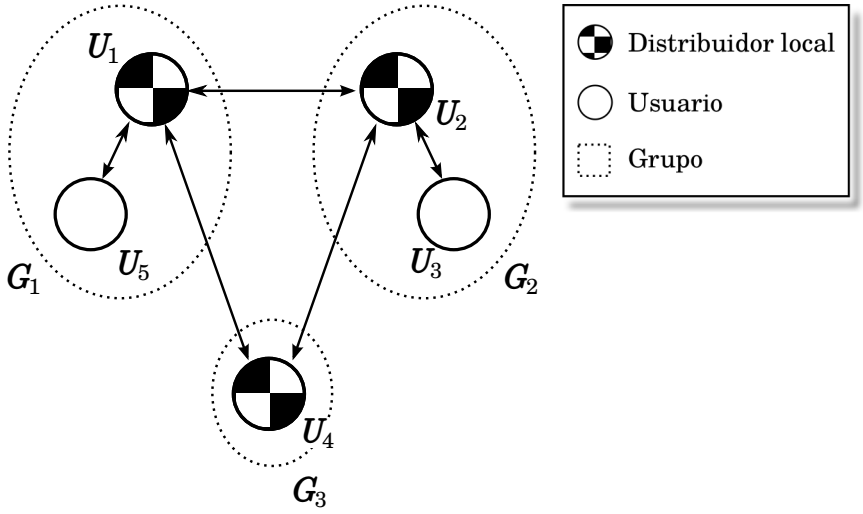


Figura 3.4: Arquitectura jerárquica propuesta en [YF96].

La comunicación entre subgrupos es hecha seleccionando a uno de los usuarios de cada subgrupo como *Distribuidor Local*, el cual va ser responsable de transmitir los datos enviados de un usuario de su subgrupo además de recibir los datos enviados de otros subgrupos. En la figura 3.4 se presenta la configuración de la arquitectura propuesta por Yen y Akyildiz.

Para la sincronización multimedia se propone resolver cuatro problemas: la diferencia de tiempo en los relojes locales, la variabilidad en el retraso de mensajes, el cálculo del tiempo global de recolección inicial y el tiempo global para el inicio de la reproducción. Para solucionar la diferencia de tiempo en los relojes locales, para este mecanismo se propone el uso de un reloj global. El problema del retraso de mansajes se resuelve mediante un pre-almacenamiento en el buffer, este pre-almacenamiento está basado en el trabajo realizado por Ramanathan y Rangan [RR93]. El tiempo global de recolección inicial sirve para almacenar dentro del buffer por un cierto tiempo la información recibida por distribuidor global y mediante este proceso, reducir el tiempo de retraso de los mensajes. El tiempo global de recolección se calcula durante un proceso de renegociación entre los distribuidores locales. El tiempo global para el inicio de la reproducción sólo se calcula en caso de que la aplicación en cuestión necesite que todos los flujos multimedia se reproduzcan al mismo tiempo.

La principal desventaja de este mecanismo es el uso de participantes como intermediarios en la comunicación, ya que al tener estos la responsabilidad de recibir, sincronizar y retransmitir la información, podrían generarse cuellos de botella y afectar la comunicación. Otro problema que presenta es el uso de un reloj global, ya que cada participante en el sistema deberá estar actualizando su reloj local con el reloj global constantemente. El mecanismo HA-MMD tampoco asegura la sincronización de los mensajes generados dentro de cada subgrupo, lo cual podría afectar la reproducción de estos en los demás subgrupos.

3.1.2.2. *Topological Structure for the Group Communication System (TS-GCS)*

Una arquitectura jerárquica para la sincronización de MMS y entrega causal en tiempo real es propuesta en [AB98, BA02], basada en formar *k grupos locales*; donde *k* es el total de grupos que componen al sistema de comunición (ver figura 3.5).

Para formar la arquitectura jerárquica los autores proponen hacer cuatro pasos: (1) cada proceso analiza su capacidad para brindar soporte a las conexiones de sus vecinos, una vez obtenido lo anterior, se procede a generar un conjunto de procesos llamado *grupo de tránsito*; (2) ya formado el grupo de tránsito, proceder a generar los grupos locales; (3) elegir a un proceso como servidor local, el cual será el encargado de intercambiar los mensajes de su grupo con otros grupos; por último (4) constituir un conjunto de servidores locales, llamado *conjunto virtual*, y seleccionar a uno de los servidores locales como *servidor maestro*.

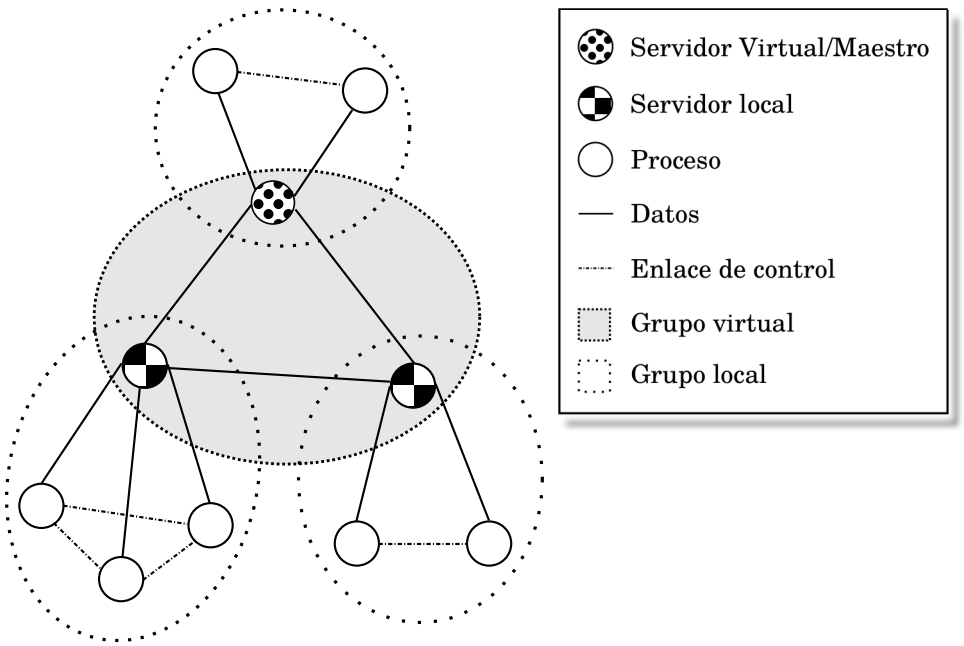


Figura 3.5: Arquitectura jerárquica propuesta en [AB98, BA02].

Para realizar la sincronización multimedia, en este mecanismo se propone el uso de un protocolo Δ-causal. Mediante este protocolo se propone el uso de un tiempo de vida Δ de un mensaje para la recepción de un mensaje causal. Mediante el mecanismo se propone resolver los problemas de la diferencia de tiempo en los relojes locales y la variabilidad en el retraso de mensajes mediante el uso de un reloj global, además de estimar un tiempo de vida y de entrega global. Como reloj global del sistema se propone el uso de un tiempo virtual, el cual es proporcionado por el proceso seleccionado como servidor maestro; se nombra como tiempo del maestro virtual (*Virtual Master Time*, VMT). Para mantener los relojes sincronizados en todos los procesos en el sistema, el servidor maestro debe enviar periódicamente el valor de su VMT para mantenerlos sincronizados en un intervalo de 5 a 10 ms.

El retraso de los mensajes es un problema de suma importancia, ya que se generan inconsistencias y problemas durante la reproducción de éstos. Para solucionar este problema, en el mecanismo se propone estimar un tiempo de vida global (Δ). Una vez que se ha formado el grupo virtual y se ha seleccionado el servidor maestro, se procede a calcular el tiempo Δ. El tiempo Δ estará limitado por el tiempo mínimo entre un servidor local con respecto a su grupo local, además de un tiempo máximo existente entre dos procesos distanciados. Este tiempo de vida sirve para garantizar la entrega causal de mensajes. Una vez que han sido sincronizados los relojes y se ha calculado el tiempo de vida global, se procede a calcular el tiempo de entrega global. El tiempo de vida global sirve para que un mensaje sea entregado en todos los procesos de manera simultánea.

Este mecanismo presenta desventajas similares al mecanismo HA-MMD, ya que se deben mantener sincronizados todos los relojes mediante una referencia global además de la generación de cuellos de botella. La principal ventaja que presenta el mecanismo HA-MMD es el uso de un protocolo Δ-causal, ya que mediante el tiempo de vida se asegura la entrega de los mensajes causalmente relacionados.

3.1.3. SMCG distribuido

3.1.3.1. Mapeo Lógico (ML)

Pomares et al. proponen en [PEMR09] un mecanismo para la sincronización de MMS sin hacer uso de referencias globales (como el tiempo físico) y memoria compartida. El mecanismo de sincronización se basa en el modelo de sincronización de ML [PMER08], que expresa las relaciones temporales entre los datos de acuerdo a sus dependencias causales.

En este mecanismo los datos multimedia son representados mediante eventos discretos y continuos (intervalos); un evento se denota por el envió ($send(m)$) y la entrega ($delivery(m)$) de un mensaje m. Los intervalos están compuesto por una serie de mensajes, a los cuales, se les identifica con un punto de inicio (*begin*, denotado por el mensaje más a la izquierda v^-) y un punto final (*end*, por el mensaje más a la derecha v^+). Un evento discreto sólo se reconoce como el envió de un mensaje, el cual sólo es identificado como un punto (ejemplo: el mensaje m_1).

Lamport establece en [Lam86] que un intervalo A ocurre antes de un intervalo B si todos los elementos que componen A preceden causalmente a todos los elementos de B. Pomares et al. [PMER08] proponen que la relación *happened-before* puede ser expresada sólo en términos de sus extremos de la siguiente manera:

Propiedad 1 *Sean A, B y C conjuntos de eventos secuencialmente ordenados. El conjunto de eventos de A ocurre antes del conjunto de eventos en B si alguna de las siguientes condiciones se cumple:*

1. $A \rightarrow B$ *si* $a^+ \rightarrow b^-$
2. $A \rightarrow B$ *si* $\exists C$, *tal que* $a^+ \rightarrow c^- \wedge c^+ \rightarrow b^-$

Finalmente, se presenta la relación de concurrencia para intervalos ordenados, la cual se define a continuación:

Definición 3.1 *Sean A y B dos intervalos ordenados. Se dice que A y B son simultáneos (denotado por $A|||B$) si se cumple la siguiente condición:*

$$A|||B \text{ si } a^-||b^- \wedge a^+||b^+ \tag{3.1}$$

La definición anterior significa que un intervalo de A puede tener lugar en el "mismo tiempo" que otro intervalo B.

El modelo ML hace la sincronización de los MMS basándose en las relaciones temporales *intervalo-intervalo*, *punto-intervalo* y *punto-punto* propuestas previamente por Allen [All83] y Vilain [Vil82] respectivamente. La traducción de mapeo lógico involucra a cada par de intervalos de una relación temporal (tabla 3.1), donde cada intervalo es etiquetado como X o Y.

En la tabla 3.1 se muestra una serie de reglas para la sincronización de los intervalos. Las reglas indican que para todo intervalo X/Y en la comunicación, éstos deberán segmentarse mediante las relaciones especificadas por $I \times I$ para formar los subintervalos A, B, C, D, W y S.

Tabla 3.1: Mapeo Lógico [PMER08]

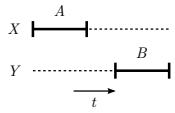
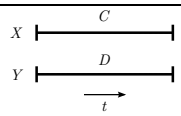
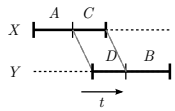
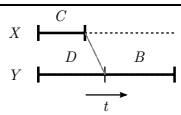
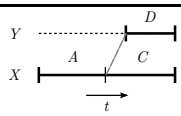
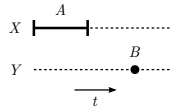
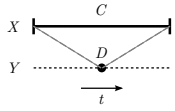
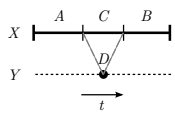
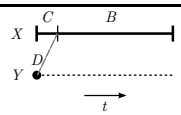
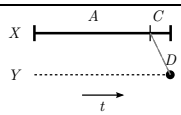
$\forall(X,Y)$	$\in$	$I \times I$
$A(X,Y)$	$\leftarrow$	- si $x^- \rightarrow y^-$, $\{x \in X : delivery(Part(Y),x) \rightarrow send(y^-)\}$ - de otra manera, $\emptyset$
$C(X,Y)$	$\leftarrow$	- si $y^+ \rightarrow x^+$, $\{x \in X : send(x) \rightarrow delivery(Part(Y),y^+)\} - A(X,Y)$ - de otra manera, $X - A(X,Y)$
$D(X,Y)$	$\leftarrow$	- si $x^+ \rightarrow y^+$, $Y - \{y \in Y : delivery(Part(Y),x^+) \rightarrow send(y)\}$ - de otra manera, Y
$B(X,Y)$	$\leftarrow$	- si $y^+ \rightarrow x^+$, $X - \{A(X,Y) \cup C(X,Y)\}$ - de otra manera, $Y - D(X,Y)$
$W(X,Y)$	$\equiv$	$C(X,Y)\|\|\|D(X,Y)$
$S(X,Y)$	$\equiv$	$A(X,Y) \rightarrow_I W(X,Y) \rightarrow_I B(X,Y)$

En el modelo se identifican cinco mapeos lógicos: *precede*, *traslape*, *fin*, *inicio* y *simultaneo* (tabla 3.2), mediante las cuales se representan las posibles relaciones temporales entre subintervalos. Estos mapeos también muestran como se pueden manejar las relaciones entre los dos tipos de *datos* identificados previamente.

Para preservar las relaciones temporales, el mecanismo utiliza relojes lógicos, representados mediante el uso de vectores. Mediante el indentificador de cada proceso se incrementa el valor de la posición correspondiente, este proceso se realiza al envío y recepción de mensajes. Además de hacer uso del vector de tiempo, cada proceso enviá junto a los datos un historial causal, en el cual se identifica con que procesos está relacionado.

El mecanismo ML, al no usar referencias globales, reduce los mensajes de control a utilizar para la sincronización además de evitar la sincronización de relojes físicos periódicamente. Aunque se evita el envío de más mensajes de control, el mecanismo no provee soporte al incremento en el número de procesos. Lo anterior se debe a que un proceso puede enviar información de control con respecto a los demás procesos en el sistema, incrementando el número de referencias temporales por cada mensaje.

Tabla 3.2: Tabla de relaciones del Mapeo Lógico [PMER08]

	Relación	Ejemplo	Extremos
Intervalo-intervalo	*precede* $A \rightarrow_I B$		$a^+ \rightarrow b^-$
	simultaneo $(C\|\|\|D)$		$c^- \parallel b^-$, $c^+ \parallel b^+$
	traslape $A \rightarrow_I (C\|\|\|D) \rightarrow_I B$		$a^+ \rightarrow c^-$, $a^+ \rightarrow d^-$ $c^- \parallel d^-$, $c^+ \parallel d^+$ $c^+ \rightarrow b^-$, $d^+ \rightarrow b^-$
	inicio $(C\|\|\|D) \rightarrow_I B$		$c^- \parallel d^-$, $c^+ \parallel d^+$ $c^+ \rightarrow b^-$, $d^+ \rightarrow b^-$
	fin $A \rightarrow_I (C\|\|\|D)$		$a^+ \rightarrow c^-$, $a^+ \rightarrow d^-$ $c^- \parallel d^-$, $c^+ \parallel d^+$
Punto-intervalo	*precede* $A \rightarrow_I B$		$a^+ \rightarrow b$
	simultaneo $(C\|\|\|D)$		$c^- \parallel d$, $c^+ \parallel d$
	traslape $A \rightarrow_I (C\|\|\|D) \rightarrow_I B$		$a^+ \rightarrow c^-$, $a^+ \rightarrow d$ $c^- \parallel d$, $c^+ \parallel d$ $c^+ \rightarrow b^-$, $d \rightarrow b^-$
	inicio $(C\|\|\|D) \rightarrow_I B$		$c^- \parallel d$, $c^+ \rightarrow b^-$ $d \rightarrow b^-$
	fin $A \rightarrow_I (C\|\|\|D)$		$a^+ \rightarrow c^-$, $a^+ \rightarrow d$ $c^- \parallel d$

Capítulo 4

Mecanismo escalable para sincronización multimedia

Acorde a la metodología, el presente capítulo muestra el mecanismo desarrollado para la sincronización multimedia en grupo. El capítulo se divide en cuatro secciones, la primera sección muestra definiciones base para el desarrollo del mecanismo. En la segunda, sección se explica detalladamente el modelo organizacional propuesto para la configuración del sistema. En la tercera sección, se presenta un esquema jerárquico para la sincronización multimedia y por último, en la cuarta sección se detalla el mecanismo mediante la presentación de los algoritmos desarrollados.

4.1. Preliminares

4.1.1. Modelo del sistema

Para el funcionamiento de nuestro mecanismo suponemos que está siendo ejecutado en un sistema distribuido, el cual se compone de procesos que se están comunicando mediante el intercambio de mensajes. El sistema no cuenta con memoria compartida ni hace uso de referencias globales. Para el envío de mensajes suponemos que todos los participantes lo hacen mediante *broadcast*.

Procesos: La aplicación bajo consideración se compone de un conjunto de pro-

cesos $P = \{i, j, ...\}$ organizados en un grupo. Los procesos sólo pueden enviar un flujo de información a la vez. Los procesos pueden pertenecer a una o más computadoras conectadas en el sistema.

Mensajes: Consideramos un conjunto finito de mensajes M, donde un mensaje $m \in M$ es identificado por la tupla $m = (p, x)$, donde $p \in P$ es el emisor de m y x es el reloj lógico para el mensaje de p cuando éste es enviado. El conjunto destino de un mensaje m siempre es P (ver figura 4.1).

Eventos: Sea m un mensaje. Se denota por $send(m)$ al evento de envío y por $delivery(p, m)$ el evento de recepción de m en un proceso $p \in P$. El conjunto de eventos asociados a M es el conjunto $E = \{send(m) : m \in M\} \cup \{delivery(p, m) : m \in M \wedge p \in P\}$.

Intervalos: Se considera un conjunto finito de intervalos I, donde cada intervalo $A \in I$ es un conjunto de mensajes $A \subseteq M$ enviados por un proceso $p = Part(A)$, definido por el mapeo $Part : I \to P$. Se denota por a^- y a^+ los extremos de un mensaje A y dado por el orden secuencial de $Part(A)$, tenemos que para todo $m \in M : a^- \neq m$ y $a^+ \neq m$ implica $a^- \to m \to a^+$. Cuando $|A| = 1$ (MMS discretos), se tiene que $a^- = a^+$; en este caso, a^- y a^+ son denotados indistintamente por a.

4.2. Modelo organizacional

En esta sección se propone un modelo para organizar a los participantes en el sistema, el cual es independiente de la topología de red y la distribución de los participantes. El modelo organizacional se basa en la agrupación de los participantes en subgrupos, dividiéndolos mediante la distancia que existe entre ellos y posteriormente se jerarquizan los subgrupos creados. Cada subgrupo deberá con-

tar con un solo proceso que funja como coordinador o maestro, el cuál servirá como referencia para realizar la sincronización de los MMS generados por cada proceso en el interior del subgrupo correspondiente.

La sección está dividida de la siguiente manera, como primer punto se explica cómo se realiza la agrupación de los participantes, posteriormente se detalla el proceso que se debe seguir para la jerarquización de los subgrupos y por último, se muestra la política para la selección de los coordinadores/maestros.

4.2.1. Formación de subgrupos

Como se mencionó anteriormente, nuestro modelo organizacional consiste en dividir P en subgrupos. Para la organización de los procesos consideramos que todos los procesos en el sistema forman un solo grupo lógico (P), por lo que forman un *grafo completo*; ver figura 4.1, esto significa que un proceso p_i envía y recibe información de todos los demás procesos en P.

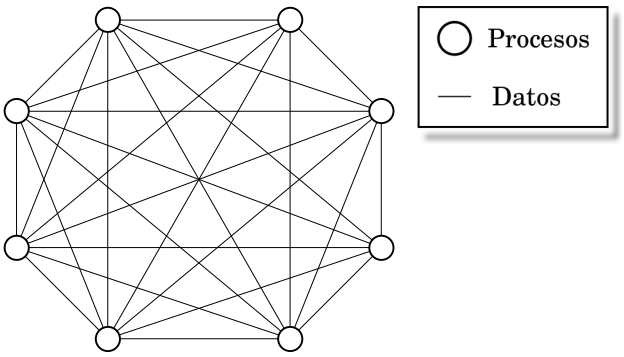


Figura 4.1: Intercambio de mensajes entre los procesos pertenecientes a P.

La división y agrupación de los procesos se realizará mediante el uso de la distancia temporal ($\delta(p_j, p_j)$) entre procesos vecinos [NET05]. La distancia $\delta(p_j, p_j)$ entre los procesos p_i y p_j es obtenida por un tiempo de ida y vuelta $d_{ij} + d_{ji}$, donde d_{ij} es el tiempo que tarda en ser recibido un mensaje enviado de p_i a p_j.

Para realizar la división y agrupación en subgrupos, cada proceso debe obtener la distancia temporal existente entre él y cada uno de los procesos presentes.

Una vez que cada proceso finaliza la obtención de las distancias, cada uno debe crear un conjunto con los identificadores de sus vecinos más cercanos y mediante el cual se formarán los subgrupos; nombraremos al conjunto como NS. Para formar el conjunto NS, primero cada proceso debe obtener la distancia máxima (δ_{max}) a la que puede considerar a un proceso como su vecino. La distancia δ_{max} se obtiene sumando todas las distancias y posteriormente dividiendo el resultado entre el total de procesos menos uno (4.1). Después de haber obtenido la distancia δ_{max} se procede a agregar los identificadores de los vecinos más cercanos al conjunto NS. Para considerar que un proceso es un vecino cercano, la distancia entre un proceso origen y un proceso destino debe ser menor o igual a δ_{max}. Por último, formados los conjuntos los procesos entran en una fase de negociación mediante la cual se evita que un proceso pueda pertenecer a varios subgrupos; a esto se le llama subgrupos traslapados.

$$\delta_{max} = \frac{\sum \delta(p_i, p_j)}{|P| - 1} \tag{4.1}$$

4.2.2. Formación de niveles de jerarquía

Como se ha mencionado anteriormente, el modelo organizacional se basa en la agrupación y posterior jerarquización de subgrupos de procesos. La agrupación es de gran utilidad para distribuir el trabajo en diferentes entidades. El jerarquizar los subgrupos sirve para mantener la sincronización en todo el sistema, tomando como referencia los MMS generados dentro de cada subgrupo.

Para formar la estructura jerárquica del sistema, primero se deben crear subgrupos de procesos partiendo del grupo original. Una vez obtenida la distancia

máxima y los subgrupos, los subgrupos creados formarán el primer nivel como nivel de la jerarquía y a su distancia será identificada como d_0; al primer nivel se le llamará *nivel 0*. Después de concluido el proceso de formación de los subgrupos de nivel 0, en cada uno de ellos se debe de seleccionar a un proceso que va a fungir como coordinador/maestro. Seleccionados los coordinadores/maestros, se debe obtener una segunda distancia (a la que llamaremos d_1), la cual es más grande que la primer distancia obtenida ($d_0 < d_1$). Esta nueva distancia servirá para agrupar a los maestros en subgrupos de coordinadores (figura 4.2) y el proceso de formación de estos nuevos subgrupos es muy similar al proceso de formación de los subgrupos de procesos.

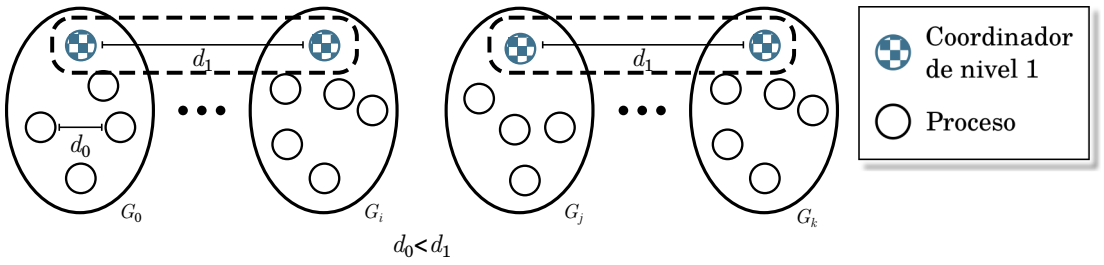


Figura 4.2: Subgrupos de coordinadores conectando subgrupos de procesos.

Tras la nueva agrupación podemos obtener subgrupos de coordinadores/maestros que estén *separados* (figura 4.2), dado que su distancia no entra en el rango de los demás coordinadores. Para solucionar el problema anterior, se plantea seleccionar a uno de estos coordinadores como el coordinador de ese subgrupo de coordinadores, elevándolo a un nuevo nivel de jerarquía con respecto a los demás procesos. Seleccionados los coordinadores de este nuevo nivel, se procede a obtener una tercera distancia (d_2) y se repite el mismo proceso de formación de subgrupos. Este mismo proceso puede repetirse hasta conectar y formar una arquitectura como la que se puede ver en la figura 4.3. El total de niveles usados para la formación de la arquitectura dependerá de las restricciones y requerimientos con las que cuente

la aplicación en cuestión.

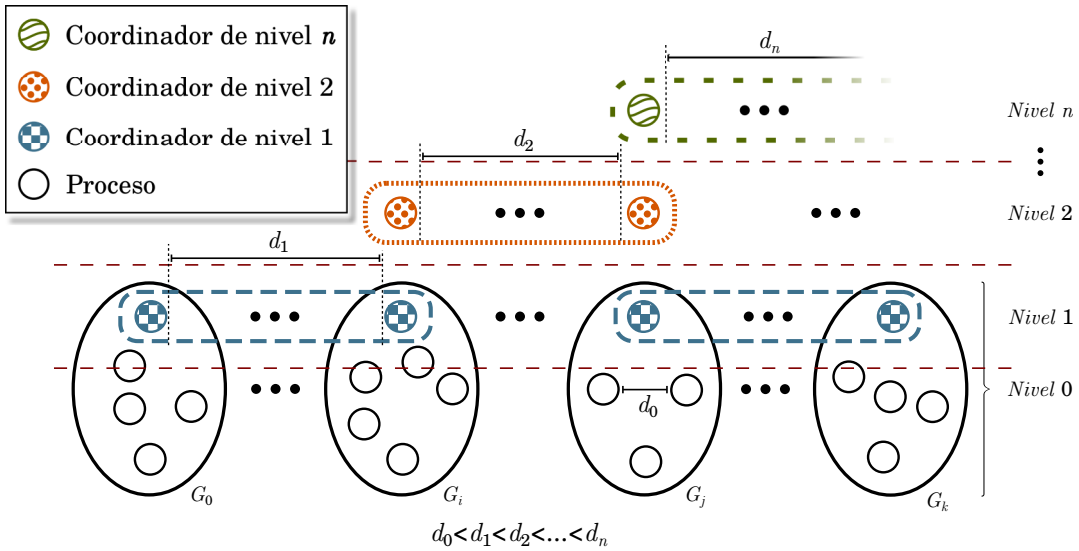


Figura 4.3: Modelo organizacional propuesto para la sincronización de flujos multimedia.

Cabe destacar que el proceso de obtención de las distancias se deberá realizar periódicamente o al momento en que un proceso ingrese o abandone el sistema. Lo anterior es de suma importancia ya que al cambiar las distancias, un proceso pudiese pasar a formar parte de otro subgrupo pero al mismo tiempo pertenecer a un subgrupo que no le corresponda. Los tiempos para obtener las nuevas distancias y la modificación de los subgrupos quedarán a cargo de la aplicación para la cual este modelo sea utilizado.

4.2.2.1. Selección de coordinadores/maestros

Hasta ahora no se ha tocado el tópico de la política de selección de coordinadores/maestros, aquí se detalla cómo se lleva acabo. La función de cada maestro dentro de cada subgrupo es servir como referencia para realizar la sincronización de los MMS *locales*.

Para seleccionar el maestro un proceso deberá utilizar la lista de sus vecinos,

la cual suponemos que está ordenada mediante los identificadores de los procesos y de manera creciente. La primera vez que selecciona un maestro se elige al primer proceso de la lista, la siguiente vez se selecciona al que le sigue y así sucesivamente; ver el diagrama de la figura 4.4. Ya que los procesos pueden estar o no trasmitiendo datos, cada vez que se busca un nuevo maestro se debe verificar que ese proceso esté activo[1]. De no encontrar un proceso vecino que esté activo y que pueda ser seleccionado como maestro, el proceso que está buscando un nuevo maestro pasará a ser el maestro de ese subgrupo.

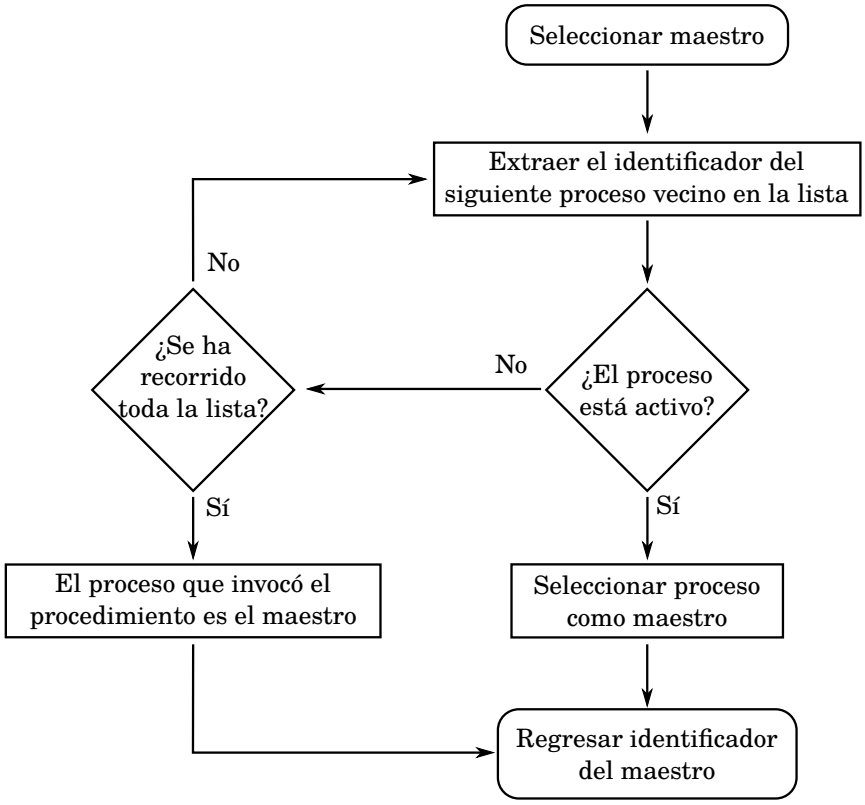


Figura 4.4: Proceso para la selección del proceso maestro.

El proceso de búsqueda de maestro se deberá realizar al inicio de la comunicación y cada vez que el maestro deje de transmitir información para evitar errores y

[1]Se denomina que un proceso está activo cuando éste transmite un flujo continuo (video/audio).

pérdidas en la sincronización de los datos.

Como ya se ha mencionado, cada subgrupo debe de contar con un proceso que funja como maestro, por lo que la búsqueda de éste se deberá realizar en cada uno de los niveles de jerarquía (figura 4.5).

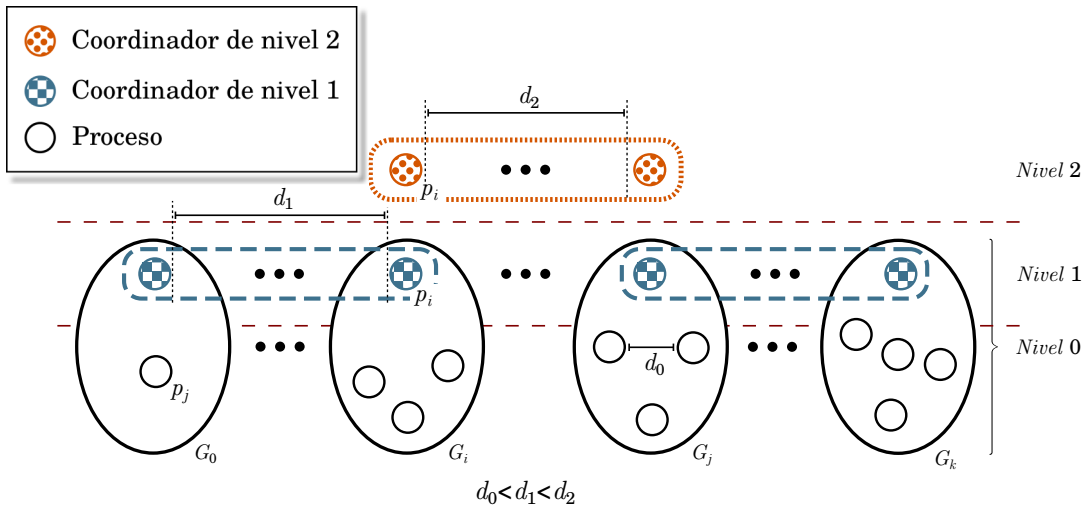


Figura 4.5: Arquitectura jerárquica de tres niveles.

4.2.2.1.1. Algoritmo para la selección de coordinadores/maestros

En este apartado se presenta el algoritmo desarrollado para la selección de coordinadores/maestros. La política de selección presentada en el apartado anterior es aplicada en el algoritmo 1, manipulando al conjunto NS como una cola. La búsqueda del proceso maestro es realizada pasándole al algoritmo el nivel de jerarquía en el que se necesita buscar. Habiendo encontrado el proceso maestro el algoritmo regresa su identificador, en caso contrario regresa un valor nulo.

Algoritmo 1 Procedimiento para la selección del proceso maestro

```
1:  procedure getMasterID(level)
2:      if Not(ActP = ∅) then                          ▷ Se verifica que existan procesos activos.
3:          id =dequeue(NS[level])
4:          if ∃x ∈ ActP|x = id then                   ▷ Si el primer proceso en el conjunto NS_i está activo
5:              return id                              ▷ es seleccionado como el proceso maestro.
6:          else
7:              enqueue(id,NS[level])
8:              x =dequeue(NS[level])                  ▷ Si el primer proceso no está activo, se busca
9:              while Not(id = x) and Not(∃y ∈ ActP|y = x) do   ▷ el siguiente proceso activo
10:                 enqueue(x,NS[level])               ▷ dentro del conjunto NS_i.
11:                 x =dequeue(NS[level])
12:             end while
13:             if Not(id = x) then
14:                 return x   ▷ De encontrar un proceso activo se devuelve el indentificador de éste.
15:             else
16:                 enqueue(x,NS[level])
17:                 return 0                           ▷ De lo contrario se retorna un cero.
18:             end if
19:         end if
20:     else
21:         return 0
22:     end if
23: end procedure
```

4.3. Esquema de sincronización

El principal objetivo de esta investigación es proporcionar un mecanismo de sincronización escalable, por lo que en este apartado se propone un esquema jerárquico para la sincronización de flujos multimedia. A través de este esquema se reduce el trabajo de los procesos y el número de información de control a enviar en cada mensaje al jerarquizar los flujos.

Ishibashi et al. proponen en [IT97] un esquema maestro/esclavo (Master/Slave (M/S)) para la sincronización de MMS, el cual consiste en agrupar a los receptores en maestro y esclavos. En este esquema los esclavos deben de sincronizar

sus flujos con respecto a las referencias temporales del maestro. Basándonos en el esquema propuesto en [IT97], en este trabajo se propone un esquema M/S escalonado (figura 4.6) aplicado a MMS. Nuestro esquema de sincronización toma como referencia la arquitectura propuesta en la sección anterior, ya que los coordinadores/maestros de cada subgrupo servirán como referencia para realizar la sincronización entre MMS.

Para explicar de mejor manera lo anterior tomemos como ejemplo la figura 4.6, supongamos que los procesos p_j y p_k pertenecen a un mismo subgrupo de nivel 0; donde p_j es el coordinador/maestro de ese subgrupo y p_k es un esclavo. Como p_k es un esclavo, se propone que éste sólo se deberá sincronizar con el flujo multimedia transmitido por p_j.

Supongamos que la figura 4.6 pertenece a una arquitectura de 2 niveles de jerarquía donde p_j es coordinador de un subgrupo de nivel 0, pero también forma parte del subgrupo de coordinadores de nivel 1; por lo que es vecino de p_i. Para realizar la sincronización en este punto supongamos que p_i es el maestro del subgrupo de coordinadores, por lo cual p_j se deberá sincronizar con el flujo transmitido por p_i y a la vez preservar las referencias con respecto a sus esclavos.

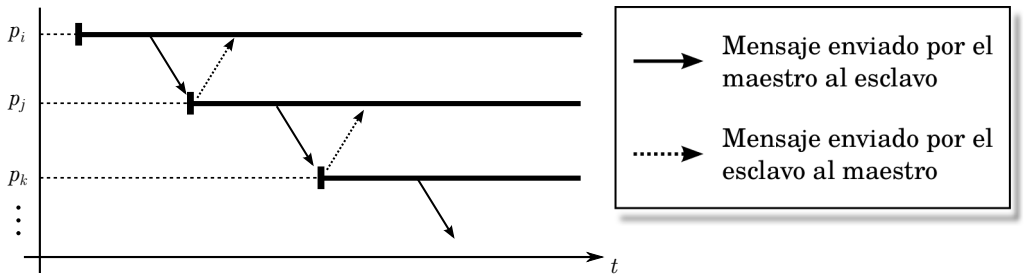


Figura 4.6: Esquema maestro/esclavo escalonado.

El Internet es un sistema distribuido asíncrono, por lo que el tiempo generación y de transmisión de mensajes no esta definido. Otro dato importante es que en

los sistemas multimedia sobre Internet hay dos tipos de datos (discretos y continuos) interactuando y se deben de preservar las referencias temporales entre ellos. Tomando en cuenta lo anterior, para nuestro esquema se consideran tres casos importantes, los cuales se describen a continuación:

1) *Sincronización con cambio de maestro.* Dado que el Internet es un sistema asíncrono, debemos tomar en consideración cuando el maestro de un subgrupo deje de transmitir y así evitar errores de sincronización. Para explicar este caso usaremos el ejemplo de la figura 4.7.

   Supongamos que p_j es seleccionado como el maestro de un subgrupo dado que es el primero en la lista, por lo que tanto p_i como p_k son sus esclavos. p_j está transmitiendo el flujo s_1, pero después de cierto tiempo deja de transmitirlo. Como p_j no tiene un flujo activo éste deja de ser el maestro del subgrupo, por lo que tanto p_i como p_k deberán de seleccionar a un nuevo maestro. Suponiendo que p_i es seleccionado como maestro, p_k deberá sincronizarse con el flujo s_0 sólo cuando éste empiece a transmitir el flujo s_3.

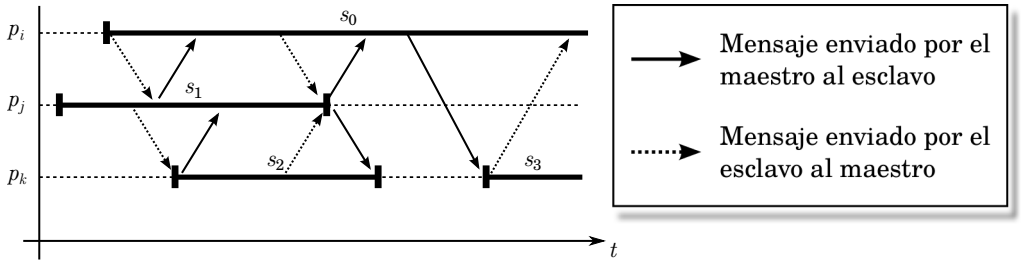


Figura 4.7: Sincronización con cambio de maestro.

2) *Sincronización con ausencia momentánea de maestro.* Este caso es similar al cambio de maestro, pero al no existir un maestro activo pudiera entenderse que se rompe con la sincronización.

En la figura 4.8 se puede observar que tanto p_i como p_j dejan de transmitir s_0 y s_1 antes de que p_k empiece a transmitir s_2. También, se puede observar que no hay ningún otro proceso transmitiendo en el instante en que p_k comienza a trasmitir. Basándonos en la *causalidad*, s_0 precede a s_2 ($s_0 \rightarrow s_2$), por lo que no se rompe en ningún momento la sincronización. En cuanto al maestro, al no existir otro proceso transmitiendo, p_k pasaría a ser el maestro de ese subgrupo.

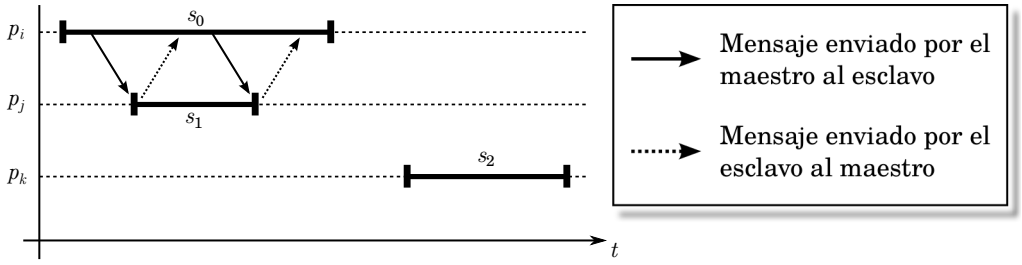


Figura 4.8: Sincronización con ausencia momentanea de maestro.

3) Sincronización M/S entre flujos continuos y discretos. En los SM actuales se intercambian distintos tipos de datos entre los participantes, debido a esto es importante que los mecanismos de sincronización brinden soporte a la heterogeneidad entre datos. Anteriormente ya se ha mencionado que hay dos tipos de datos, de los cuales los discretos son considerados solo un evento y los continuos como una serie de eventos en un intervalo de cierta duración. Dado que la duración de un *evento discreto* es menor a la de un *evento continuo*, un proceso que transmita eventos discretos no puede ser seleccionado como un maestro.

En la figura 4.9 podemos observar que p_j sólo está transmitiendo eventos discretos. Suponiendo que p_j pudiese estar en posición de ser maestro, éste no podría serlo dado al tipo de dato que transmite. Ya que p_j no puede ser

seleccionado como maestro, se seleciona algún otro proceso *activo* (en este caso p_i) para que ocupe el rol de maestro de ese subgrupo.

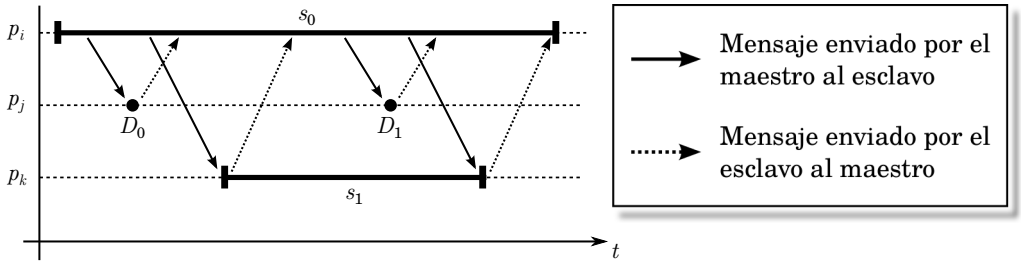


Figura 4.9: Sincronización M/S entre flujos continuos y discretos.

4.4. Mecanismo escalable de Mapeo Lógico Jerárquico

En esta sección se presenta el mecanismo desarrollado, lo llamaremos Mapeo Lógico Jerárquico (MLJ). Este mecanismo es una extensión al mecanismo de ML presentado en [PEMR09], aplicando el modelo organizacional y el esquema de sincronización M/S desarrollados. Como primer punto se presentan la estructura de los mensajes y las estructuras de datos a utilizar en la sincronización. Después, se presentan los algoritmos desarrollados, seguido de la descripción general del funcionamiento del mecanismo. Por último, se presentan las funciones para el manejo de retardo de mensajes.

4.4.1. Mensajes y estructuras de datos

Mensajes. Para la comunicación de nuestro algoritmo se utilizan mensajes formados por la tupla $m = (i, t, MS, TP, H(m), data)$ donde:

- i es el identificador de la fuente o emisor y denota al proceso que envía el mensaje m.

- $t = VT(p)[i]$ es el valor del reloj local del proceso con identificador i cuando un mensaje es enviado. El valor t es el número secuencial del mensaje actualizado antes de su envío.

- MS es el identificador del proceso que funge como maestro del subgrupo al que corresponde el proceso emisor.

- TP contiene el tipo del mensaje (*begin*, $fifo\_p$, *cut*, *end*, *discrete*).

- $H(m)$ contiene la información causal del mensaje, formada por duplas (i, t) que contienen la información causal de los mensajes que precede al mensaje m. La estructura es formada antes del envío del mensaje y es ligada a éste durante su envío. Mediante el uso de esta estructura se asegura la entrega causal del mensaje. Para mensajes del tipo $FIFO$ el valor siempre será $H(m) = \emptyset$.

- *data* es la información enviada por la aplicación en cuestión y es ligada al mensaje durante su envío.

Estructuras de datos. El estado de un proceso está definidio por la estructuras: $VT(p)$, $CI(p)$, $last\_fifo(p)$, $Slaves(p)$ y $ActP(p)$.

- $VT(p)$ es el vector de tiempo de un proceso i. El tamaño del vector está dado por el número de procesos presentes en el sistema. Cada proceso tiene un elemento $VT(p)[i]$ donde i es el identificador del proceso. El vector permite la entrega causal de mensajes debido a que contiene el máximo número de mensajes que el proceso ha visto en un orden causal.

- $CI(p)$ es un conjunto formado por entradas (i, t). Cada elemento en $CI(p)$ denota un mensaje que puede no ser entregado en orden causal por el participante p.

- $last\_fifo(p)$ contiene duplas (i, t) con los identificadores de los últimos mensajes $FIFO$ recibidos. Esta estructura es de suma importancia ya que contiene información de mensajes potencialmente causales.

- $Slaves(p)$ contiene los indentificadores de los procesos que fungen como esclavos para un cierto *proceso maestro*.

- $ActP(p)$ esta estructura contiene los identificadores de los procesos activos en el sistema. Se reconoce a un proceso como activo a la recepción de un mensaje $BEGIN$ y deja de ser activo a la recepción de un mensaje END.

4.4.2. Especificación del mecanismo

En este apartado se presentan los algoritmos que conforman el mecanismo MLJ, el cual se compone por seis algoritmos. Los mecanismo aquí presentados tienen como función principal la inicialización de las estructuras de datos y el envío/recepción de MMS continuos/discretos. Las funciones para el envío de MMS hacen uso del algoritmo para la selección de coordinadores/maestros explicado en la sección 4.2.

El algoritmo 2 se encarga de la inicialización de las variables y las estructuras de datos a usar durante la sincronización. También, el algoritmo se encarga de definir el tipo de MMS que será transmitido durante la comunicación, por el proceso en cuestión.

Algoritmo 2 Inicialización

```
1: procedure INITIALLY
2:     ♯define {continuous|discrete}                ▷ Configuración del tipo de dato a intercambiar
3:     ♯if define (continuous)                      ▷ para utilizar las funciones adecuadas.
4:         send_continuous(input: TP = {begin|end|cut|fifo_p})   ▷ Declaración de los tipos de mensajes.
5:     ♯elif define (discrete)
6:         send_discrete(input: TP = discrete)
7:     ♯endif
8:     VT(p)[j] = 0   ∀j : 1...n, CI(p) ← ∅, last_fifo(p) ← ∅, Act = 0, Master = 0, Slaves ← ∅, ActP ← ∅
9: end procedure
```

Como se ha mencionado anteriormente, nuestro mecanismo brinda el servicio de sincronización entre MMS discretos (texto, imágenes fijas, etc.) y MMS continuos (audio, video). Para lograr la heterogeneidad de MMS nuestro mecanismo usa dos funciones para el envío de datos: el algoritmo 3 para el envío de MMS discretos y el algoritmo 4 para el envío de MMS continuos.

La segmentación y sincronización de los MMS de nuestro mecanismose alcanza por medio del algoritmo 5 que es el encargado de la recepción y procesamientos de los mensajes. En este algoritmo se ejecuta la identificación de la información causal, la cual servirá para segmentar y sincronizar los MMS en el maestro y en los esclavos.

Algoritmo 3 Envío de MMS discretos

10: **procedure** SendDiscrete(input: $TP = discrete$)
11: $VT(p)[i] = VT(p)[i] + 1$
12: **for all** $(s, r) \in CI(p)$ **do** ▷ Actualización del conjunto $CI(p)$.
13: **if** $\exists(x, f) \in last\_fifo(p) | s = x$ **then** ▷ Se añaden los paquetes $fifo\_p$ a $CI(p)$.
14: **if** Not$((s, r) = max((x, f), (s, r)))$ **then**
15: $CI(p) \leftarrow (CI(p) \backslash (s, r)) \cup (x, f)$ ▷ Se reemplaza (s, r) de $CI(p)$ por (x, f).
16: **end if**
17: **end if**
18: **end for**
19: **if** $Master = 0$ **then** ▷ Se verifica si ya se tiene seleccionado a un proceso como maestro, de lo contrario
20: $Master =$getMasterID(0) ▷ se selecciona un maestro invocando a la función getMasterID
21: **end if**
22: **if** $\exists(s, t) \in CI(p) | s = Master$ **then** ▷ Si existe información causal del maestro
23: $H(m) \leftarrow \{(s, r)\}$ ▷ ésta es ligada a $H(m)$.
24: **else**
25: $H(m) \leftarrow \emptyset$
26: **for all** $(s, t) \in CI(p)$ **do** ▷ Se liga la información de los procesos pertenecientes
27: **if** isNeighborAtLevel(s,0) **then** ▷ al mismo subgrupo de nivel 0.
28: $H(m) \leftarrow H(m) \cup (s, t)$
29: **end if**
30: **end for**
31: **end if**
32: $last\_fifo(p) \leftarrow \emptyset$
33: $CI(p) \leftarrow \emptyset$ ▷ Se limpia $CI(p)$ para este mensaje causal
34: $m = (i, t = VT(p)[i], Master, TP, H(m), data)$
35: $sending(m)$
36: **end procedure**

Algoritmo 4 Envío de MMS continuos

```
37: procedure SendContinuous(input: TP = {begin|end|cut|fifo_p})
38:     VT(p)[i] = VT(p)[i] + 1
39:     if Not(TP = fifo_p) then
40:         if (TP = end) then
41:             Act = 0                          ▷ Determina que el participante p está inactivo.
42:         else if TP = begin then
43:             Act = 1                          ▷ Determina que el participante p está activo.
44:             for all (s, r) ∈ CI(p) do        ▷ Actualización del conjunto CI(p).
45:                 if ∃(x, f) ∈ last_fifo(p)|s = x then   ▷ Se añaden los paquetes fifo_p a CI(p).
46:                     if Not((s, r) = max((x, f), (s, r))) then
47:                         CI(p) ← (CI(p)\(s, r)) ∪ (x, f)   ▷ Se reemplaza (s, r) de CI(p) por (x, f).
48:                     end if
49:                 end if
50:             end for
51:             last_fifo(p) ← ∅
52:             Slaves ← ∅
53:             level = 0
54:             while level < |NS| and Master = 0 do   ▷ Se busca un proceso maestro dentro de todos
55:                 Master =getMasterID(level)         ▷ los niveles a los que pertenece el participante p.
56:                 level = level + 1
57:             end while
58:         end if
59:         H(m) ← ∅
60:         if Not(Master = 0) or Not(Slaves = ∅) then   ▷ Construcción de H(m).
61:             for all (s, r) ∈ CI(p) do
62:                 if s = Master or ∃x ∈ Slaves|x = s then   ▷ Se liga la información causal del maestro
63:                     H(m) ← H(m) ∪ (s, r)   ▷ o de los esclavos según sea el rol asignado al participante p.
64:                 end if
65:             end for
66:             Slaves ← ∅
67:         else
68:             for all (s, t) ∈ CI(p) do        ▷ Se liga la información causal de los procesos
69:                 if isNeighborAtLevel(s,0) then   ▷ que pertenecen al subgrupo de nivel 0.
70:                     H(m) ← H(m) ∪ (s, t)
71:                 end if
72:             end for
73:         end if
74:         CI(p) ← ∅                            ▷ Se limpia CI(p) para este mensaje causal
75:     else
76:         H(m) ← ∅
77:     end if
78:     m = (i, t = VT(p)[i], Master, TP, H(m), data)
79:     sending(m)
80:     if TP = end then
81:         Master = 0                           ▷ Al pasar a inactivo, se indica que no hay maestro.
82:     end if
83: end procedure
```

Algoritmo 5 Proceso de recepción

```
84: procedure RECEIVE(i ≠ j, m = (i, t, MS, TP, H(m), data))
85:     if (t = VT(p)[i] + 1) then                          ▷ Condición para la entrega de mensajes FIFO.
86:         if Not(TP = fifo_p) then
87:             if Not(t' ≤ VT(p)[l]∀(l, t') ∈ H(m)) then   ▷ Condición para la entrega de mensajes causales.
88:                 wait()
89:             else                                        ▷ Procedimiento para la entrega causal
90:                 delivery(m)
91:                 VT(p)[i] = VT(p)[i] + 1
92:                 if ∃(s, r) ∈ CI(p)|i = s then
93:                     CI(p) ← CI(p)\(s, r)                ▷ Se elimina (s, r) de CI(p).
94:                 end if
95:                 CI(p) ← CI(p) ∪ (i, t)                  ▷ Se agrega (i, t) a CI(p)
96:                 for all (l, t') ∈ H(m) do               ▷ Actualización de CI(p) y de last_fifo(p)
97:                     if ∃(s, r) ∈ CI(p)|l = s ∧ r ≤ t' and Not(l = MS ∧ isNeighbor(l)) then
98:                         CI(p) ← CI(p)\(s, r)
99:                     end if
100:                    if ∃(x, f) ∈ last_fifo(p)|l = x ∧ f ≤ t' and Not(l = MS ∧ isNeighbor(l)) then
101:                        last_fifo(p) ← last_fifo(p)\(x, f)
102:                    end if
103:                end for
104:                if TP = begin then
105:                    ActP ← ActP ∪ {i}                   ▷ Identificación de los procesos activos.
106:                    if Act = 1 then
107:                        if MS = 0 and Master = 0 and i < j and isNeighbor(i) then
108:                            Master = i
109:                        else if Master = MS and isNeighbor(i) then
110:                            if Not(isNeighborAtLevel(MS,getLevel(i))) and i < j then
111:                                Master = i
112:                            end if
113:                        else if MS = j then
114:                            Slaves ← Slaves ∪ {i}       ▷ Identificación de los procesos eslavo.
115:                        end if
116:                    end if
117:                else if Act = 1 and Not(TP = cut) and (MS = j ∨ i = Master) then
118:                    ActP ← ActP\{i}                     ▷ El proceso i se elimina de los procesos activos.
119:                    if TP = discrete then
120:                        Slaves ← Slaves ∪ {i}
121:                    else if i = Master then
122:                        Slaves ← Slaves ∪ {i}
123:                        Master = 0
124:                    end if
125:                    SendContinuous(cut)                 ▷ Envío de mensaje cut.
126:                else if TP = discrete and Act = 1 and MS = 0 and isNeighbor(i) then
127:                    if Master = 0 or Not(isNeighborAtLevel(Master,0)) then
128:                        Slaves ← Slaves ∪ {i}
129:                        SendContinuous(cut)             ▷ Envío de mensaje cut.
130:                    end if
```

Algoritmo 6 Proceso de recepción (cont.)

```
131:            else if TP = cut and MS = j and Not(∃x ∈ Slaves|x = i) then
132:                Slaves ← Slaves ∪ {i}
133:            else if TP = end then
134:                ActP ← ActP\{i}
135:                if i = Master then
136:                    Master = 0
137:                end if
138:            end if
139:        end if
140:    else                                          ▷ Procedimiento para la entrega FIFO
141:        delivery(m)
142:        VT(p)[i] = VT(p)[i] + 1
143:        if ∃(x, f) ∈ last_fifo(p)|i = x then
144:            last_fifo(p) ← last_fifo(p)\(x, f)
145:        end if
146:        last_fifo(p) ← last_fifo(p) ∪ (i, t)      ▷ Actualización de last_fifp(p) con paquetes más
    recientes.
147:        if MS = j and ∄x ∈ Slaves|x = i then
148:            Slaves ← Slaves ∪ {i}
149:        end if
150:    end if
151:  else
152:    wait_FIFO()
153:  end if
154: end procedure
```

4.4.2.1. Descripción general del mecanismo jerárquico

En esta sección se da una explicación general del funcionamiento de los algoritmos desarrollados. Para explicar los pasos a seguir usaremos las reglas presentadas en las tablas 3.1 y 3.2 ya que nos basamos en el trabajo propuesto en [PMER08].

Sincronización de intervalos con respecto al maestro. Como se ha mencionado anteriormente, para que un proceso sincronice el MMS que se encuentra transmitiendo éste debe utilizar las referencias temporales originadas por un flujo maestro. Para que un proceso identifique cual será su flujo maestro primero deberá conocer que procesos se encuentran activos. Cuando un proceso comienza a trans-

mitir un flujo continuo éste siempre envía un mensaje *begin* (algoritmo 4, líneas 42-58). El mensaje *begin* nos sirve para la identificación del proceso maestro, ya que a la recepción de éste es que se puede identificar cuando un proceso está activo (algoritmo 5, líneas 104-116). Cuando un proceso deja de transmitir un flujo continuo envía un mensaje *end* (algoritmo 4, líneas 40-41) y es a la recepción de éste cuando cambia su estado a inactivo dentro de los demás procesos (algoritmo 5, líneas 117-125).

Para explicar como se sincronizan los flujos de procesos esclavos con el flujo transmitido por un maestro usaremos el escenario de la figura 4.10. En la figura se presentan cuatro procesos transmitiendo flujos continuos y discretos. Para este ejemplo suponemos que los procesos pertenecen a un mismo subgrupo. Como se muestra en la figura 4.10 el proceso X es el primero en trasmitir un flujo continuo mediante el envío del mensaje x_1 (*begin*). Antes del envío del mensaje x_1, el proceso X busca dentro de los procesos activos el identificador de su proceso maestro (algoritmo 4, líneas 54-57). Al no existir en ese instante un proceso activo (algoritmo 1, línea 2), el proceso X pasa a ser el maestro de ese subgrupo. Cuando el mensaje x_1 es recibido por los demás procesos, el identificador del proceso X es agregado a la lista de procesos activos $ActP$. Para el caso de la recepción en X, al ser maestro de Z agrega el identificador de éste último dentro de su lista de procesos esclavos (algoritmo 5, líneas 119-124). Lo anterior se realizará siempre y cuando el identificador MS dentro de todo mensaje m sea igual al identificador del proceso receptor.

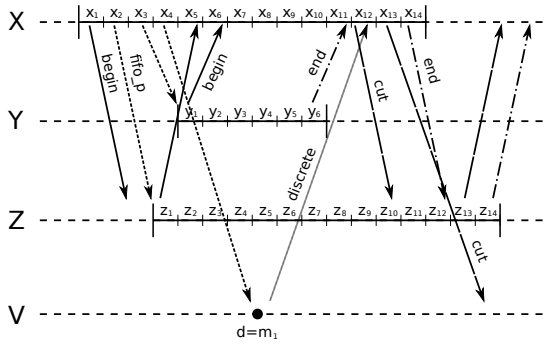


Figura 4.10: Escenario de ejemplo de un subgrupo simple.

Inmediatamente después de que el proceso Z recibe el mensaje x_2 (previamente enviado por X), éste envía el mensaje z_1 ($begin$) con el historial causal $H(m) = \{x_2\}$. Antes de enviar el mensaje x_2, el proceso busca dentro de los procesos activos a su proceso maestro. Al ser X el primer proceso en la lista (algoritmo 1, líneas 3-6), el proceso Z lo toma como su proceso maestro. Después de identificado el proceso maestro, el proceso Z agrega al historial causal la referencia temporal correspondiente a su maestro (algoritmo 4, líneas 60-73). Cuando el mensaje z_1 es recibido por los demás procesos, el identificador del proceso Z se agrega a la lista de procesos activos. En este caso la lista de procesos activos dentro de Y y V contiene los identificadores de X seguido del identificador de Z; $ActP = \{X, Z\}$.

Cuando el proceso Y envía el mensaje y_1, también envía en su historial causal la referencia temporal con respecto a X ($H(m) = \{x_3\}$). Debido a que el identificador del proceso X es el primero en la lista $ActP$ del proceso Y, éste es seleccionado como maestro. Como el proceso X es el maestro del proceso Y, Y sólo envía la información causal con respecto a su maestro y no con respecto a los demás procesos presentes.

Después de cierto tiempo, el proceso Y finaliza la transmisión de su flujo continuo. Para indicar el fin de un flujo continuo un proceso siempre envía el mensaje *end*. Cuando Y se dispone a enviar y_6, éste solo agrega la información causal con

respecto al último mensaje de X recibido hasta ese instante. Un mensaje del tipo *end* sólo será importante para el proceso cuyo identificador sea igual al indicado por el campo MS dentro del mensaje m. Cuando el proceso X recibe el mensaje y_6 (recordar que X es el maestro de Y), éste envía casi inmediatamente el mensaje x_{12} (*cut*) (algoritmo 5, líneas 117-125). Antes del envío del mensaje *cut*, X agrega la información causal con respecto a todos sus esclavos indicados por $Slaves$. Por último, cabe señalar que a la recepción de un mensaje *end* se elimina el identificador del proceso emisor de $ActP$, esto dentro de cada procesos receptor (algoritmo 5, líneas 133-138).

Envío y recepción de MMS discretos. En la figura 4.10 se muestra al proceso V transmitiendo un flujo discreto (d). El mensaje d es enviado inmediatamente después de la recepción del mensaje x_4. Antes del envío del mensaje d, el proceso V busca a su proceso maestro y posteriormente, agrega la información causal del maestro seleccionado. (algoritmo 3, líneas 19-31). En el caso del mensaje d, el proceso enviará la información causal con respecto al último mensaje recibido del proceso X ($H(m) = \{x_4\}$). Inmediatamente después de que el proceso X recibe el mensaje d, éste envía el mensaje x_{12} (*cut*) (algoritmo 5, líneas 126-130). Antes de que el proceso X envíe el mensaje x_{12}, se agrega la información causal de todos los esclavos indicados por $Slaves$.

En el anexo A se muestra una prueba de funcionamiento realizada a nuestro algoritmo, tomando como ejemplo un sistema compuesto por doce procesos organizados mediante una arquitectura a tres niveles. Por medio de esta prueba se muestra el comportamiento de las estructuras y los mensajes que son enviados entre los procesos.

4.4.2.2. Prueba de correctez (*Correctness*)

Para mostrar que el mecanismo de sincronización desarrollado asegura la entrega ordena de mensajes, en el siguiente apartado se presenta una prueba de correctez. La prueba se centra en mostrar que mediante el uso de esquema maestro esclavo y mediante el envío sólo de la información del maestro, la entrega ordenada de mensajes es respetada.

Teorema 1 *Sean $X, Y \in I$ dos MMS generados por $p_i, p_j \in P$, $i \neq j$, respectivamente. Si $\exists x \in X, y \in Y$ tal que $send(x) \rightarrow send(y)$ entonces $delivery(k, x) \rightarrow delivery(k, y)$ para cualquier $p_k \in P$.*

Se hace notar que como consecuencia de la arquitectura jerarquica establecida y de acuerdo a la información causal que un proceso envía en $H(m)$, si $X \rightarrow Y$ implica que p_i es forzosamente un proceso maestro. Considerando esto para demostrar el teorema 1 se identifican dos casos que se exponen a continuación. El primer caso es cuando X precede inmediatamente a Y. El segundo caso es cuando X no precede inmediatamente a Y, lo que significa que existe un maestro intermedio p_l que genera un flujo $V \in I$ tal que $X \rightarrow V \rightarrow Y$.

- **Caso 1:** Sea p_i el proceso maestro directo del proceso p_j. Si $\exists x_\alpha \in X, y^- \in Y$ tal que $send(x_\alpha) \rightarrow send(y^-)$ entonces $x_\alpha \in H(y^-)$, donde x_α es el último mensaje fifo o causal recibido de X por p_j antes de la emisión de y^-.

- **Caso 2:** Sea p_i el proceso maestro de p_l y p_l es el maestro directo de p_j. Si $\exists x_\alpha \in X, v^-, v_\beta \in V, y^- \in Y$ tal que $send(x_\alpha) \rightarrow send(y^-)$ y $send(x_\beta) \rightarrow send(y^-)$, entonces $x_\alpha \in H(v^-)$ y $v_\beta \in H(y^-)$, donde x_α y v_β son los últimos mensajes fifo o causal recibidos por p_l y p_j, respectivamente y antes de la emisión de v^- y y^-.

Prueba del caso 1. Antes de demostrar el caso 1, se debe tomar en cuenta lo siguiente. Todo intervalo $W \in I$ enviado por un proceso $p_r \in P$ está compuesto

por el envío secuencial de mensajes $W = \{w_\gamma \rightarrow_i w_{\gamma+1} \rightarrow_i \ldots \rightarrow_i w_{\gamma+k}\}$. Denotamos como w^- y w^+ los extremos del intervalo W ($w^- = w_\gamma$ y $w^+ = w_{\gamma+h}$) y cualquier $w \neq m_\gamma \wedge w \neq m_{\gamma+h}$ es un mensaje de tipo $fifo\_p$.

La prueba del caso 1 se realizará mediante una prueba directa. Comenzamos demostrando que por la propiedad FIFO (línea 85, algoritmo 5) para todo mensaje $x_f \in X$ tal que $f < \alpha$ se asegura $delivery(k, x_f) \rightarrow delivery(k, x_\alpha)$. Lo anterior se asegura por la condición de entrega FIFO y la actualización de los vectores (líneas 85 y 91 del algoritmo 5).

Posteriormente, apartir de que se asume que X es el maestro directo de Y tenemos que si $send(x_\alpha) \rightarrow send(y^-)$ se asegura por las líneas 22-24 del algoritmo 3 y las líneas 61-65 del algoritmo 4 que $x_\alpha \in H(y^-)$. Por la condición de entrega causal (línea 87 del algoritmo 5) se asegura entonces que $delivery(k, x_\alpha) \rightarrow delivery(k, y^-)$. Ésto se cumple ya que y^- será entregado sólo si x_α ha sido previamente entregado.

Por otra parte, tenemos que por propiedad FIFO para $y_\gamma = y^-$ y para todo y_f tal que $\gamma < f$ se asegura $delivery(k, y_\gamma) \rightarrow delivery(k, y_f)$. Sea $A \subseteq X$ donde $A = \{x_\gamma \rightarrow_i \ldots \rightarrow_i x_\alpha\}$, por la propiedad 1 se tiene por lo tanto que $A \rightarrow Y$.

Prueba del caso 2. Por el caso 1, se asume que X es maestro inmediato de V y V es maestro inmediato de Y. Por lo anterior se tiene que si $send(x_\alpha) \rightarrow send(v^-)$, $send(v_\beta) \rightarrow send(y^-)$ entonces $x_\alpha \in H(v^-)$ y $v_\beta \in H(y^-)$, por lo que se asegura que $delivery(k, x_\alpha) \rightarrow delivery(k, v^-)$ y $delivery(k, v_\beta) \rightarrow delivery(k, y^-)$. Nuevamente, por la propiedad FIFO se asegura que para $v_\sigma = v^-$ y para cualquier v_g, tal que $\sigma < g$, $delivery(k, v_\sigma) \rightarrow delivery(k, v_g)$. Dado que v_β denota a un mensaje FIFO se tiene que $send(v_\sigma) \rightarrow send(v_\beta)$. Como $send(x_\alpha) \rightarrow send(v^-)$ y $send(v_\beta) \rightarrow send(y^-)$, por lo demostrado en el caso 1 y por la propiedad transitiva de la *happened-before relation* se asegura que $delivery(k, x_\alpha) \rightarrow delivery(k, y^-)$.

4.4.3. Corrección del error en la sincronización

Dado que el Internet está siendo desarrollado sobre un medio no fiable, los mecanismos que traten el problema de la sincronización multimedia deben de soportar el retraso y la pérdida de paquetes. Para nuestro caso sólo se ha dado soporte al retraso de paquetes, dejando la corrección a la pérdida de paquetes para trabajos futuros.

El soporte al retraso de mensajes en nuestro trabajo es dado a través de dos acciones: la primera es retrasar la entrega de los mensajes que no satisfagan las dependencias causales, y la segunda, descartar los mensajes después de que se haya finalzado un cierto tiempo de espera Δ_{il} [HX97].

El tiempo que un mensaje m debe esperar para ser entregado, se determina por medio de la función 4.2 para la espera de mensajes. En esta función se indica que un mensaje m va a ser entregado si y sólo si cumple con las siguientes condiciones: si para todo mensaje m' con identificador l y tiempo t' en el historial causal $H(m)$, a) han arribado todos los mensajes que hagan cumplir con las dependencias causales indicadas por $H(m)$, o b) se ha agotado el tiempo de espera para ese mensaje.

$$\begin{aligned} wait(m) :& i = Part(m) \\ & \{\forall m' = (l, t') \in H(m), \\ & \text{a) } t' \leq VT(p)[l] \text{ o} \\ & \text{b) } receive\_time_p(m) + remainder\_time_{m'}(i, l) \leq current\_time(p)\} \end{aligned} \tag{4.2}$$

El tiempo que un mensaje deberá de esperar se calcula mediante la suma del tiempo físico en el cual fué recibido y el tiempo *restante* de espera definido por la función $remainder\_time_{m'}(i, l)$. La función $remainder\_time_{m'}(i, l)$ (4.3) calcula el tiempo restante haciendo uso del máximo error permitido Δ_{il}[2] menos la dife-

[2]El máximo error permitido Δ_{il} indica el máximo error de sincronización tolerado entre el tipo de MMS.

rencia de tiempo que existe entre el último mensaje recibido de un proceso con identificador i y el último mensaje de recibido de un proceso l.

$$remainder\_time_{m'}(i,l) = \begin{cases} \Delta_{il} - (last\_rcv_p(i) - last\_rcv_p(l)) & \text{Si } > 0 \\ 0 & \text{En otro caso} \end{cases} \tag{4.3}$$

Un ejemplo del funcionamiento de la función de espera es dado a partir del escenario que se muestra en la figura 4.11. En la figura se observa que después de que el proceso Y recibe el mensaje a^- enviado por X, éste inmediatamente envía el mensaje d.

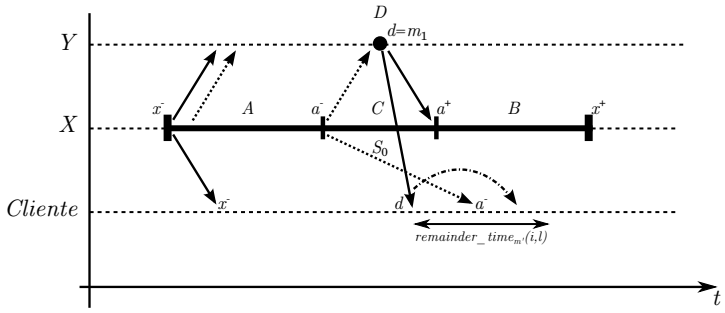


Figura 4.11: Ejemplificación del retraso de mensajes.

Dada las condiciones de transmisión de la red se puede dar el caso que el mensaje d sea entregado antes del arribo del mensaje a^-, pudiendo ocasionar una inconsistencia en la sincronización y reproducción de los datos. Mediante el uso de nuestra función de espera el mensaje d, antes de que éste sea entregado a la aplicación, deberá esperar un determinado tiempo. Cuando el mensaje a^- sea recibido en tiempo, se procederá a realizar la entrega del mensaje d. En caso de que el tiempo de espera haya caducado y el mensaje a^- no haya sido recibido, se entregará el mensaje d y se descartará el mensaje a^- y todos aquellos mensajes que pudiesen precederlo.

4.4.3.1. Función para la entrega de mensajes en espera

Nuestro esquema M/S pone ciertas restricciones para la recepción y entrega de mensaje, que se deben tomar en cuenta para la entrega de mensajes que estén en espera. A continuación se presenta una función para la actualización y entrega de mensajes en forma correcta; ver algoritmo 7.

El primer mensaje a enviar durante la transmisión de un flujo continuo es del tipo *BEGIN*, si este mensaje es descartado al vencer su tiempo de espera el proceso receptor no agregaría al proceso emisor al conjunto de procesos activos. Para corregir este problema se propone que al no arribar un mensaje *BEGIN* durante el tiempo de espera, el primer mensaje *FIFO_P* del flujo correspondiente sea tomado como un mensaje *BEGIN*. Mediante esta acción se evitan errores y pérdidas de información necesarias para realizar la sincronización.

Algoritmo 7 Proceso de actualización de $CI(p)$ y $last\_fifo(p)$

```
1: procedure UPDATE_CI_LAST_FIFO(m = (i, t, MS, TP, H(m), data))
2:     if Not(TP = fifo_p) then
3:         delivery(m)
4:         if ∃(s, r) ∈ CI(p)|i = s then
5:             CI(p) ← CI(p)\(s, r)
6:         end if
7:         CI(p) ← CI(p) ∪ (i, t)
8:         for all (l, t') ∈ H(m) do
9:             if ∃(s, r) ∈ CI(p)|l = s ∧ r ≤ t' and Not(l = MS ∧ isNeighbor(l)) then
10:                CI(p) ← CI(p)\(s, r)
11:            end if
12:            if ∃(x, f) ∈ last_fifo(p)|l = x ∧ f ≤ t' and Not(l = MS ∧ isNeighbor(l)) then
13:                last_fifo(p) ← last_fifo(p)\(x, f)
14:            end if
15:        end for
16:        if TP = begin then
17:            ActP ← ActP ∪ {i}
18:            if Act = 1 then
19:                if MS = 0 and Master = 0 and i < j and isNeighbor(i) then
20:                    Master = i
21:                else if Master = MS and isNeighbor(i) then
22:                    if Not(isNeighborAtLevel(MS,getLevel(i))) and i < j then
23:                        Master = i
24:                    end if
25:                else if MS = j then
26:                    Slaves ← Slaves ∪ {i}
27:                end if
28:            end if
```

Algoritmo 8 Proceso de actualización de $CI(p)$ y $last\_fifo(p)$ (cont.)

```
29:         else if Act = 1 and Not(TP = cut) and (MS = j ∨ i = Master) then
30:             ActP ← ActP\{i}
31:             if TP = discrete then
32:                 Slaves ← Slaves ∪ {i}
33:             else if i = Master then
34:                 Slaves ← Slaves ∪ {i}
35:                 Master = 0
36:             end if
37:             SendContinuous(cut)
38:         else if TP = discrete and Act = 1 and MS = 0 and isNeighbor(i) then
39:             if Master = 0 or Not(isNeighborAtLevel(Master,0)) then
40:                 Slaves ← Slaves ∪ {i}
41:                 SendContinuous(cut)
42:             end if
43:         else if TP = cut and MS = j and Not(∃x ∈ Slaves|x = i) then
44:             Slaves ← Slaves ∪ {i}
45:         else if TP = end then
46:             ActP ← ActP\{i}
47:             if i = Master then
48:                 Master = 0
49:             end if
50:         end if
51:     else
52:         delivery(m)
53:         if ∃(x, f) ∈ last_fifo(p)|i = x then
54:             last_fifo(p) ← last_fifo(p)\(x, f)
55:         end if
56:         last_fifo(p) ← last_fifo(p) ∪ (i, t)
57:         if ∄x ∈ ActP|x = i then
58:             ActP ← ActP ∪ {i}
59:             if Act = 1 then
60:                 if MS = 0 and Master = 0 and i < j and isNeighbor(i) then
61:                     Master = i
62:                 else if Master = MS and isNeighbor(i) then
63:                     if Not(isNeighborAtLevel(MS,getLevel(i))) and i < j then
64:                         Master = i
65:                     end if
66:                 else if MS = j then
67:                     Slaves ← Slaves ∪ {i}
68:                 end if
69:             end if
70:         else if MS = j and ∄x ∈ Slaves|x = i then
71:             Slaves ← Slaves ∪ {i}
72:         end if
73:     end if
74: end procedure
```

Capítulo 5

Simulación y resultados

En este capítulo se presentan los resultados obtenidos mediante las simulaciones hechas a nuestro mecanismo jerárquico. Las simulaciones fueron realizadas mediante el uso del simulador para algoritmos de red *Sinalgo* [sin13], nos provee de herramientas para simular diversas condiciones de red como el retraso y pérdida de paquetes. Nuestro mecanismo se evaluó sólo tomando en cuenta el restraso de mensajes.

El objetivo de las simulaciones es comprobar el funcionamiento de nuestro mecanismo jerárquico así como el comportamiento de las funciones de corrección ante retardos de mensajes. Las simulaciones se ejectuan utilizando un escenario compuesto de veinte procesos ordenados jeráquicamente en subgrupos de procesos, mediante el cual se representa un sistema distribuido asíncrono sin importar la configuración y topología de la red.

El capítulo está organizado de la siguiente manera, en la primera sección se describe el escenario de prueba utilizado para realizar las simulaciones de nuestro mecanismo, en la segunda sección se muestra las funciones para calcular el error de sincronización. En la última sección se muestran los resultados obtenidos mediante las simulaciones y se compara los resultados con los resultados obtenidos de la simulación del algoritmo de ML.

5.1. Descripción del escenario de prueba

Como ya se ha mencionado con anterioridad nuestro escenario representa un sistema distribuido asíncrono, en el cual no están definidos los tiempos de envío/recepción de paquetes, no hay memoria compartida y no existen referencias globales. También, suponemos que los procesos se comunican a través del envío de mensajes.

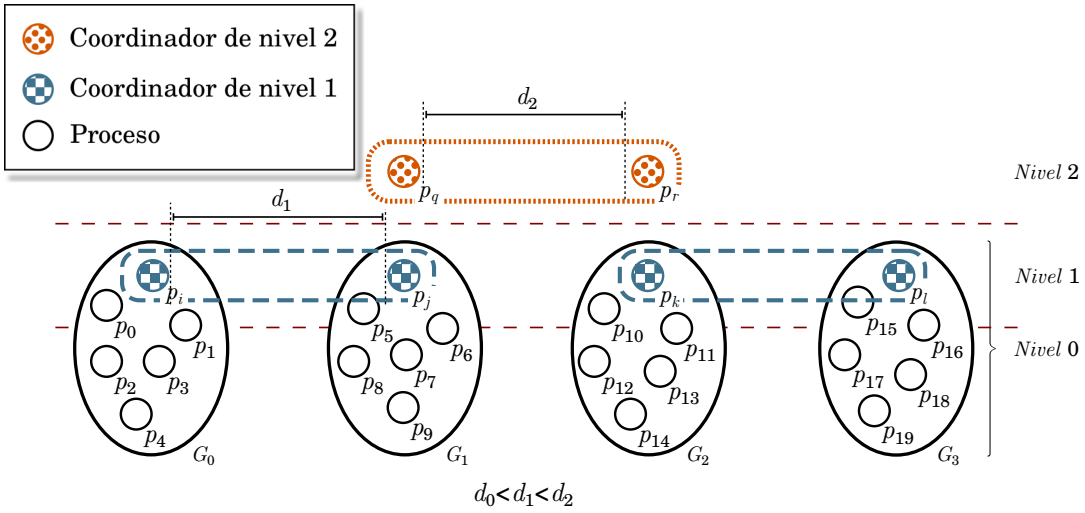


Figura 5.1: Arquitectura a tres niveles con veinte procesos.

Para las simulaciones consideramos que el sistema es utilizado para educación a distancia, en el cual diferentes procesos se encuentran transmitiendo MMS de audio, MMS de video y otros transmiten filminas. En el escenario de prueba suponemos que el sistema en cuestión está compuesto por veinte procesos, organizados previamente mediante subgrupos de cinco procesos cada uno usando nuestro esquema organizacional propuesto anteriormente; ver figura 5.1. Se eligió que el sistema esté compuesto por veinte procesos debido que, el protocolo estándar para la sincronización de MMS, RTP sólo soporta hasta dieciséis procesos en una sola sesión [Per03]. Cada proceso genera de manera aleatoria el número de intervalos y el número de mensajes a enviar, de igual manera se genera de forma aleatoria

el tiempo de inicio de transmisión de cada intervalo.

En nuestro escenario no se contempla la configuración ni la topología de la red, por lo que algunos procesos podrían llegar a pertenecer a una misma máquina, estar dentro de redes PAN, LAN o WAN.

5.2. Cálculo del error de sincronización

Los mecanismos encargados de la sincronización multimedia deben de cumplir con las restricciones de tiempo para cada tipo de dato para el correcto funcionamiento de las aplicaciones en tiempo real. La Unión Internacional de Telecomunicaciones (International Telecommunication Union (ITU)) propone, para aplicaciones de videoconferencia en tiempo real, que el error máximo de sincronización sea de 80 ms para realizar Lip synchronization (Lip-sync)[1] [itu01].

Nuestro mecanismo se evalúa por medio de las funciones $rcv\_synch\_error_p(m)$ y $dlv\_synch\_error_p(m)$ [PMER08, PEMR09]. La función $rcv\_synch\_error_p(m)$ (5.1) calcula el error de sincronización a la hora de la recepción de un mensaje sin la aplicación de nuestra función de corrección a los retardos. La función $dlv\_synch\_error_p(m)$ (5.2) calcula el error de sincronización a la hora de entregar el mensaje después de haber aplicado nuestra función de corrección.

La función $rcv\_synch\_error_p(m)$ obtiene el error de sincronización mediante la sumatoria de la sustracción del tiempo en que se recibió el mensaje m y el tiempo en que se recibieron los últimos mensajes pertenecientes a su historial causal. El resultado de la sumatoria se promedia respecto al número de duplas contenidas en $H(m)$.

[1] Es la sincronización del movimiento de la boca y la lengua durante el habla en la reproducción de audio y video [MRBF97].

$$rcv\_synch\_error_p(m) = \frac{\sum_{(l,t')\in H(m)} receive\_time_p(m) - last\_rcv_p(l)}{|H(m)|} \quad (5.1)$$

La función $dlv\_synch\_error_p(m)$ obtiene el error de sincronización mediante la sumatoria de la sustracción del tiempo en que es entregado el mensaje m y el tiempo en que fueron entregados los últimos mensajes pertenecientes a su historial causal. Como en el caso anterior, el resultado de la sumatoria se promedia respecto al número de duplas contenidas en $H(m)$.

$$dlv\_synch\_error_p(m) = \frac{\sum_{(l,t')\in H(m)} delivery\_time_p(m) - last\_dlv_p(l)}{|H(m)|} \quad (5.2)$$

5.3. Resultados

El mecanismo desarrollado se evalúo ejecutando diversas pruebas en el simulador. Los retardos en la red se simularon a través de diferentes rangos de tiempo. Los rangos de tiempo utilizados son los propuestos por Hać y Xue [HX97], Pomares et al. y los retardos para videoconferencia usando lipsyc recomendados por el ITU (ver tabla 5.1).

Las simulaciones efectuaron con el simulador de red Sinalgo, versión 0.75.3. Se emplearon dos equipos Dell™ Workstation Precision™ 7500 cuyas características son las siguientes:

- Procesador Intel® Xeon® E5620 a 2.40 Ghz
- Memoria total de 8 Gb
- Microsoft® Windows® 7 Professional de 64 bits con Service Pack 1
- Java™ versión 1.70_21 para 64 bits.

El objetivo principal de las simulaciones es comprobar el correcto funcionamiento de las funciones de corrección ante el retardo de paquetes. El funcionamiento de las funciones de corrección se comprobó comparando y analizando los errores de sincronización obtenidos mediante las funciones $rcv\_synch\_error_p(m)$ y $dlv\_synch\_error_p(m)$.

Con el fin de comparar el mecanismo aquí presentado, se realizaron simulaciones del mecanismo de ML. Ambos mecanismos fueron comparados por medio del uso de las mismas funciones para el cálculo del error de sincronización y bajo las mismas condiciones de red. Se utilizaron funciones pseudoaleatorias para una correcta comparación de ambos mecanismos. Las funciones pseudoaleatorias fueron inicializadas con las mismas semillas para la generar intervalos del mismo tamaño. Los resultados de estas evaluaciones se presentan en las siguientes subsecciones.

Tabla 5.1: Retardos de transmisión establecidos para la simulación.

Tipo de dato	Prueba 1 (ms)	Prueba 2 (ms)	Prueba 3 (ms)
Audio/Video	300 ± 120	300 ± 180	400 ± 150
Imagen/Texto/Etc	200 ± 50	200 ± 100	–
Máximo error permitido	240	240	80

5.3.1. Resultados de la prueba 1

Para esta prueba los canales de comunicación se configuraron con retrasos en un rango de 50 a 300 ms. El tiempo máximo de espera (Δ_{ij}) para ambos mecanismos se fijó en 240 ms; ver tabla 5.1. Los resultados son descritos primero presentando los errores de sincronización obtenidos mediante la ejecución de nuestro mecanismo MLJ, seguido de los resultados obtenidos del mecanismo ML. Por último, se presenta una comparativa de ambos mecanismo, tomando los mismos

puntos de sincronización y sólo mediante el uso del error de sincronización a la entrega.

En la figura 5.2 se presenta la gráfica del error de sincronización a la recepción (línea punteada negra) y el error de sincronización a la entrega (línea punteada roja) del mecanismo MLJ. Con ayuda de las simulaciones obtenemos que el promedio del error de sincronización a la recepción es de 62.55 ms, mientras que el promedio del error aplicando el mecanismo de corrección es 18.76 ms. También, se obtiene que el 86.61 % de los mensajes, aplicando el mecanismo de corrección, están por debajo del error de sincronización a la recepción y del máximo error permitido.

La figura 5.3 presenta la gráfica de los errores de sincronización a la recepción y a la entrega del mecanismo ML. A través de las simulaciones obtenemos que el promedio del error de sincronización a la recepción es de 59.36 ms, mientras que el promedio del error aplicando el mecanismo de corrección es 81.95 ms. También, se obtiene que el 27.09 % de los mensajes, aplicando el mecanismo de corrección, están por debajo del error de sincronización a la recepción y del máximo error permitido.

En la figura 5.4 se muestra la comparativa realizada a ambos mecanismos por medio de los mismos puntos de sincronización. Mediante las simulaciones obtenemos que el promedio del error de sincronización del mecanismo ML es de 66.75 ms, mientras que el promedio del mecanismo MLJ es de 13.97 ms. También, se obtiene que el 83.40 % de los mensajes del mecanismo MLJ están por debajo del error de sincronización del mecanismo ML y del máximo error permitido.

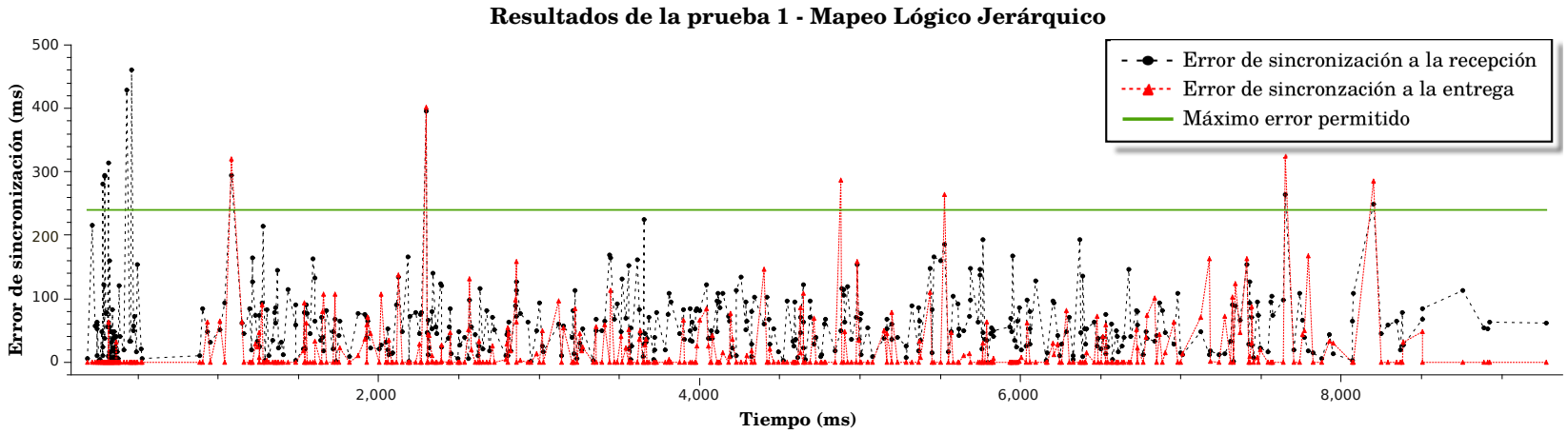


Figura 5.2: Resultados de la prueba 1 con el mecanismo MLJ.

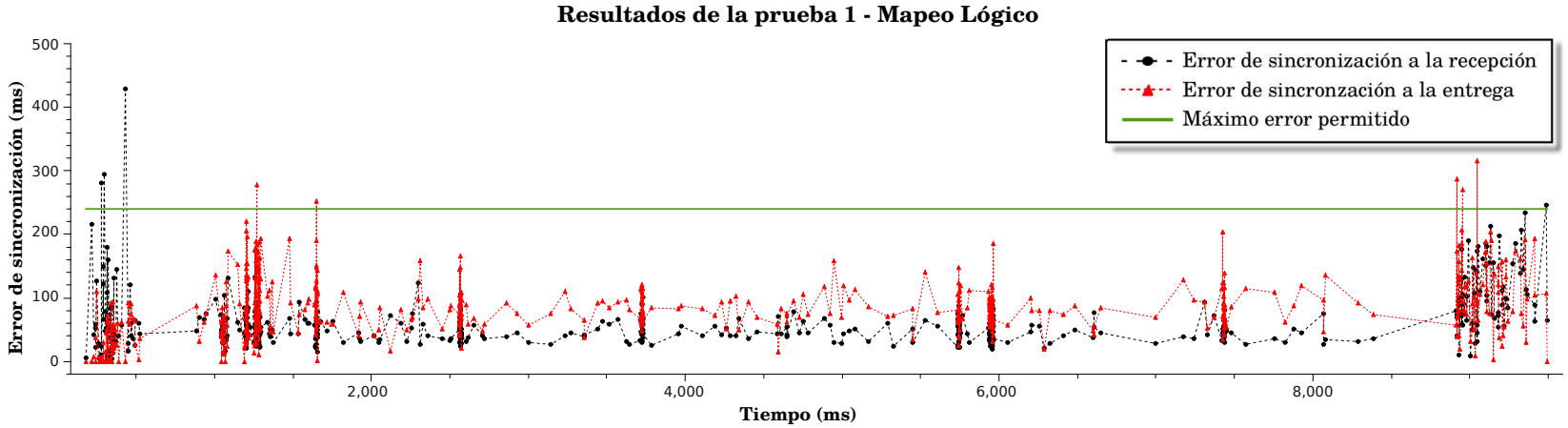


Figura 5.3: Resultados de la prueba 1 con el algoritmo ML.

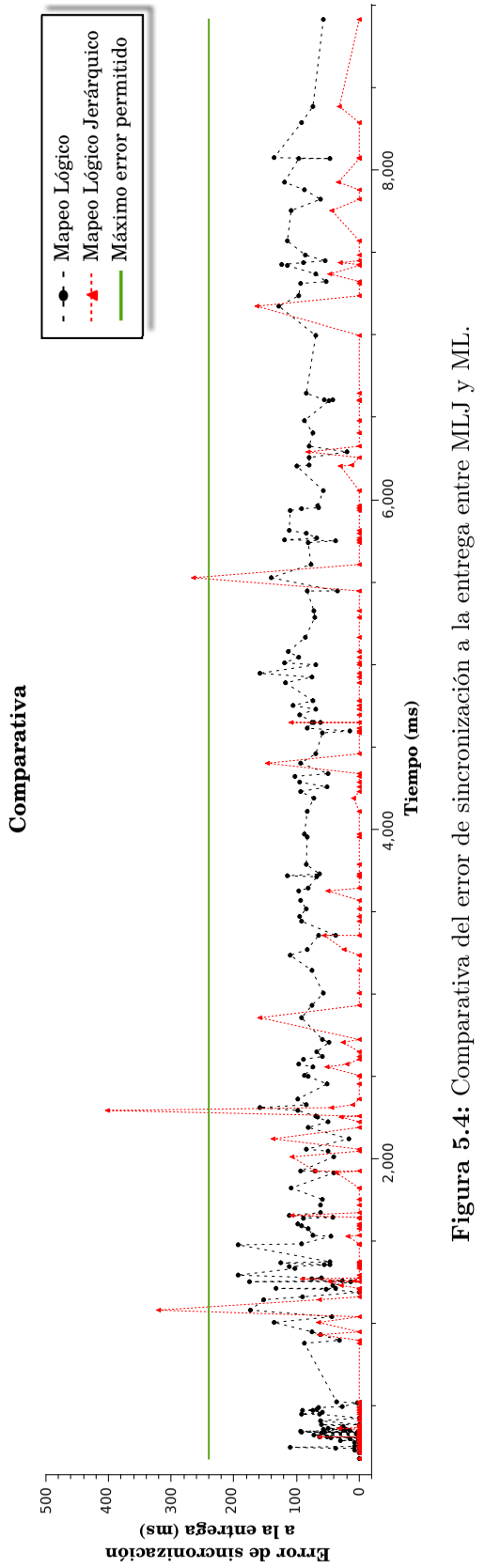


Figura 5.4: Comparativa del error de sincronización a la entrega entre MLJ y ML.

5.3.2. Resultados de la prueba 2

Los canales de comunicación se configuraron con retrasos en un rango de 100 a 300 ms en está prueba. El tiempo máximo de espera (Δ_{ij}) para ambos mecanismos se fijó en 240 ms. Los resultados de las simulaciones se describen a continuación.

En la figura 5.5 se presenta la gráfica de los errores de sincronización a la recepción y a la entrega del mecanismo MLJ. Con las simulaciones obtenemos que el promedio del error de sincronización a la recepción es de 62 ms, mientras que el promedio del error aplicando el mecanismo de corrección es 10.03 ms. También, se obtiene que el 95.08 % de los mensajes están por debajo del error de sincronización a la recepción y del máximo error permitido.

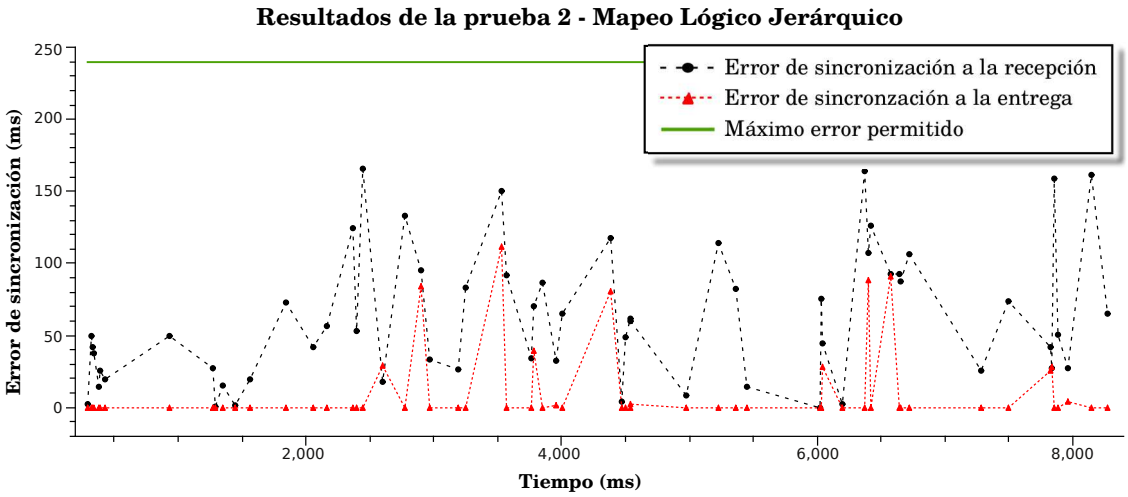


Figura 5.5: Resultados de la prueba 2 con el algoritmo MLJ.

La figura 5.6 muestra la gráfica de los errores de sincronización a la recepción y a la entrega del mecanismo ML. Por medio de las simulaciones obtenemos que el promedio del error de sincronización a la recepción es de 48.79 ms, mientras que el promedio del error aplicando el mecanismo de corrección es 67.47 ms. También, se obtiene que el 15.73 % de los mensajes están por debajo del error de sincronización a la recepción y del máximo error permitido.

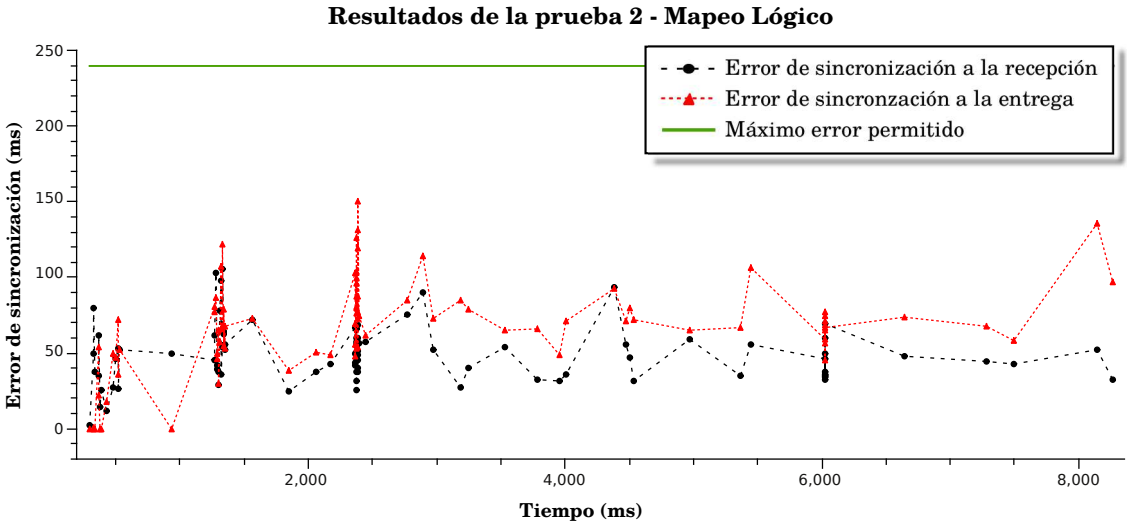


Figura 5.6: Resultados de la prueba 2 con el algoritmo ML.

En la figura 5.7 se describe la comparativa realizada a ambos mecanismos mediante los mismos puntos de sincronización. A través de las simulaciones obtenemos que el promedio del error de sincronización del mecanismo ML es de 60.14 ms, mientras que el promedio del mecanismo MLJ es de 7.72 ms. También, se obtiene que el 82.92 % de los mensajes del mecanismo MLJ, están por debajo del error de sincronización del mecanismo ML y del máximo error permitido.

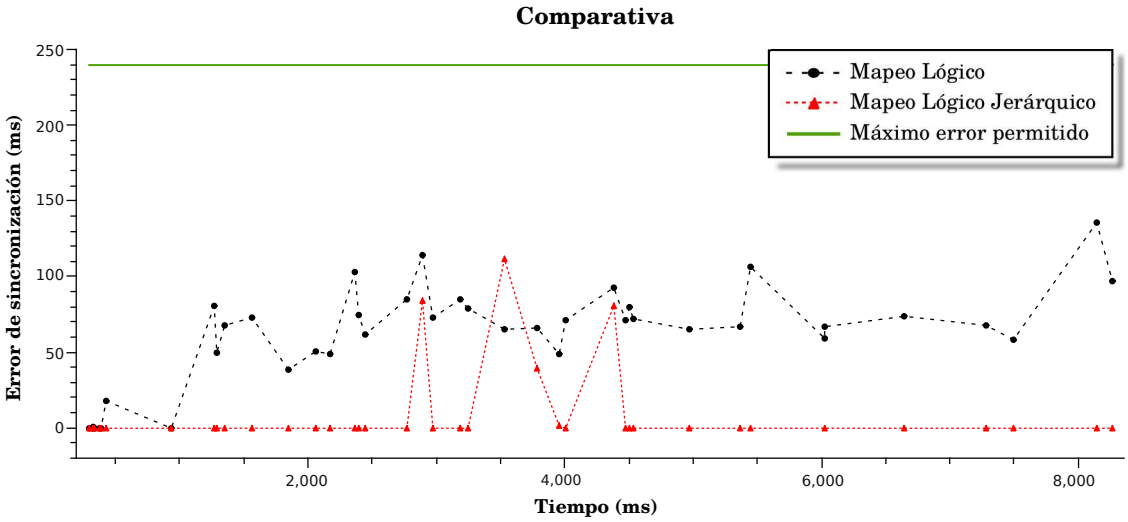


Figura 5.7: Comparativa del error de sincronización a la entrega entre MLJ y ML.

5.3.3. Resultados de la prueba 3

Para esta prueba los canales de comunicación se configuraron con retrasos en un rango de 150 a 400 ms. El tiempo máximo de espera (Δ_{ij}) para ambos mecanismos se fijó en 80 ms. Los resultados de las simulaciones se describen a continuación.

En la figura 5.8 se presenta la gráfica de los errores de sincronización a la recepción y a la entrega del mecanismo MLJ. A partir de las simulaciones obtenemos que el promedio del error de sincronización a la recepción es de 65.13 ms, mientras que el promedio del error aplicando el mecanismo de corrección es 24.83 ms. Se obtiene, además, que el 77.77 % de los mensajes están por debajo del error de sincronización a la recepción y del máximo error permitido.

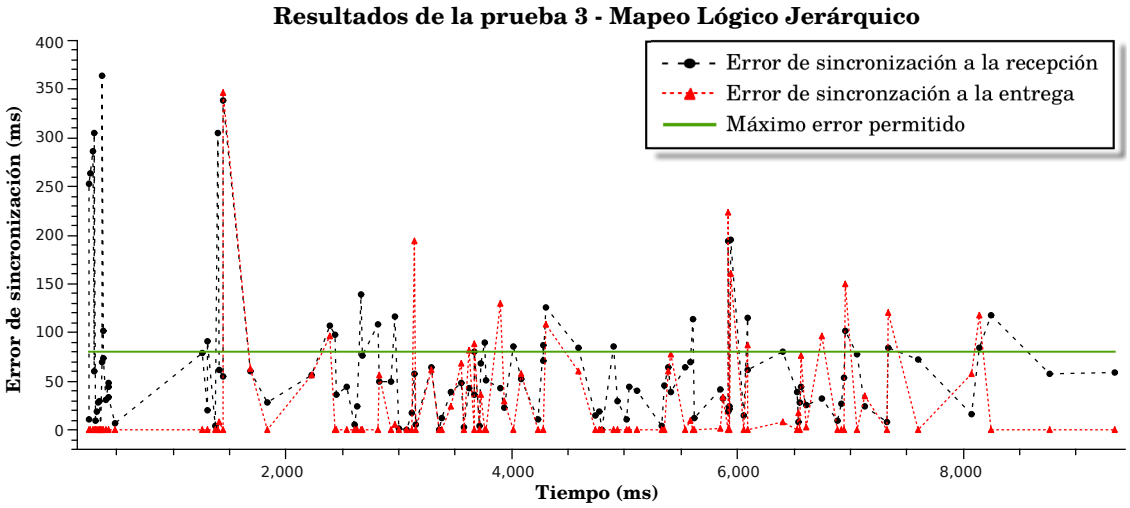


Figura 5.8: Resultados de la prueba 3 con el algoritmo MLJ.

La figura 5.9 despliega la gráfica de los errores de sincronización a la recepción y a la entrega del mecanismo ML. Mediante las simulaciones obtenemos que el promedio del error de sincronización a la recepción es de 52.75 ms, mientras que el promedio del error aplicando el mecanismo de corrección es 72.02 ms. También, se obtiene que el 12.44 % de los mensajes están por debajo del error de sincronización

a la recepción y del máximo error permitido.

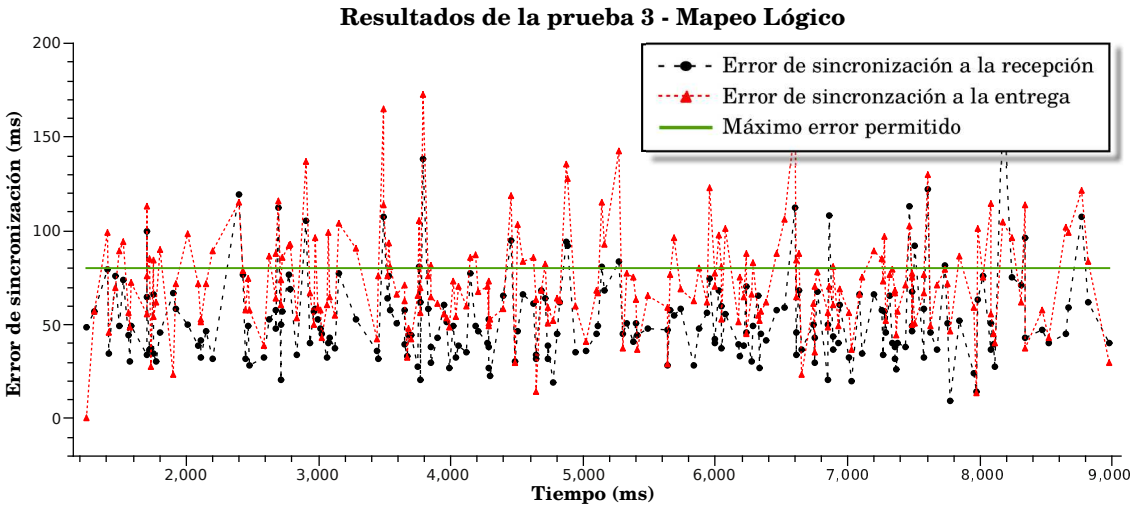


Figura 5.9: Resultados de la prueba 3 con el algoritmo ML.

En la figura 5.10 se muestra la comparativa realizada a ambos mecanismos por medio de los mismos puntos de sincronización. Con la ayuda de las simulaciones obtenemos que el promedio del error de sincronización del mecanismo ML es de 71.41 ms, mientras que el promedio del mecanismo MLJ es de 34.82 ms. También, se obtiene que el 66.66 % de los mensajes del mecanismo MLJ, están por debajo del error de sincronización del mecanismo ML y del máximo error permitido.

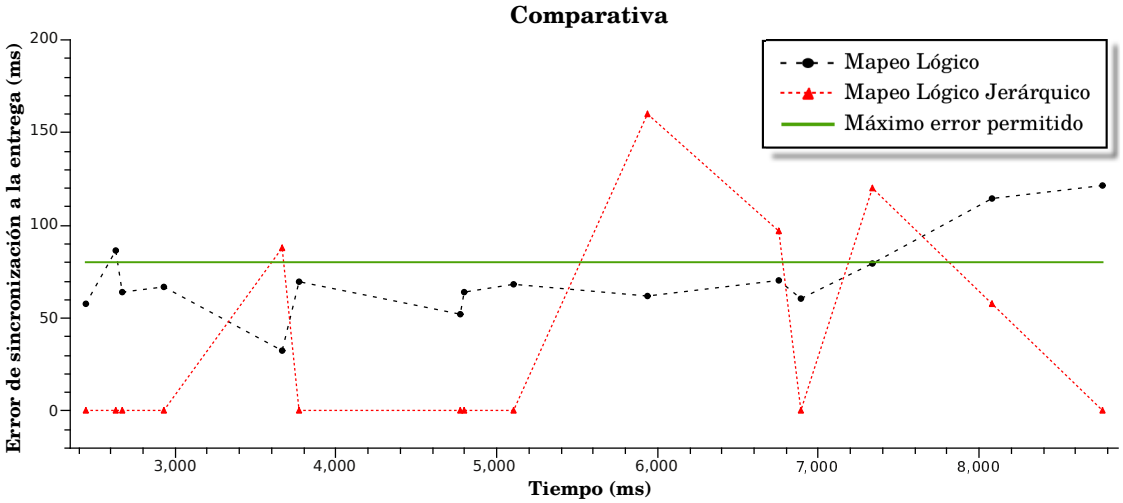


Figura 5.10: Comparativa del error de sincronización a la entrega entre MLJ y ML.

Tabla 5.2: Resultados obtenidos de las pruebas realizadas a los algoritmos MLJ y ML.

Prueba	Mecanismo	Error de sincronización (ms)	
		Entrega	Recepción
1	MLJ	62.55	18.76
	ML	59.69	81.95
2	MLJ	62	10.03
	ML	48.79	67.47
3	MLJ	65.13	24.83
	ML	52.75	72.02

En la tabla 5.2 se presentan los promedios de los resultados obtenidos mediante las pruebas realizadas a los mecanismos MLJ y ML.

5.3.4. Análisis de *overhead* en la comunicación

Uno de los principales objetivos de nuestro mecanismo es la reducción del *overhead* en la comunicación. La reducción del *overhead* es de suma importancia para el soporte a la escalabilidad, dado que los mecanismos deben de seguir trabajando de manera correcta al incrementar sus componentes.

La forma en que se comprobó que el mecanismo desarrollado logra reducir la cantidad de información enviada, fue contabilizando la cantidad de información causal enviada en cada mensaje. El *overhead* por mensaje fue medido con respecto al número de procesos presentes en el sistema. Para resaltar la reducción de

la información causal por mensaje, el *overhead* del mecanismo MLJ se comparó directamente con el *overhead* generado por el mecanismo ML.

En la figura 5.11 se muestra la gráfica comparativa entre el *overhead* obtenido mediante las simulaciones de los mecanismos ML y MLJ. Se observa en la figura que el *overhead* del mecanismo ML comienza creciendo linealmente y a partir de que el número de procesos aumenta a más de catorce se nota una reducción en la información causal.

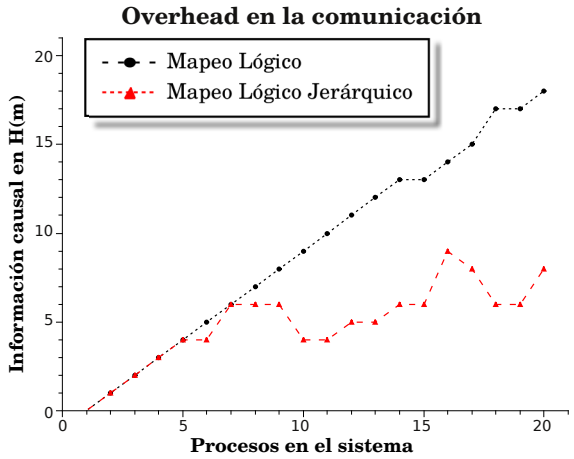


Figura 5.11: Comparativa del *overhead* en la comunicación entre MLJ y ML.

En el caso del mecanismo MLJ el *overhead* presenta una disminución cuando el número de procesos aumenta a más seis al contrario del *overhead* del mecanismo ML; ver figura 5.11. La reducción se debe a que la cantidad de información causal siempre estará acotada por el tamaño de los subgrupos y el número de niveles en la jerarquía.

Capítulo 6

Conclusión y trabajo futuro

6.1. Conclusiones

En este trabajo de tesis se presentó un mecanismo escalable para la sincronización de datos multimedia en tiempo real para la comunicación en grupo, caracterizado por la jerarquización de las entidades en el sistema. Se diseñó un modelo organizacional para la jerarquización de las entidades y un esquema de sincronización maestro/esclavo. El modelo organizacional se caracteriza por la agrupación de los participantes en subgrupos, agrupándolos y jerarquizándolos con el uso de una distancia temporal. La principal característica del esquema de sincronización es la jerarquización de los flujos multimedia, obteniendo como consecuencia la reducción del *overhead* en la comunicación.

Mediante la simulación de nuestro mecanismo, se ha demostrado que jerarquizando las entidades se brinda el soporte a la escalabilidad en los sistemas multimedia. La escalabilidad se midió en términos del *overhead* en la comunicación. El mecanismo aquí presentado logra reducir el *overhead* debido a que cada proceso sólo envía información con respecto a su proceso maestro. También se ha logrado demostrar que la sincronización no se ve afectada con la reducción de información de control, ya que aunque los procesos en el sistema no se sincronizan utilizando la información de todos los procesos en el sistema, si se sincronizan de manera indirecta con un maestro global. Lo anterior se debe a las condiciones de entrega

de mensajes ya que si un mensaje causal de un proceso maestro no ha sido entregado, los mensajes con dependencia causal de un proceso esclavo no pueden ser entregados.

Cabe señalar que nuestro mecanismo sólo ha sido simulado en condiciones de retraso de mensajes. Mediante las simulaciones se mostró que el mecanismo mantiene un correcto funcionamiento ante el retraso de mensajes, obteniendo errores en la sincronización menores al límite establecido para cada prueba. Algo que no se probó fue el comportamiento del mecanismo ante pérdida de mensajes. Para brindar el soporte a la pérdida de mensajes, se recomienda el diseño y desarrollo de mecanismos de recuperación de información. Se hace esta recomendación debido a la reducción de información de control por mensaje.

Por último, mediante la simulación de nuestro mecanismo se ha comprobado su correcto funcionamiento. En consecuencia, nuestro mecanismo es apto para su uso en comunicaciones en grupo y en tiempo real ya que cumple satisfactoriamente con las restricciones de tiempo y de calidad de servicio (Quality of Service (QoS)).

6.2. Trabajo futuro

El trabajo de investigación aquí presentado se centró en brindar el soporte al retraso de mensajes para la sincronización multimedia, se dejan aún abiertos otros problemas existentes. Uno de los siguientes problemas a atacar es la pérdida de paquetes, ya que se deberán de crear mecanismo para la recuperación de datos y así no afectar la sincronización.

Para el funcionamiento del esquema organizacional, se supuso que en la generación de los subgrupos no existían traslapes. Luego, el siguiente paso será refinar el modelo organizacional para ofrecer el soporte a grupos traslapados se debe,

entonces, desarrollar políticas más estrictas para la formación de los subgrupos.

Tras comprobar el funcionamiento de nuestro mecanismo mediante las simulaciones, se concluye que éste podría ser usado en la sincronización de MMS en distintos tipos de redes. Un ejemplo son las redes usadas para la comunicación de dispositivos móviles, ya que al contar con recursos limitados y más restricciones de QoS generan nuevos retos para la transmisión y sincronización de MMS.

Bibliografía

[AB98] Abdelhafid Abouaissa and Abderrahim Benslimane. Hierarchical architecture for real time causal delivery. In *IEEE Conference on Local Computer Networks*, pages 374–383, 1998.

[AHNA12] Ahsan Arefin, Zixia Huang, Klara Nahrstedt, and Pooja Agarwal. 4d telecast: Towards large scale multi-site and multi-view dissemination of 3dti contents. In *IEEE 32nd International Conference on Distributed Computing Systems (ICDCS)*, pages 82–91, 2012.

[All83] James F Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[BA02] Abderrahim Benslimane and Abdelhafid Abouaissa. Dynamical grouping model for distributed real time causal ordering. *Computer Communications*, 25:288–302, 2002.

[BBFV99] Roberto Baldoni, Roberto Beraldi, Roy Friedman, and Robbert Van Renesse. The hierarchical daisy architecture for causal delivery. *Distributed Systems Engineering*, 6:71–81, 1999.

[BGL08] Fernando Boronat Seguí, Juan Carlos Guerri Cebollada, and Jaime Lloret Mauri. An rtp/rtcp based approach for multimedia group and inter-stream synchronization. *Multimedia Tools and Applications*, 40:285–319, 2008.

[BLG09] Fernando Boronat, Jaime Lloret, and Miguel García. Multimedia group and inter-stream synchronization techniques: A comparative study. *Information Systems*, 34(1):108–131, 2009.

[Bon00] André B. Bondi. Characteristics of scalability and their impact on performance. In *Proceedings of the 2nd international workshop on Software and performance*, WOSP '00, pages 195–203, New York, NY, USA, 2000. ACM.

[BS96] Gerold Blakowski and Ralf Steinmetz. A media synchronization survey: Reference model, specification, and case studies. *Selected Areas in Communications, IEEE Journal on*, 14(1):5–35, 1996.

[Cri09] Joel M. Crichlow. *Distributed Systems*. PHI Learning, december 2009.

[Hwa09] Jenq-Neng Hwang. *Multimedia Networking: From Theory to Practice*. Cambridge University Press, 2009.

[HWN+10] Zixia Huang, Wanmin Wu, Klara Nahrstedt, Ahsan Arefin, and Raoul Rivas. Tsync: a new synchronization framework for multi-site 3d teleimmersion. In *Proceedings of the 20th international workshop on Network and operating systems support for digital audio and video*, pages 39–44. ACM, 2010.

[HX97] Anna Haj Hać and Cindy X. Xue. Synchronization in multimedia data retrieval. *International Journal of Network Management*, 7(1):33–62, January 1997.

[IT95] Yutaka Ishibashi and Shuji Tasaka. A synchronization mechanism for continuous media in multimedia communications. In *IEEE INFOCOM*, pages 1010–1019, 1995.

[IT97] Yutaka Ishibashi and Shuji Tasaka. A group synchronization mechanism for live media in multicast communications. In *Global Telecommunications Conference, 1997. GLOBECOM'97., IEEE*, volume 2, pages 746–752. IEEE, 1997.

[itu01] End-user multimedia qos categories. series g: Transmission systems and media, digital systems and networks. quality of service and performance. G.1010, november 2001. ITU-T Telecommunication Standarization Sector of ITU.

[Lam78] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[Lam86] Leslie Lamport. On interprocess communication. *Distributed Computing*, 1(2):77–85, 1986.

[LZKM10] Yue Lu, Yong Zhao, Fernando Kuipers, and Piet Mieghem. Measurement study of multi-party video conferencing. In Mark Crovella, LauraMarie Feeney, Dan Rubenstein, and S.V. Raghavan, editors, *NETWORKING 2010*, volume 6091 of *Lecture Notes in Computer Science*, pages 96–108. Springer Berlin Heidelberg, 2010.

[Mat89] Friedemann Mattern. Virtual time and global states of distributed systems. In *Proceedings of the International Workshop on Parallel and Distributed Algorithms*, pages 215–226. Elsevier, 1989.

[MBSv12] Mario Montagud, Fernando Boronat, Hans Stokking, and Ray van Brandenburg. Inter-destination multimedia synchronization: schemes, use cases and standardization. *Multimedia Systems*, 18(6):459–482, 2012.

[MIF+11] Yuji Miyashita, Yutaka Ishibashi, Norishige Fukushima, Shinji Sugawara, and Kostas E. Psannis. Qoe assessment of group synchroniza-

tion in networked chorus with voice and video. In *IEEE Region 10 Conference TENCON 2011*, pages 192–196, 2011.

[Mil91] David L Mills. Internet time synchronization: the network time protocol. *IEEE Transactions on Communications*, 39(10):1482–1493, 1991.

[MRBF97] David F McAllister, Robert D Rodman, Donald L Bitzer, and Andrew S Freeman. Lip synchronization of speech. In *Audio-Visual Speech Processing: Computational & Cognitive Science Approaches*, 1997.

[NET05] Yasutaka Nishimura, Tomoya Enokido, and Makoto Takizawa. Design of a hierarchical group to realize a scalable group. In *Proceedings of the 19th International Conference on Advanced Information Networking and Applications*, volume 1, pages 9–14. IEEE, 2005.

[PEMR09] Saul Pomares Hernandez, Jorge Estudillo Ramirez, Luis A. Morales Rosales, and Gustavo Rodríguez Gómez. An intermedia synchronisation mechanism for multimedia distributed systems. *International Journal of Internet Protocol Technology*, 4:207–218, 2009.

[Per03] Colin Perkins. *Rtp: Audio and Video for the Internet*. Kaleidoscope series. Pearson Addison Wesley Prof, 2003.

[PMER08] Saul E. Pomares Hernandez, Luis A. Morales Rosales, Jorge Estudillo Ramirez, and Gustavo Rodríguez Gómez. Logical mapping: An intermedia synchronization model for multimedia distributed systems. *Journal of Multimedia*, 3:33–41, 2008.

[RR93] Srinivas Ramanathan and P. Venkat Rangan. Adaptive feedback techniques for synchronized multimedia retrieval over integrated networks. *IEEE/ACM Trans. Networking*, 1(2):246–260, April 1993.

[sin13] Sinalgo - simulator for network algorithms. http://disco.ethz.ch/projects/sinalgo/, july 2013. Distributed Computing Group, ETH Zurich.

[SN04] Ralf Steinmetz and Klara Nahrstedt. *Multimedia Systems*. X.media.publishing. Springer Berlin Heidelberg, 2004.

[UB12] Shahab Ud Din and Dick Bulterman. Synchronization techniques in distributed multimedia presentation. In *MMEDIA 2012, The Fourth International Conferences on Advances in Multimedia*, pages 1–9, 2012.

[Var09] Vasudeva Varma. *Software Architecture: A Case Based Approach*. Pearson Education, India, november 2009.

[Vil82] Marc B Vilain. A system for reasoning about time. In *2nd. (US) National Conference on Artificial Intelligence*, volume 82, pages 197–201, 1982.

[YF96] Wei Yen and Ian F. Akyildiz. A hierarchical architecture for buffer management in integrated services networks. *Multimedia Systems*, 4:131–139, 1996.

[YK13] Amir Yahyavi and Bettina Kemme. Peer-to-peer architectures for massively multiplayer online games: A survey. *ACM Computing Surveys*, 1, 2013.

Apéndice A

Ejemplo de funcionamiento del algoritmo jerárquico

En este anexo se da un ejemplo del funcionamiento del algoritmo jerárquico expuesto en el capítulo 4, mediante un escenario compuesto por doce procesos (ver figuras A.1 y A.2). El ejemplo muestra cómo se comportan las estructuras de datos utilizadas, el intercambio de mensajes entre los procesos, la identificación de un proceso maestro y la sincronización de los esclavos mediante las referencias de un proceso maestro.

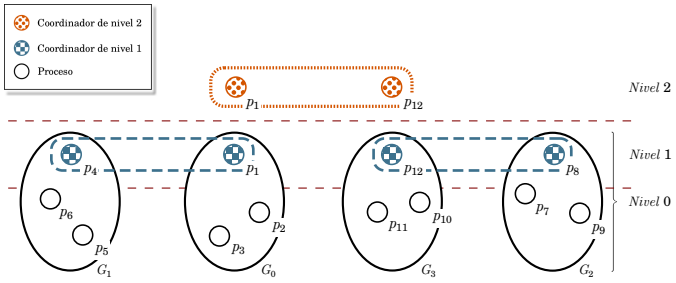


Figura A.1: Arquitectura a tres niveles con doce procesos.

En la figura A.1 se presenta una arquitectura a tres niveles, formada por cuatro subgrupos de tres procesos y tres subgrupos de coordinadores/maestros con dos procesos organizados jerárquicamente. Los diagramas de la figura A.2 muestran como los procesos intercambian información mediante el envío de mensajes. Este escenario ayudará a entender los pasos expuestos en las tablas A.1, A.2, A.3 y A.4.

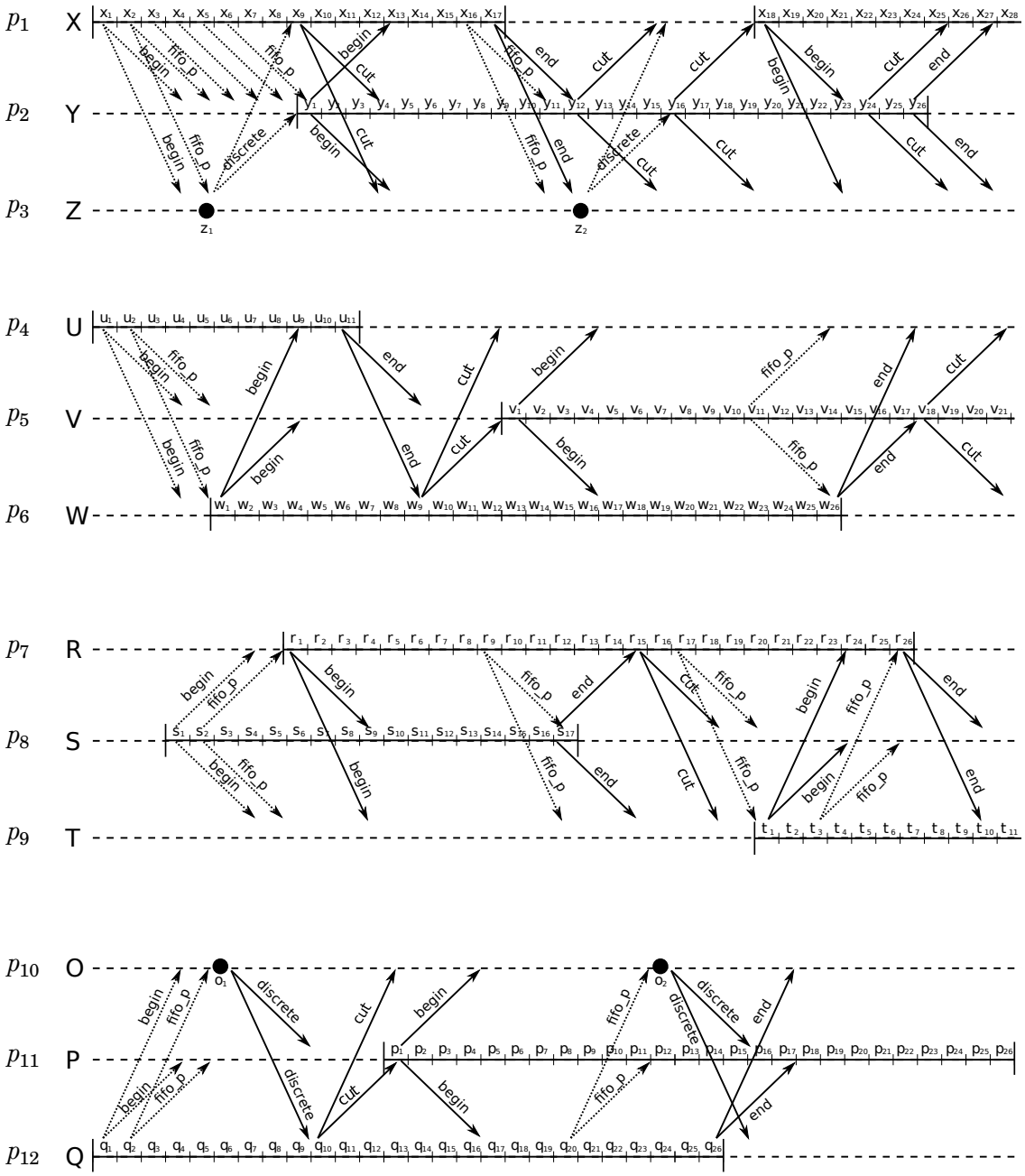


Figura A.2: Flujos generados por cada proceso pertenecientes al ejemplo de la imagen A.1.

En la tabla A.1 se muestra la interacción entre los procesos pertenecientes al subgrupo G_0 y como estos sincronizan sus MMS. En esta tabla se muestra el com-

portamiento del proceso p_1, el cual funge como maestro del subgrupo G_0 además de ser tanto maestro de nivel 1 como de nivel 2.

La tabla A.2 presenta el comportamiento de los procesos pertenecientes al subgrupo G_1. Mediante esta tabla se muestra el comportamiento del proceso p_4, el cual funge como maestro del subgrupo G_0 y como maestro de nivel 1. En este mismo ejemplo se puede observar como los procesos en G_1 preservan una vista parcial del sistema, aún sólo sincronizando sus MMS con respecto a su maestro.

En la tabla A.3 se muestra la interacción entre los procesos pertenecientes al subgrupo G_2. En esta tabla se muestra el comportamiento del proceso p_8, el cual funge como maestro del subgrupo G_2 y como maestro de nivel 1. Al igual que en la tabla A.2, en esta tabla se observar como los procesos sincronizan sus MMS locales.

La tabla A.4 presenta el comportamiento de los procesos pertenecientes al subgrupo G_3. En esta tabla se muestra el comportamiento del proceso p_12, el cual funge como maestro del subgrupo G_3 además de ser tanto maestro de nivel 1 como de nivel 2.

Tabla A.1: Prueba del escenario de la figura A.2, subgrupo G_0.

Prueba del escenario de la figura A.2, subgrupo G_0		
p_1	p_2	p_3
x_1. $SendContinuous(begin)$ $VT(p)[1] = 1$ $Act = 1$ $last\_fifo(p) \leftarrow \emptyset$ $Slaves \leftarrow \emptyset$ $level = 0$ $Master = 0$ $level = 1$ $Master = 0$ $level = 2$ $Master = 0$ $level = 3$ $H(m) \leftarrow CI(p)$ $m = (1, 1, 0, begin, \emptyset, data)$ $sending(m)$		
u_1. $Receive(begin)$ $delivery(m)$ $VT(p)[4] = 1$ $CI(p) \leftarrow CI(p) \cup (4, 1)$ $ActP \leftarrow ActP \cup \{4\}$ $Slaves \leftarrow Slaves \cup \{4\}$	u_1. $Receive(begin)$ $delivery(m)$ $VT(p)[4] = 1$ $CI(p) \leftarrow CI(p) \cup (4, 1)$ $ActP \leftarrow ActP \cup \{4\}$	u_1. $Receive(begin)$ $delivery(m)$ $VT(p)[4] = 1$ $CI(p) \leftarrow CI(p) \cup (4, 1)$ $ActP \leftarrow ActP \cup \{4\}$
q_1. $Receive(begin)$ $delivery(m)$ $VT(p)[12] = 1$ $CI(p) \leftarrow CI(p) \cup (12, 1)$ $ActP \leftarrow ActP \cup \{12\}$	q_1. $Receive(begin)$ $delivery(m)$ $VT(p)[12] = 1$ $CI(p) \leftarrow CI(p) \cup (12, 1)$ $ActP \leftarrow ActP \cup \{12\}$	q_1. $Receive(begin)$ $delivery(m)$ $VT(p)[12] = 1$ $CI(p) \leftarrow CI(p) \cup (12, 1)$ $ActP \leftarrow ActP \cup \{12\}$
	x_1. $Receive(begin)$ $delivery(m)$ $VT(p)[1] = 1$ $CI(p) \leftarrow CI(p) \cup (1, 1)$ $ActP \leftarrow ActP \cup \{1\}$	x_1. $Receive(begin)$ $delivery(m)$ $VT(p)[1] = 1$ $CI(p) \leftarrow CI(p) \cup (1, 1)$ $ActP \leftarrow ActP \cup \{1\}$

Tabla A.1: Continuación de la prueba del escenario de la figura A.2, subgrupo G_0.

Continuación de la prueba del escenario de la figura A.2, subgrupo G_0		
p_1	p_2	p_3
		z_1. $SendDiscrete(discrete)$ $VT(p)[3] = 1$ $CI(p) \leftarrow (CI(p) \backslash (1,1)) \cup (1,2)$ $Master = 1$ $H(m) \leftarrow \{(1,2)\}$ $last\_fifo(p) \leftarrow \emptyset$ $CI(p) \leftarrow \emptyset$ $m = (3,1,1,discrete, \{(1,2)\},$ $data)$ $sending(m)$
s_1. $Receive(begin)$ $delivery(m)$ $VT(p)[8] = 1$ $CI(p) \leftarrow CI(p) \cup (8,1)$ $ActP \leftarrow ActP \cup \{8\}$	s_1. $Receive(begin)$ $delivery(m)$ $VT(p)[8] = 1$ $CI(p) \leftarrow CI(p) \cup (8,1)$ $CI(p) \leftarrow CI(p) \backslash (12,1)$ $ActP \leftarrow ActP \cup \{8\}$	s_1. $Receive(begin)$ $delivery(m)$ $VT(p)[8] = 1$ $CI(p) \leftarrow CI(p) \cup (8,1)$ $CI(p) \leftarrow CI(p) \backslash (12,1)$ $ActP \leftarrow ActP \cup \{8\}$
z_1. $Receive(discrete)$ $delivery(m)$ $VT(p)[3] = 1$ $CI(p) \leftarrow CI(p) \cup (3,1)$ $ActP \leftarrow ActP \cup \{3\}$ $Slaves \leftarrow Slaves \cup \{3\}$ $SendContinuous(cut)$	z_1. $Receive(discrete)$ $delivery(m)$ $VT(p)[3] = 1$ $CI(p) \leftarrow CI(p) \cup (3,1)$	

Tabla A.1: Continuación de la prueba del escenario de la figura A.2, subgrupo G_0.

Continuación de la prueba del escenario de la figura A.2, subgrupo G_0		
p_1	p_2	p_3
x_9. $SendContinuous(cut)$ $VT(p)[1] = 9$ $H(m) \leftarrow H(m) \cup (4,1)$ $H(m) \leftarrow H(m) \cup (3,1)$ $Slaves \leftarrow \emptyset$ $CI(p) \leftarrow \emptyset$ $m = (1, 9, 0, cut, \{(4,1),(3,1)\},$ $data)$ $sending(m)$	y_1. $SendContinuous(begin)$ $VT(p)[2] = 1$ $Act = 1$ $CI(p) \leftarrow (CI(p) \backslash (4,1)) \cup (4,6)$ $CI(p) \leftarrow (CI(p) \backslash (1,1)) \cup (1,6)$ $CI(p) \leftarrow (CI(p) \backslash (8,1)) \cup (8,2)$ $last\_fifo(p) \leftarrow \emptyset$ $Slaves \leftarrow \emptyset$ $level = 0$ $Master = 1$ $level = 1$ $H(m) \leftarrow H(m) \cup (1,6)$ $m = (2, 1, 1, begin, \{(1,6)\}, data)$ $sending(m)$	
w_1. $Receive(begin)$ $delivery(m)$ $VT(p)[6] = 1$ $CI(p) \leftarrow CI(p) \cup (6,1)$ $ActP \leftarrow ActP \cup \{6\}$	w_1. $Receive(begin)$ $delivery(m)$ $VT(p)[6] = 1$ $CI(p) \leftarrow CI(p) \cup (6,1)$ $ActP \leftarrow ActP \cup \{6\}$	w_1. $Receive(begin)$ $delivery(m)$ $VT(p)[6] = 1$ $CI(p) \leftarrow CI(p) \cup (6,1)$ $CI(p) \leftarrow CI(p) \backslash (4,1)$ $ActP \leftarrow ActP \cup \{6\}$
o_1. $Receive(discrete)$ $delivery(m)$ $VT(p)[10] = 1$ $CI(p) \leftarrow CI(p) \cup (10,1)$	o_1. $Receive(discrete)$ $delivery(m)$ $VT(p)[10] = 1$ $CI(p) \leftarrow CI(p) \cup (10,1)$	o_1. $Receive(discrete)$ $delivery(m)$ $VT(p)[10] = 1$ $CI(p) \leftarrow CI(p) \cup (10,1)$
y_1. $Receive(begin)$ $delivery(m)$ $VT(p)[2] = 1$ $CI(p) \leftarrow CI(p) \cup (2,1)$ $ActP \leftarrow ActP \cup \{2\}$ $Slaves \leftarrow Slaves \cup \{2\}$		y_1. $Receive(begin)$ $delivery(m)$ $VT(p)[2] = 1$ $CI(p) \leftarrow CI(p) \cup (2,1)$ $ActP \leftarrow ActP \cup \{2\}$
	x_9. $Receive(cut)$ $delivery(m)$ $VT(p)[1] = 9$ $CI(p) \leftarrow CI(p) \cup (1,9)$	x_9. $Receive(cut)$ $delivery(m)$ $VT(p)[1] = 9$ $CI(p) \leftarrow CI(p) \cup (1,9)$

Tabla A.1: Continuación de la prueba del escenario de la figura A.2, subgrupo G_0.

Continuación de la prueba del escenario de la figura A.2, subgrupo G_0		
p_1	p_2	p_3
u_{11}. $Receive(end)$ $delivery(m)$ $VT(p)[4] = 11$ $CI(p) \leftarrow CI(p) \cup (4, 11)$ $ActP \leftarrow ActP \backslash \{4\}$ $SendContinuous(cut)$	u_{11}. $Receive(end)$ $delivery(m)$ $VT(p)[4] = 11$ $CI(p) \leftarrow CI(p) \cup (4, 11)$ $CI(p) \leftarrow CI(p) \backslash (6, 1)$ $ActP \leftarrow ActP \cup \{4\}$	u_{11}. $Receive(end)$ $delivery(m)$ $VT(p)[4] = 11$ $CI(p) \leftarrow CI(p) \cup (4, 11)$ $CI(p) \leftarrow CI(p) \backslash (6, 1)$ $ActP \leftarrow ActP \cup \{4\}$
x_{16}. $SendContinuous(cut)$ $VT(p)[1] = 16$ $H(m) \leftarrow H(m) \cup (4, 11)$ $H(m) \leftarrow H(m) \cup (2, 1)$ $Slaves \leftarrow \emptyset$ $CI(p) \leftarrow \emptyset$ $m = (1, 16, 0, cut, \{(4, 11), (2, 1)\},$ $data)$ $sending(m)$		
q_{10}. $Receive(cut)$ $delivery(m)$ $VT(p)[12] = 10$ $CI(p) \leftarrow CI(p) \cup (12, 10)$ $SendContinuous(cut)$	q_{10}. $Receive(cut)$ $delivery(m)$ $VT(p)[12] = 10$ $CI(p) \leftarrow CI(p) \cup (12, 10)$ $CI(p) \leftarrow CI(p) \backslash (10, 1)$	q_{10}. $Receive(cut)$ $delivery(m)$ $VT(p)[12] = 10$ $CI(p) \leftarrow CI(p) \cup (12, 10)$ $CI(p) \leftarrow CI(p) \backslash (10, 1)$
r_1. $Receive(begin)$ $delivery(m)$ $VT(p)[7] = 1$ $CI(p) \leftarrow CI(p) \cup (7, 1)$ $ActP \leftarrow ActP \cup \{7\}$	r_1. $Receive(begin)$ $delivery(m)$ $VT(p)[7] = 1$ $CI(p) \leftarrow CI(p) \cup (7, 1)$ $ActP \leftarrow ActP \cup \{7\}$	r_1. $Receive(begin)$ $delivery(m)$ $VT(p)[7] = 1$ $CI(p) \leftarrow CI(p) \cup (7, 1)$ $CI(p) \leftarrow CI(p) \backslash (8, 1)$ $ActP \leftarrow ActP \cup \{7\}$
	x_{16}. $Receive(cut)$ $delivery(m)$ $VT(p)[1] = 16$ $CI(p) \leftarrow CI(p) \backslash (1, 9)$ $CI(p) \leftarrow CI(p) \cup (1, 16)$ $CI(p) \leftarrow CI(p) \backslash (4, 11)$	x_{16}. $Receive(cut)$ $delivery(m)$ $VT(p)[1] = 16$ $CI(p) \leftarrow CI(p) \backslash (1, 9)$ $CI(p) \leftarrow CI(p) \cup (1, 16)$ $CI(p) \leftarrow CI(p) \backslash (4, 11)$ $CI(p) \leftarrow CI(p) \backslash (2, 1)$

Tabla A.1: Continuación de la prueba del escenario de la figura A.2, subgrupo G_0.

Continuación de la prueba del escenario de la figura A.2, subgrupo G_0		
p_1	p_2	p_3
x_{17}. $SendContinuous(end)$ $VT(p)[1] = 17$ $Act = 0$ $H(m) \leftarrow H(m) \cup (12, 10)$ $Slaves \leftarrow \emptyset$ $CI(p) \leftarrow \emptyset$ $m = (1, 17, 0, end, \{(12, 10)\},$ $data)$ $sending(m)$ $Master = 0$		
w_9. $Receive(cut)$ $delivery(m)$ $VT(p)[6] = 9$ $CI(p) \leftarrow CI(p) \cup (6, 9)$	w_9. $Receive(cut)$ $delivery(m)$ $VT(p)[6] = 9$ $CI(p) \leftarrow CI(p) \cup (6, 9)$	w_9. $Receive(cut)$ $delivery(m)$ $VT(p)[6] = 9$ $CI(p) \leftarrow CI(p) \cup (6, 9)$
p_1. $Receive(begin)$ $delivery(m)$ $VT(p)[11] = 1$ $CI(p) \leftarrow CI(p) \cup (11, 1)$ $ActP \leftarrow ActP \cup \{11\}$	p_1. $Receive(begin)$ $delivery(m)$ $VT(p)[11] = 1$ $CI(p) \leftarrow CI(p) \cup (11, 1)$ $CI(p) \leftarrow CI(p) \backslash (12, 10)$ $ActP \leftarrow ActP \cup \{11\}$	p_1. $Receive(begin)$ $delivery(m)$ $VT(p)[11] = 1$ $CI(p) \leftarrow CI(p) \cup (11, 1)$ $CI(p) \leftarrow CI(p) \backslash (12, 10)$ $ActP \leftarrow ActP \cup \{11\}$
	x_{17}. $Receive(end)$ $delivery(m)$ $VT(p)[1] = 17$ $CI(p) \leftarrow CI(p) \backslash (1, 16)$ $CI(p) \leftarrow CI(p) \cup (1, 17)$ $ActP \leftarrow ActP \backslash \{1\}$ $Slaves \leftarrow Slaves \cup \{1\}$ $Master = 0$ $SendContinuous(cut)$	x_{17}. $Receive(end)$ $delivery(m)$ $VT(p)[1] = 17$ $CI(p) \leftarrow CI(p) \backslash (1, 16)$ $CI(p) \leftarrow CI(p) \cup (1, 17)$ $ActP \leftarrow ActP \backslash \{1\}$ $Master = 0$

Tabla A.1: Continuación de la prueba del escenario de la figura A.2, subgrupo G_0.

Continuación de la prueba del escenario de la figura A.2, subgrupo G_0		
p_1	p_2	p_3
	y_{12}. $SendContinuous(cut)$ $VT(p)[2] = 12$ $H(m) \leftarrow H(m) \cup (1, 17)$ $Slaves \leftarrow \emptyset$ $CI(p) \leftarrow \emptyset$ $m = (2, 12, 0, cut, \{(1, 17)\}, data)$ $sending(m)$	z_1. $SendDiscrete(discrete)$ $VT(p)[3] = 2$ $CI(p) \leftarrow (CI(p) \backslash (7, 1)) \cup (7, 9)$ $CI(p) \leftarrow (CI(p) \backslash (6, 9)) \cup (6, 12)$ $CI(p) \leftarrow (CI(p) \backslash (11, 1)) \cup (11, 5)$ $Master = 2$ $H(m) \leftarrow CI(p)$ $last\_fifo(p) \leftarrow \emptyset$ $CI(p) \leftarrow \emptyset$ $m = (3, 2, 2, discrete, \{(7, 9),$ $(6, 12), (11, 5)\}, data)$ $sending(m)$

Tabla A.2: Prueba del escenario de la figura A.2, subgrupo G_1.

Prueba del escenario de la figura A.2, subgrupo G_1		
p_4	p_5	p_6
u_1. $SendContinuous(begin)$ $VT(p)[4] = 1$ $Act = 1$ $last\_fifo(p) \leftarrow \emptyset$ $Slaves \leftarrow \emptyset$ $level = 0$ $Master = 0$ $level = 1$ $Master = 0$ $level = 2$ $H(m) \leftarrow CI(p)$ $m = (4, 1, 0, begin, \emptyset, data)$ $sending(m)$		

Tabla A.2: Continuación de la prueba del escenario de la figura A.2, subgrupo G_1.

Continuación de la prueba del escenario de la figura A.2, subgrupo G_1		
p_4	p_5	p_6
	u_1. $Receive(begin)$ $delivery(m)$ $VT(p)[4] = 1$ $CI(p) \leftarrow CI(p) \cup (4, 1)$ $ActP \leftarrow ActP \cup \{4\}$	u_1. $Receive(begin)$ $delivery(m)$ $VT(p)[4] = 1$ $CI(p) \leftarrow CI(p) \cup (4, 1)$ $ActP \leftarrow ActP \cup \{4\}$
q_1. $Receive(begin)$ $delivery(m)$ $VT(p)[12] = 1$ $CI(p) \leftarrow CI(p) \cup (12, 1)$ $ActP \leftarrow ActP \cup \{12\}$	q_1. $Receive(begin)$ $delivery(m)$ $VT(p)[12] = 1$ $CI(p) \leftarrow CI(p) \cup (12, 1)$ $ActP \leftarrow ActP \cup \{12\}$	q_1. $Receive(begin)$ $delivery(m)$ $VT(p)[12] = 1$ $CI(p) \leftarrow CI(p) \cup (12, 1)$ $ActP \leftarrow ActP \cup \{12\}$
x_1. $Receive(begin)$ $delivery(m)$ $VT(p)[1] = 1$ $CI(p) \leftarrow CI(p) \cup (1, 1)$ $ActP \leftarrow ActP \cup \{1\}$ $Master = 1$	x_1. $Receive(begin)$ $delivery(m)$ $VT(p)[1] = 1$ $CI(p) \leftarrow CI(p) \cup (1, 1)$ $ActP \leftarrow ActP \cup \{1\}$	x_1. $Receive(begin)$ $delivery(m)$ $VT(p)[1] = 1$ $CI(p) \leftarrow CI(p) \cup (1, 1)$ $ActP \leftarrow ActP \cup \{1\}$
		w_1. $SendContinuous(begin)$ $VT(p)[6] = 1$ $Act = 1$ $CI(p) \leftarrow (CI(p) \backslash (4, 2)) \cup (4, 2)$ $last\_fifo(p) \leftarrow \emptyset$ $Slaves \leftarrow \emptyset$ $level = 0$ $Master = 4$ $level = 1$ $H(m) \leftarrow H(m) \cup (4, 2)$ $Slaves \leftarrow \emptyset$ $CI(p) \leftarrow \emptyset$ $m = (6, 1, 4, begin, \{(4, 2)\}, data)$ $sending(m)$

Tabla A.2: Continuación de la prueba del escenario de la figura A.2, subgrupo G_1.

Continuación de la prueba del escenario de la figura A.2, subgrupo G_1		
p_4	p_5	p_6
s_1. $Receive(begin)$ $delivery(m)$ $VT(p)[8] = 1$ $CI(p) \leftarrow CI(p) \cup (8,1)$ $CI(p) \leftarrow CI(p) \backslash (12,1)$ $ActP \leftarrow ActP \cup \{8\}$	s_1. $Receive(begin)$ $delivery(m)$ $VT(p)[8] = 1$ $CI(p) \leftarrow CI(p) \cup (8,1)$ $CI(p) \leftarrow CI(p) \backslash (12,1)$ $ActP \leftarrow ActP \cup \{8\}$	s_1. $Receive(begin)$ $delivery(m)$ $VT(p)[8] = 1$ $CI(p) \leftarrow CI(p) \cup (8,1)$ $CI(p) \leftarrow CI(p) \backslash (12,1)$ $ActP \leftarrow ActP \cup \{8\}$
w_1. $Receive(begin)$ $delivery(m)$ $VT(p)[6] = 1$ $CI(p) \leftarrow CI(p) \cup (6,1)$ $ActP \leftarrow ActP \cup \{6\}$ $Slaves \leftarrow Slaves \cup \{6\}$	w_1. $Receive(begin)$ $delivery(m)$ $VT(p)[6] = 1$ $CI(p) \leftarrow CI(p) \cup (6,1)$ $ActP \leftarrow ActP \cup \{6\}$	
z_1. $Receive(discrete)$ $delivery(m)$ $VT(p)[3] = 1$ $CI(p) \leftarrow CI(p) \cup (3,1)$	z_1. $Receive(discrete)$ $delivery(m)$ $VT(p)[3] = 1$ $CI(p) \leftarrow CI(p) \cup (3,1)$ $CI(p) \leftarrow CI(p) \backslash (1,1)$	z_1. $Receive(discrete)$ $delivery(m)$ $VT(p)[3] = 1$ $CI(p) \leftarrow CI(p) \cup (3,1)$
o_1. $Receive(discrete)$ $delivery(m)$ $VT(p)[10] = 1$ $CI(p) \leftarrow CI(p) \cup (10,1)$	o_1. $Receive(discrete)$ $delivery(m)$ $VT(p)[10] = 1$ $CI(p) \leftarrow CI(p) \cup (10,1)$	o_1. $Receive(discrete)$ $delivery(m)$ $VT(p)[10] = 1$ $CI(p) \leftarrow CI(p) \cup (10,1)$
u_{11}. $SendContinuous(end)$ $VT(p)[4] = 11$ $Act = 0$ $H(m) \leftarrow H(m) \cup (1,1)$ $H(m) \leftarrow H(m) \cup (6,1)$ $Slaves \leftarrow \emptyset$ $CI(p) \leftarrow \emptyset$ $m = (4, 11, 1, end, \{(1,1),(6,1)\},$ $data)$ $sending(m)$ $Master = 0$		

Tabla A.2: Continuación de la prueba del escenario de la figura A.2, subgrupo G_1.

Continuación de la prueba del escenario de la figura A.2, subgrupo G_1		
p_4	p_5	p_6
y_1. $Receive(begin)$ $delivery(m)$ $VT(p)[2] = 1$ $CI(p) \leftarrow CI(p) \cup (2,1)$ $ActP \leftarrow ActP \cup \{2\}$	y_1. $Receive(begin)$ $delivery(m)$ $VT(p)[2] = 1$ $CI(p) \leftarrow CI(p) \cup (2,1)$ $ActP \leftarrow ActP \cup \{2\}$	y_1. $Receive(begin)$ $delivery(m)$ $VT(p)[2] = 1$ $CI(p) \leftarrow CI(p) \cup (2,1)$ $ActP \leftarrow ActP \cup \{2\}$
x_9. $Receive(cut)$ $delivery(m)$ $VT(p)[1] = 9$ $CI(p) \leftarrow CI(p) \cup (1,9)$	x_9. $Receive(cut)$ $delivery(m)$ $VT(p)[1] = 9$ $CI(p) \leftarrow CI(p) \cup (1,9)$ $CI(p) \leftarrow CI(p) \backslash (3,1)$	x_9. $Receive(cut)$ $delivery(m)$ $VT(p)[1] = 9$ $CI(p) \leftarrow CI(p) \cup (1,9)$ $CI(p) \leftarrow CI(p) \backslash (3,1)$
	u_{11}. $Receive(end)$ $delivery(m)$ $VT(p)[4] = 11$ $CI(p) \leftarrow CI(p) \backslash (4,1)$ $CI(p) \leftarrow CI(p) \cup (4,11)$ $CI(p) \leftarrow CI(p) \backslash (6,1)$ $ActP \leftarrow ActP \backslash \{4\}$ $Master = 0$	u_{11}. $Receive(end)$ $delivery(m)$ $VT(p)[4] = 11$ $CI(p) \leftarrow CI(p) \cup (4,11)$ $ActP \leftarrow ActP \backslash \{4\}$ $Slaves \leftarrow Slaves \cup \{4\}$ $Master = 0$ $SendContinuous(cut)$
q_{10}. $Receive(cut)$ $delivery(m)$ $VT(p)[12] = 10$ $CI(p) \leftarrow CI(p) \cup (12,10)$	q_{10}. $Receive(cut)$ $delivery(m)$ $VT(p)[12] = 10$ $CI(p) \leftarrow CI(p) \cup (12,10)$ $CI(p) \leftarrow CI(p) \backslash (10,1)$	q_{10}. $Receive(cut)$ $delivery(m)$ $VT(p)[12] = 10$ $CI(p) \leftarrow CI(p) \cup (12,10)$
r_1. $Receive(begin)$ $delivery(m)$ $VT(p)[7] = 1$ $CI(p) \leftarrow CI(p) \cup (7,1)$ $ActP \leftarrow ActP \cup \{7\}$	r_1. $Receive(begin)$ $delivery(m)$ $VT(p)[7] = 1$ $CI(p) \leftarrow CI(p) \cup (7,1)$ $CI(p) \leftarrow CI(p) \backslash (8,1)$ $ActP \leftarrow ActP \cup \{7\}$	r_1. $Receive(begin)$ $delivery(m)$ $VT(p)[7] = 1$ $CI(p) \leftarrow CI(p) \cup (7,1)$ $ActP \leftarrow ActP \cup \{7\}$
w_9. $Receive(cut)$ $delivery(m)$ $VT(p)[6] = 9$ $CI(p) \leftarrow CI(p) \cup (6,9)$	w_9. $Receive(cut)$ $delivery(m)$ $VT(p)[6] = 9$ $CI(p) \leftarrow CI(p) \cup (6,9)$	

Tabla A.2: Continuación de la prueba del escenario de la figura A.2, subgrupo G_1.

Continuación de la prueba del escenario de la figura A.2, subgrupo G_1		
p_4	p_5	p_6
	v_1. $SendContinuous(begin)$ $VT(p)[5] = 1$ $Act = 1$ $CI(p) \leftarrow (CI(p) \backslash (12, 10))$ $\cup (12, 13)$ $CI(p) \leftarrow (CI(p) \backslash (1, 9)) \cup (1, 14)$ $last\_fifo(p) \leftarrow \emptyset$ $Slaves \leftarrow \emptyset$ $level = 0$ $Master60$ $level = 1$ $H(m) \leftarrow H(m) \cup (6, 9)$ $Slaves \leftarrow \emptyset$ $CI(p) \leftarrow \emptyset$ $m = (5, 1, 6, begin, \{(6, 9)\}, data)$ $sending(m)$	
x_{16}. $Receive(cut)$ $delivery(m)$ $VT(p)[1] = 16$ $CI(p) \leftarrow CI(p) \backslash (1, 9)$ $CI(p) \leftarrow CI(p) \cup (1, 16)$ $CI(p) \leftarrow CI(p) \backslash (2, 1)$	x_{16}. $Receive(cut)$ $delivery(m)$ $VT(p)[1] = 16$ $CI(p) \leftarrow CI(p) \cup (1, 16)$	x_{16}. $Receive(cut)$ $delivery(m)$ $VT(p)[1] = 16$ $CI(p) \leftarrow CI(p) \cup (1, 16)$
p_1. $Receive(begin)$ $delivery(m)$ $VT(p)[10] = 1$ $CI(p) \leftarrow CI(p) \cup (10, 1)$ $CI(p) \backslash CI(p) \cup (12, 10)$ $ActP \leftarrow ActP \cup \{10\}$	p_1. $Receive(begin)$ $delivery(m)$ $VT(p)[10] = 1$ $CI(p) \leftarrow CI(p) \cup (10, 1)$ $ActP \leftarrow ActP \cup \{10\}$	p_1. $Receive(begin)$ $delivery(m)$ $VT(p)[10] = 1$ $CI(p) \leftarrow CI(p) \cup (10, 1)$ $CI(p) \backslash CI(p) \cup (12, 10)$ $ActP \leftarrow ActP \cup \{10\}$
v_1. $Receive(begin)$ $delivery(m)$ $VT(p)[5] = 1$ $CI(p) \leftarrow CI(p) \cup (5, 1)$ $ActP \leftarrow ActP \cup \{5\}$		v_1. $Receive(begin)$ $delivery(m)$ $VT(p)[5] = 1$ $CI(p) \leftarrow CI(p) \cup (5, 1)$ $ActP \leftarrow ActP \cup \{5\}$ $Slaves \leftarrow Slaves \cup \{5\}$

Tabla A.3: Prueba del escenario de la figura A.2, subgrupo G_2.

Prueba del escenario de la figura A.2, subgrupo G_2		
p_7	p_8	p_9
u_1. $Receive(begin)$	u_1. $Receive(begin)$	u_1. $Receive(begin)$
$delivery(m)$	$delivery(m)$	$delivery(m)$
$VT(p)[4] = 1$	$VT(p)[4] = 1$	$VT(p)[4] = 1$
$CI(p) \leftarrow CI(p) \cup (4, 1)$	$CI(p) \leftarrow CI(p) \cup (4, 1)$	$CI(p) \leftarrow CI(p) \cup (4, 1)$
$ActP \leftarrow ActP \cup \{4\}$	$ActP \leftarrow ActP \cup \{4\}$	$ActP \leftarrow ActP \cup \{4\}$
q_1. $Receive(begin)$	q_1. $Receive(begin)$	q_1. $Receive(begin)$
$delivery(m)$	$delivery(m)$	$delivery(m)$
$VT(p)[12] = 1$	$VT(p)[12] = 1$	$VT(p)[12] = 1$
$CI(p) \leftarrow CI(p) \cup (12, 1)$	$CI(p) \leftarrow CI(p) \cup (12, 1)$	$CI(p) \leftarrow CI(p) \cup (12, 1)$
$ActP \leftarrow ActP \cup \{12\}$	$ActP \leftarrow ActP \cup \{12\}$	$ActP \leftarrow ActP \cup \{12\}$
x_1. $Receive(begin)$	x_1. $Receive(begin)$	x_1. $Receive(begin)$
$delivery(m)$	$delivery(m)$	$delivery(m)$
$VT(p)[1] = 1$	$VT(p)[1] = 1$	$VT(p)[1] = 1$
$CI(p) \leftarrow CI(p) \cup (1, 1)$	$CI(p) \leftarrow CI(p) \cup (1, 1)$	$CI(p) \leftarrow CI(p) \cup (1, 1)$
$ActP \leftarrow ActP \cup \{1\}$	$ActP \leftarrow ActP \cup \{1\}$	$ActP \leftarrow ActP \cup \{1\}$
	s_1. $SendContinuous(begin)$	
	$VT(p)[8] = 1$	
	$Act = 1$	
	$last\_fifo(p) \leftarrow \emptyset$	
	$Slaves \leftarrow \emptyset$	
	$level = 0$	
	$Master = 0$	
	$level = 1$	
	$Master = 12$	
	$level = 2$	
	$H(m) \leftarrow H(m) \cup (12, 1)$	
	$Slaves \leftarrow \emptyset$	
	$CI(p) \leftarrow \emptyset$	
	$m = (8, 1, 12, begin, \{(12, 1)\},$	
	$data)$	
	$sending(m)$	

Tabla A.3: Continuación de la prueba del escenario de la figura A.2, subgrupo G_2.

Continuación de la prueba del escenario de la figura A.2, subgrupo G_2		
p_7	p_8	p_9
s_1. $Receive(begin)$ $delivery(m)$ $VT(p)[8] = 1$ $CI(p) \leftarrow CI(p) \cup (8,1)$ $VT(p)[8] = 1$ $CI(p) \leftarrow CI(p) \backslash (12,1)$ $ActP \leftarrow ActP \cup \{8\}$		s_1. $Receive(begin)$ $delivery(m)$ $VT(p)[8] = 1$ $CI(p) \leftarrow CI(p) \cup (8,1)$ $VT(p)[8] = 1$ $CI(p) \leftarrow CI(p) \backslash (12,1)$ $ActP \leftarrow ActP \cup \{8\}$
r_1. $SendContinuous(begin)$ $VT(p)[7] = 1$ $Act = 1$ $CI(p) \leftarrow (CI(p) \backslash (4,1)) \cup (4,5)$ $CI(p) \leftarrow (CI(p) \backslash (1,1)) \cup (1,3)$ $CI(p) \leftarrow (CI(p) \backslash (8,1)) \cup (8,2)$ $last\_fifo(p) \leftarrow \emptyset$ $Slaves \leftarrow \emptyset$ $level = 0$ $Master = 8$ $level = 1$ $H(m) \leftarrow H(m) \cup (8,2)$ $Slaves \leftarrow \emptyset$ $CI(p) \leftarrow \emptyset$ $m = (7,1,8,begin,\{(8,2)\},data)$ $sending(m)$		
o_1. $Receive(discrete)$ $delivery(m)$ $VT(p)[10] = 1$ $CI(p) \leftarrow CI(p) \cup (10,1)$	o_1. $Receive(discrete)$ $delivery(m)$ $VT(p)[10] = 1$ $CI(p) \leftarrow CI(p) \cup (10,1)$	o_1. $Receive(discrete)$ $delivery(m)$ $VT(p)[10] = 1$ $CI(p) \leftarrow CI(p) \cup (10,1)$
z_1. $Receive(discrete)$ $delivery(m)$ $VT(p)[3] = 1$ $CI(p) \leftarrow CI(p) \cup (3,1)$ $CI(p) \leftarrow CI(p) \backslash (1,1)$	z_1. $Receive(discrete)$ $delivery(m)$ $VT(p)[3] = 1$ $CI(p) \leftarrow CI(p) \cup (3,1)$	z_1. $Receive(discrete)$ $delivery(m)$ $VT(p)[3] = 1$ $CI(p) \leftarrow CI(p) \cup (3,1)$ $CI(p) \leftarrow CI(p) \backslash (1,1)$

Tabla A.3: Continuación de la prueba del escenario de la figura A.2, subgrupo G_2.

Continuación de la prueba del escenario de la figura A.2, subgrupo G_2		
p_7	p_8	p_9
w_1. $Receive(begin)$ $delivery(m)$ $VT(p)[6] = 1$ $CI(p) \leftarrow CI(p) \cup (6,1)$ $ActP \leftarrow ActP \cup \{6\}$	w_1. $Receive(begin)$ $delivery(m)$ $VT(p)[6] = 1$ $CI(p) \leftarrow CI(p) \cup (6,1)$ $ActP \leftarrow ActP \cup \{6\}$	w_1. $Receive(begin)$ $delivery(m)$ $VT(p)[6] = 1$ $CI(p) \leftarrow CI(p) \cup (6,1)$ $CI(p) \leftarrow CI(p) \backslash (4,1)$ $ActP \leftarrow ActP \cup \{6\}$
	r_1. $Receive(begin)$ $delivery(m)$ $VT(p)[7] = 1$ $CI(p) \leftarrow CI(p) \cup (7,1)$ $ActP \leftarrow ActP \cup \{7\}$ $Slaves \leftarrow Slaves \cup \{7\}$	r_1. $Receive(begin)$ $delivery(m)$ $VT(p)[7] = 1$ $CI(p) \leftarrow CI(p) \cup (7,1)$ $ActP \leftarrow ActP \cup \{7\}$
q_{10}. $Receive(cut)$ $delivery(m)$ $VT(p)[12] = 10$ $CI(p) \leftarrow CI(p) \cup (12,10)$ $CI(p) \leftarrow CI(p) \backslash (10,1)$	q_{10}. $Receive(cut)$ $delivery(m)$ $VT(p)[12] = 10$ $CI(p) \leftarrow CI(p) \cup (12,10)$ $CI(p) \leftarrow CI(p) \backslash (10,1)$	q_{10}. $Receive(cut)$ $delivery(m)$ $VT(p)[12] = 10$ $CI(p) \leftarrow CI(p) \cup (12,10)$ $CI(p) \leftarrow CI(p) \backslash (10,1)$
u_{11}. $Receive(end)$ $delivery(m)$ $VT(p)[4] = 11$ $CI(p) \leftarrow CI(p) \cup (4,11)$ $ActP \leftarrow ActP \backslash \{4\}$	u_{11}. $Receive(end)$ $delivery(m)$ $VT(p)[4] = 11$ $CI(p) \leftarrow CI(p) \cup (4,11)$ $ActP \leftarrow ActP \backslash \{4\}$	u_{11}. $Receive(end)$ $delivery(m)$ $VT(p)[4] = 11$ $CI(p) \leftarrow CI(p) \cup (4,11)$ $ActP \leftarrow ActP \backslash \{4\}$
y_1. $Receive(begin)$ $delivery(m)$ $VT(p)[2] = 1$ $CI(p) \leftarrow CI(p) \cup (2,1)$ $ActP \leftarrow ActP \cup \{2\}$	y_1. $Receive(begin)$ $delivery(m)$ $VT(p)[2] = 1$ $CI(p) \leftarrow CI(p) \cup (2,1)$ $ActP \leftarrow ActP \cup \{2\}$	y_1. $Receive(begin)$ $delivery(m)$ $VT(p)[2] = 1$ $CI(p) \leftarrow CI(p) \cup (2,1)$ $ActP \leftarrow ActP \cup \{2\}$
x_9. $Receive(cut)$ $delivery(m)$ $VT(p)[1] = 9$ $CI(p) \leftarrow CI(p) \cup (1,9)$ $CI(p) \leftarrow CI(p) \backslash (3,1)$	x_9. $Receive(cut)$ $delivery(m)$ $VT(p)[1] = 9$ $CI(p) \leftarrow CI(p) \cup (1,9)$ $CI(p) \leftarrow CI(p) \backslash (3,1)$	x_9. $Receive(cut)$ $delivery(m)$ $VT(p)[1] = 9$ $CI(p) \leftarrow CI(p) \cup (1,9)$ $CI(p) \leftarrow CI(p) \backslash (3,1)$

Tabla A.3: Continuación de la prueba del escenario de la figura A.2, subgrupo G_2.

Continuación de la prueba del escenario de la figura A.2, subgrupo G_2		
p_7	p_8	p_9
w_9. $Receive(cut)$ $delivery(m)$ $VT(p)[6] = 9$ $CI(p) \leftarrow CI(p) \backslash (6, 1)$ $CI(p) \leftarrow CI(p) \cup (6, 9)$ $CI(p) \leftarrow CI(p) \backslash (4, 11)$	w_9. $Receive(cut)$ $delivery(m)$ $VT(p)[6] = 9$ $CI(p) \leftarrow CI(p) \backslash (6, 1)$ $CI(p) \leftarrow CI(p) \cup (6, 9)$ $CI(p) \leftarrow CI(p) \backslash (4, 11)$	w_9. $Receive(cut)$ $delivery(m)$ $VT(p)[6] = 9$ $CI(p) \leftarrow CI(p) \backslash (6, 1)$ $CI(p) \leftarrow CI(p) \cup (6, 9)$ $CI(p) \leftarrow CI(p) \backslash (4, 11)$
p_1. $Receive(begin)$ $delivery(m)$ $VT(p)[11] = 1$ $CI(p) \leftarrow CI(p) \cup (11, 1)$ $CI(p) \leftarrow CI(p) \backslash (12, 10)$ $ActP \leftarrow ActP \cup \{11\}$	p_1. $Receive(begin)$ $delivery(m)$ $VT(p)[11] = 1$ $CI(p) \leftarrow CI(p) \cup (11, 1)$ $ActP \leftarrow ActP \cup \{11\}$	p_1. $Receive(begin)$ $delivery(m)$ $VT(p)[11] = 1$ $CI(p) \leftarrow CI(p) \cup (11, 1)$ $CI(p) \leftarrow CI(p) \backslash (12, 10)$ $ActP \leftarrow ActP \cup \{11\}$
x_{16}. $Receive(cut)$ $delivery(m)$ $VT(p)[1] = 16$ $CI(p) \leftarrow CI(p) \backslash (1, 9)$ $CI(p) \leftarrow CI(p) \cup (1, 16)$ $CI(p) \leftarrow CI(p) \backslash (4, 11)$ $CI(p) \leftarrow CI(p) \backslash (2, 1)$	x_{16}. $Receive(cut)$ $delivery(m)$ $VT(p)[1] = 16$ $CI(p) \leftarrow CI(p) \backslash (1, 9)$ $CI(p) \leftarrow CI(p) \cup (1, 16)$ $CI(p) \leftarrow CI(p) \backslash (4, 11)$ $CI(p) \leftarrow CI(p) \backslash (2, 1)$	x_{16}. $Receive(cut)$ $delivery(m)$ $VT(p)[1] = 16$ $CI(p) \leftarrow CI(p) \backslash (1, 9)$ $CI(p) \leftarrow CI(p) \cup (1, 16)$ $CI(p) \leftarrow CI(p) \backslash (4, 11)$ $CI(p) \leftarrow CI(p) \backslash (2, 1)$

Tabla A.4: Prueba del escenario de la figura A.2, subgrupo G_3.

Prueba del escenario de la figura A.2, subgrupo G_3		
p_{10}	p_{11}	p_{12}
		$q_1.\ SendContinuous(begin)$ $VT(p)[12] = 1$ $Act = 1$ $last\_fifo(p) \leftarrow \emptyset$ $Slaves \leftarrow \emptyset$ $level = 0$ $Master = 0$ $level = 1$ $Master = 0$ $level = 2$ $Master = 0$ $level = 3$ $H(m) \leftarrow CI(p)$ $m = (12, 1, 0, begin, \emptyset, data)$ $sending(m)$
$u_1.\ Receive(begin)$ $delivery(m)$ $VT(p)[4] = 1$ $CI(p) \leftarrow CI(p) \cup (4, 1)$ $ActP \leftarrow ActP \cup \{4\}$	$u_1.\ Receive(begin)$ $delivery(m)$ $VT(p)[4] = 1$ $CI(p) \leftarrow CI(p) \cup (4, 1)$ $ActP \leftarrow ActP \cup \{4\}$	$u_1.\ Receive(begin)$ $delivery(m)$ $VT(p)[4] = 1$ $CI(p) \leftarrow CI(p) \cup (4, 1)$ $ActP \leftarrow ActP \cup \{4\}$
$q_1.\ Receive(begin)$ $delivery(m)$ $VT(p)[12] = 1$ $CI(p) \leftarrow CI(p) \cup (12, 1)$ $ActP \leftarrow ActP \cup \{12\}$	$q_1.\ Receive(begin)$ $delivery(m)$ $VT(p)[12] = 1$ $CI(p) \leftarrow CI(p) \cup (12, 1)$ $ActP \leftarrow ActP \cup \{12\}$	$q_1.\ Receive(begin)$ $delivery(m)$ $VT(p)[12] = 1$ $CI(p) \leftarrow CI(p) \cup (12, 1)$ $ActP \leftarrow ActP \cup \{12\}$
$x_1.\ Receive(begin)$ $delivery(m)$ $VT(p)[1] = 1$ $CI(p) \leftarrow CI(p) \cup (1, 1)$ $ActP \leftarrow ActP \cup \{1\}$	$x_1.\ Receive(begin)$ $delivery(m)$ $VT(p)[1] = 1$ $CI(p) \leftarrow CI(p) \cup (1, 1)$ $ActP \leftarrow ActP \cup \{1\}$	$x_1.\ Receive(begin)$ $delivery(m)$ $VT(p)[1] = 1$ $CI(p) \leftarrow CI(p) \cup (1, 1)$ $ActP \leftarrow ActP \cup \{1\}$ $Master = 1$

Tabla A.4: Continuación de la prueba del escenario de la figura A.2, subgrupo G_3.

Continuación de la prueba del escenario de la figura A.2, subgrupo G_3		
p_{10}	p_{11}	p_{12}
o_1. $SendDiscrete(discrete)$ $VT(p)[10] = 2$ $CI(p) \leftarrow (CI(p) \backslash (12,1)) \cup (12,2)$ $Master = 12$ $H(m) \leftarrow \{(12,2)\}$ $last\_fifo(p) \leftarrow \emptyset$ $CI(p) \leftarrow \emptyset$ $m = (10, 1, 12, discrete, \{(12,2)\},$ $data)$ $sending(m)$		
s_1. $Receive(begin)$ $delivery(m)$ $VT(p)[8] = 1$ $CI(p) \leftarrow CI(p) \cup (8,1)$ $ActP \leftarrow ActP \cup \{8\}$	s_1. $Receive(begin)$ $delivery(m)$ $VT(p)[8] = 1$ $CI(p) \leftarrow CI(p) \cup (8,1)$ $ActP \leftarrow ActP \cup \{8\}$	s_1. $Receive(begin)$ $delivery(m)$ $VT(p)[8] = 1$ $CI(p) \leftarrow CI(p) \cup (8,1)$ $ActP \leftarrow ActP \cup \{8\}$ $Slaves \leftarrow Slaves \cup \{8\}$
	o_1. $Receive(discrete)$ $delivery(m)$ $VT(p)[10] = 1$ $CI(p) \leftarrow CI(p) \cup (10,1)$	o_1. $Receive(discrete)$ $delivery(m)$ $VT(p)[10] = 1$ $CI(p) \leftarrow CI(p) \cup (10,1)$ $ActP \leftarrow ActP \backslash \{10\}$ $Slaves \leftarrow Slaves \cup \{10\}$ $SendContinuous(cut)$
		q_{10}. $SendContinuous(cut)$ $VT(p)[12] = 10$ $H(m) \leftarrow H(m) \cup (1,1)$ $H(m) \leftarrow H(m) \cup (10,1)$ $Slaves \leftarrow \emptyset$ $CI(p) \leftarrow \emptyset$ $m = (12, 10, 1, cut, \{(1,1),(10,1)\},$ $data)$ $sending(m)$

Tabla A.4: Continuación de la prueba del escenario de la figura A.2, subgrupo G_3.

Continuación de la prueba del escenario de la figura A.2, subgrupo G_3		
p_{10}	p_{11}	p_{12}
w_1. $Receive(begin)$ $delivery(m)$ $VT(p)[6] = 1$ $CI(p) \leftarrow CI(p) \cup (6,1)$ $ActP \leftarrow ActP \cup \{6\}$	w_1. $Receive(begin)$ $delivery(m)$ $VT(p)[6] = 1$ $CI(p) \leftarrow CI(p) \cup (6,1)$ $CI(p) \leftarrow CI(p) \backslash (4,1)$ $ActP \leftarrow ActP \cup \{6\}$	w_1. $Receive(begin)$ $delivery(m)$ $VT(p)[6] = 1$ $CI(p) \leftarrow CI(p) \cup (6,1)$ $ActP \leftarrow ActP \cup \{6\}$
z_1. $Receive(discrete)$ $delivery(m)$ $VT(p)[3] = 1$ $CI(p) \leftarrow CI(p) \cup (3,1)$	z_1. $Receive(discrete)$ $delivery(m)$ $VT(p)[3] = 1$ $CI(p) \leftarrow CI(p) \cup (3,1)$ $CI(p) \leftarrow CI(p) \backslash (1,1)$	z_1. $Receive(discrete)$ $delivery(m)$ $VT(p)[3] = 1$ $CI(p) \leftarrow CI(p) \cup (3,1)$
q_{10}. $Receive(cut)$ $delivery(m)$ $VT(p)[12] = 10$ $CI(p) \leftarrow CI(p) \cup (12,10)$	q_{10}. $Receive(cut)$ $delivery(m)$ $VT(p)[12] = 10$ $CI(p) \leftarrow CI(p) \backslash (12,1)$ $CI(p) \leftarrow CI(p) \cup (12,10)$ $CI(p) \leftarrow CI(p) \backslash (10,1)$	
	p_1. $SendContinuous(begin)$ $VT(p)[11] = 1$ $Act = 1$ $CI(p) \leftarrow (CI(p) \backslash (8,1)) \cup (8,5)$ $CI(p) \leftarrow (CI(p) \backslash (6,1)) \cup (6,2)$ $last\_fifo(p) \leftarrow \emptyset$ $Slaves \leftarrow \emptyset$ $level = 0$ $Master = 0$ $level = 1$ $Master = 12$ $level = 2$ $H(m) \leftarrow H(m) \cup (12,10)$ $Slaves \leftarrow \emptyset$ $CI(p) \leftarrow \emptyset$ $m = (11, 1, 12, begin, \{(12,10)\},$ $data)$ $sending(m)$	

Tabla A.4: Continuación de la prueba del escenario de la figura A.2, subgrupo G_3.

Continuación de la prueba del escenario de la figura A.2, subgrupo G_3		
p_{10}	p_{11}	p_{12}
r_1. $Receive(begin)$ $delivery(m)$ $VT(p)[7] = 1$ $CI(p) \leftarrow CI(p) \cup (7,1)$ $CI(p) \leftarrow CI(p) \backslash (8,1)$ $ActP \leftarrow ActP \cup \{7\}$	r_1. $Receive(begin)$ $delivery(m)$ $VT(p)[7] = 1$ $CI(p) \leftarrow CI(p) \cup (7,1)$ $ActP \leftarrow ActP \cup \{7\}$	r_1. $Receive(begin)$ $delivery(m)$ $VT(p)[7] = 1$ $CI(p) \leftarrow CI(p) \cup (7,1)$ $ActP \leftarrow ActP \cup \{7\}$
y_1. $Receive(begin)$ $delivery(m)$ $VT(p)[2] = 1$ $CI(p) \leftarrow CI(p) \cup (2,1)$ $ActP \leftarrow ActP \cup \{2\}$	y_1. $Receive(begin)$ $delivery(m)$ $VT(p)[2] = 1$ $CI(p) \leftarrow CI(p) \cup (2,1)$ $ActP \leftarrow ActP \cup \{2\}$	y_1. $Receive(begin)$ $delivery(m)$ $VT(p)[2] = 1$ $CI(p) \leftarrow CI(p) \cup (2,1)$ $ActP \leftarrow ActP \cup \{2\}$
x_9. $Receive(cut)$ $delivery(m)$ $VT(p)[1] = 9$ $CI(p) \leftarrow CI(p) \cup (1,9)$ $CI(p) \leftarrow CI(p) \backslash (3,1)$	x_9. $Receive(cut)$ $delivery(m)$ $VT(p)[1] = 9$ $CI(p) \leftarrow CI(p) \cup (1,9)$	x_9. $Receive(cut)$ $delivery(m)$ $VT(p)[1] = 9$ $CI(p) \leftarrow CI(p) \cup (1,9)$ $CI(p) \leftarrow CI(p) \backslash (3,1)$
u_{11}. $Receive(begin)$ $delivery(m)$ $VT(p)[4] = 11$ $CI(p) \leftarrow CI(p) \cup (4,11)$ $CI(p) \leftarrow CI(p) \backslash (6,1)$ $ActP \leftarrow ActP \backslash \{4\}$	u_{11}. $Receive(begin)$ $delivery(m)$ $VT(p)[4] = 11$ $CI(p) \leftarrow CI(p) \cup (4,11)$ $ActP \leftarrow ActP \backslash \{4\}$	u_{11}. $Receive(begin)$ $delivery(m)$ $VT(p)[4] = 11$ $CI(p) \leftarrow CI(p) \cup (4,11)$ $CI(p) \leftarrow CI(p) \backslash (6,1)$ $ActP \leftarrow ActP \backslash \{4\}$
p_1. $Receive(begin)$ $delivery(m)$ $VT(p)[11] = 1$ $CI(p) \leftarrow CI(p) \cup (11,1)$ $ActP \leftarrow ActP \cup \{11\}$		p_1. $Receive(begin)$ $delivery(m)$ $VT(p)[11] = 1$ $CI(p) \leftarrow CI(p) \cup (11,1)$ $ActP \leftarrow ActP \cup \{11\}$ $Slaves \leftarrow Slaves \cup \{11\}$
w_9. $Receive(cut)$ $delivery(m)$ $VT(p)[6] = 9$ $CI(p) \leftarrow CI(p) \cup (6,9)$ $CI(p) \leftarrow CI(p) \backslash (4,11)$	w_9. $Receive(cut)$ $delivery(m)$ $VT(p)[6] = 9$ $CI(p) \leftarrow CI(p) \cup (6,9)$ $CI(p) \leftarrow CI(p) \backslash (4,11)$	w_9. $Receive(cut)$ $delivery(m)$ $VT(p)[6] = 9$ $CI(p) \leftarrow CI(p) \cup (6,9)$ $CI(p) \leftarrow CI(p) \backslash (4,11)$

Tabla A.4: Continuación de la prueba del escenario de la figura A.2, subgrupo G_3.

Continuación de la prueba del escenario de la figura A.2, subgrupo G_3		
p_{10}	p_{11}	p_{12}
x_{16}. $Receive(cut)$ $delivery(m)$ $VT(p)[1] = 16$ $CI(p) \leftarrow CI(p) \backslash (1, 9)$ $CI(p) \leftarrow CI(p) \cup (1, 16)$ $CI(p) \leftarrow CI(p) \backslash (2, 1)$	x_{16}. $Receive(cut)$ $delivery(m)$ $VT(p)[1] = 16$ $CI(p) \leftarrow CI(p) \backslash (1, 9)$ $CI(p) \leftarrow CI(p) \cup (1, 16)$ $CI(p) \leftarrow CI(p) \backslash (2, 1)$	x_{16}. $Receive(cut)$ $delivery(m)$ $VT(p)[1] = 16$ $CI(p) \leftarrow CI(p) \backslash (1, 9)$ $CI(p) \leftarrow CI(p) \cup (1, 16)$ $CI(p) \leftarrow CI(p) \backslash (2, 1)$
		x_{17}. $Receive(end)$ $delivery(m)$ $VT(p)[1] = 17$ $CI(p) \leftarrow CI(p) \backslash (1, 16)$ $CI(p) \leftarrow CI(p) \cup (1, 17)$ $ActP \leftarrow ActP \backslash \{1\}$ $Master = 0$ $SendContinuous(cut)$
		q_{22}. $SendContinuous(cut)$ $VT(p)[12] = 22$ $H(m) \leftarrow H(m) \cup (11, 1)$ $H(m) \leftarrow H(m) \cup (1, 17)$ $Slaves \leftarrow \emptyset$ $CI(p) \leftarrow \emptyset$ $m = (12, 22, 0, cut, \{(11, 1),$ $(1, 17)\}, data)$ $sending(m)$