INSTITUTO TECNOLÓGICO SUPERIOR DE MISANTLA

# "MECANISMO CAUSAL DIFUSO PARA GENERAR SNAPSHOT EN SISTEMAS DE CÓMPUTO MÓVIL"

## T E S I S

PARA OBTENER EL GRADO DE

## MAESTRO EN SISTEMAS COMPUTACIONALES

P R E S E N T A


## HÉCTOR EMMANUEL TEJEDA RODRÍGUEZ

ASESORES: DR. LUIS ALBERTO MORALES ROSALES

DR. IGNACIO ALGREDO BADILLO


MISANTLA, VERACRUZ

MAYO, 2015

# Resumen

El sistema de cómputo móvil es un tipo de sistema distribuido en donde la mayoría de los procesos se ejecutan sobre dispositivos móviles. Gracias a la movilidad que tienen dichos dispositivos, los usuarios pueden acceder a su información en cualquier lugar y en cualquier momento. No obstante, las aplicaciones en un ambiente distribuido usualmente son susceptibles a presentar fallas que causen inconsistencias o pérdidas de información que el sistema ha generado.

Los sistemas de cómputo móvil utilizan los mecanismos para generar *snapshot* o instantáneas que almacenan el estado del sistema con el objetivo de preservar la mayor cantidad de información posible. Sin embargo, los mecanismos tradicionales no son suficientes ni aplicables para llevar a cabo dicha tarea debido a la movilidad y a las capacidades limitadas que tienen los dispositivos móviles.

Es por eso, que este trabajo presenta como se desarrolló un mecanismo asíncrono que permite generar *snapshot* distribuido en sistemas de cómputo móvil. El mecanismo está basado en relaciones causales difusas con las cuales se miden el tráfico que existe en la red y las capacidades que tiene cada dispositivo móvil. El conjunto de dichas relaciones le permiten al sistema observar el comportamiento *actual* del estado del sistema.

El mecanismo evalúa a través de un control difuso dicho estado y la calidad de servicio para tomar la decisión de si el sistema tiene o no que generar el *snapshot* distribuido. Este procedimiento lo llevan a cabo las estaciones de soporte móvil, por lo que también son las responsables de enviar las peticiones para generar los *snapshot* así como el almacenamiento y resguardo de los mismos que se generen durante el funcionamiento del sistema.

Finalmente, la aplicación del mecanismo demostró que fue posible generar *snapshot* distribuido en sistemas de cómputo móvil sin tener que llevar a cabo una coordinación entre dispositivos y sin recurrir a una generación de *snapshot* de manera periódica, tal cual como lo hacen los mecanismos tradicionales. Asimismo, se demostró que el *overhead* generado por este mecanismo es menor en comparación con otros trabajos que se han desarrollado.

# Índice de contenido

# Índice de figuras

## Nomenclatura

| | |
|---|---|
| MH | Dispositivo Móvil (*Mobile Host*) |
| MSS | Estación de Soporte Móvil (*Mobile Support Station*) |
| SH | Dispositivo fijo (*Static Host*) |
| FCR | Relación Causal Difusa (*Fuzzy Causal Relation*) |
| DR | Relación de Distancia (*Distance Relation*) |
| FCC | Consistencia Causal Difusa (*Fuzzy Causal Consistency*) |
| GP | Grado de Ponderación |
| TD | Distancia Temporal (*Temporal Distance*) |
| CD | Distancia Causal (*Causal Distance*) |
| SD | Distancia Espacial (*Spatial Distance*) |
| GPN | Grado de Ponderación de la Red |
| GPM | Grado de Ponderación de los Móviles |
| NC | Condiciones de la Red (*Network Conditions*) |
| MC | Condiciones de los Móviles (*Mobile Conditions*) |

# CAPITULO I GENERALIDADES

## 1.1 Introducción

Un sistema distribuido es una colección de distintos procesos los cuales están espacialmente separados y se comunican entre ellos a través del intercambio o paso de mensajes sin que exista una memoria compartida entre ellos ni un reloj global que sincronice la comunicación.

Los recientes avances en computadoras portátiles con interfaces de comunicación inalámbrica y servicios satelitales han hecho posible crear aplicaciones distribuidas que pueden obtener acceso a información en cualquier lugar y en cualquier momento para los usuarios en movimiento. A este ambiente de computación se le llama entorno de cómputo móvil distribuido.

Un sistema de cómputo móvil es un sistema distribuido donde algunos de los procesos se ejecutan sobre dispositivos móviles. Este sistema consiste de dispositivos móviles (*Mobile Hosts, MH*), de estaciones de soporte móvil (*Mobile Support Stations, MSS*) y en algunas ocasiones pueden existir dispositivos fijos (*Static Hosts, SH*).

Los sistemas de cómputo móvil utilizan los *snapshot* para salvaguardar el trabajo que hayan realizado. El *snapshot* es un estado local de un proceso que se guarda en un almacenamiento estable o fijo y que permite la reconstrucción del sistema en cualquier tiempo. Los *snapshot* son de utilidad para diferentes propósitos entre los cuales destacan la tolerancia a fallas, la depuración de programas y la migración de procesos.

En un sistema distribuido dada sus características, el *snapshot* distribuido es el conjunto de *snapshot* locales de cada proceso de tal manera que al final sea visto como un estado global [1]. Para que el *snapshot* distribuido sea consistente o estable debe ser generado conforme a la ocurrencia de los eventos sin contener mensajes huérfanos, es decir, mensajes cuyo evento de recepción es parte del *snapshot* pero su evento de envío no lo es.

Sin embargo, un sistema distribuido no está exento de presentar problemas durante el transcurso del tiempo, es por eso que este debe contar con un mecanismo que le permita al sistema distribuido ser capaz de hacer una restauración a un estado consistente tan cercano como sea posible al último

trabajo computacional que haya realizado. El mecanismo puede estar basado en cuatro métodos para generar un *snapshot* distribuido, estos son:

- Síncrono, en el cual se asume que un proceso hace la petición hacia los otros procesos para realizar la construcción de sus *snapshot* de manera coordinada y que durante ese tiempo no exista o no ocurra alguna falla [2].
- Asíncrono, donde los procesos realizan la construcción de sus *snapshot* de manera independiente permitiendo la máxima autonomía para que cada proceso decida cuándo debe construir sus *snapshot* [3].
- Cuasi-síncrono, es una combinación de los dos anteriores por lo que al ser un método hibrido los *snapshot* se generan tanto de manera independiente como de manera coordinada [4].
- Registro, cada proceso guarda en el almacenamiento fijo cada uno de los mensajes que ha recibido de otros procesos por lo que si un proceso falla otro nuevo lo reemplaza para mantener la consistencia del sistema [4].

El problema de los mecanismos actuales al utilizar alguno de estos métodos, es que sólo toman en cuenta algunos factores de la red para generar el estado global o bien generan a los *snapshot* de manera periódica. Por lo que al no tomar en cuenta las condiciones del tráfico de la red y de los recursos que disponen los dispositivos móviles, los mecanismos generan mayor carga de trabajo de la que genera el propio sistema.

También, dichos mecanismos hacen un uso indiscriminado de los recursos móviles porque generan *snapshot* innecesarios cuando el estado del sistema se mantiene estable o cuando no existe algún cambio dentro de él. Por lo tanto, este trabajo presenta el desarrollo de un mecanismo asíncrono que determina cuándo el sistema de cómputo móvil debe tomar la decisión de generar el *snapshot* distribuido sin afectar su rendimiento.

## 1.2 Problemática

Los *snapshot* son técnicas eficientes en los sistemas distribuidos para diferentes propósitos como la tolerancia a fallas, la depuración de programas y la migración de procesos, los mecanismos tradicionales no son suficientes ni aplicables para generar *snapshot* cuando se trata de un sistema distribuido en tiempo real y que consta de dispositivos móviles,

Lo anterior se debe a que el entorno del cómputo móvil introduce nuevos cambios que provienen de las capacidades limitadas (almacenamiento y fuente de energía), la susceptibilidad a los daños físicos y las conexiones inalámbricas (más lentas que las conexiones alámbricas por el poco ancho de banda que presentan las redes celulares). Asimismo, las frecuentes conexiones y desconexiones debido a la movilidad de los dispositivos móviles así como los costos para localizarlos generan problemas para la construcción de mecanismos de generación de *snapshot* distribuido.

Por lo que dentro de este entorno de computo móvil surgió la siguiente pregunta de investigación: *¿Cuáles son los factores que se necesitan para que el sistema de cómputo móvil determine cuándo debe iniciar la generación del snapshot distribuido?*

Respondiendo a esta pregunta, cada *MSS* debe tomar en cuenta las condiciones de la red sobre las que el sistema está trabajando debido a la poca confiabilidad que existen en los canales de comunicación y a las condiciones de los dispositivos móviles debido a que éstos disponen de una cantidad de recursos limitados de cómputo.

Para las condiciones de la red en el entorno móvil los factores a tomar en cuenta son el ancho de banda disponible, los retrasos entre mensajes, la tasa de perdida de datos y la intensidad de la señal de la *MSS* con el *MH*. Por otra parte, los factores de las condiciones de los móviles se deben a la batería limitada, el almacenamiento fijo disponible, el procesamiento del *CPU* con el que se cuenta, así como también la memoria *RAM*.

Como los algoritmos actuales no toman en cuenta ambas condiciones para generar un *snapshot* distribuido causan problemas en la red como cuellos de botella, bloqueos del sistema y utilización excesiva del almacenamiento de los *MH*. Por lo que determinar un snapshot distribuido consistente con estos problemas no es adecuado para aquellas aplicaciones que sean en tiempo real.

Con estas condiciones necesarias, surgió otra pregunta de investigación. *¿Cada cuándo el sistema debe iniciar el proceso para generar un snapshot distribuido?*

Este problema proviene de los distintos métodos que utilizan los mecanismos para generar un *snapshot* consistente. Si el *snapshot* se construye utilizando el método síncrono sería necesario realizar un bloqueo parcial o total del sistema utilizando mensajes especiales entre los *MH* y las *MSS*, por lo que este método no es recomendable para un sistema distribuido en tiempo real.

Por el contrario, si el *snapshot* se construye utilizando método asíncrono, los procesos generan sus *snapshot* de manera independiente pero puede no existir un *snapshot* consistente. Por lo que se presentaría la restauración de varios *snapshot* (uno tras otro) siendo posible el peor caso que lleve a la restauración total del sistema y llegar al punto inicial del mismo. A esta restauración en cascada se le conoce como efecto dominó.

Combinar ambos métodos (método cuasi-asíncrono) evita el efecto dominó porque no se requeriría que todos los procesos estén completamente coordinados. Sin embargo, este método genera *snapshot* locales que no forman parte de un *snapshot* distribuido y terminan siendo no útiles (*inútiles*) para el sistema. Por lo que al ser guardados en el almacenamiento estable hacen un uso innecesario de este recurso.

Sin en cambio, si los procesos llevan un registro de los mensajes (método de registro), la generación del *snapshot* dependerá del número de los procesos que estén disponibles. Ya que si el número de los procesos que fallan es mayor a este número, al sistema le tomaría un tiempo considerable completar la restauración. Por lo tanto este método también se vuelve inadecuado para sistemas distribuidos en tiempo real.

Un punto a considerar es que, tanto en el método asíncrono como en el asíncrono, los mecanismos se ven forzados a realizar una limpieza periódica de los *snapshot* no útiles. Otro punto a considerar de los métodos síncrono, asíncrono y cuasi-síncrono es que la información para generar el *snapshot* distribuido comúnmente se envía en mensajes por separado de los mensajes de aplicación. Por lo que posiblemente el tránsito de los mensajes se vuelve lento en los canales de comunicación y que además se presenten pérdidas de mensajes que hacen que el *snapshot* distribuido se vuelva inconsistente.

Esto nos indica que cuando los mecanismos construyen los *snapshot* generan mayor carga de trabajo en mensajes de peticiones y respuesta tanto en las *MSS* como para los *MH*. Dicha carga afecta el rendimiento del sistema ya que se le da prioridad a este suceso y no al procesamiento del sistema.

Entonces surgió otra pregunta de investigación, *¿Quién debe iniciar el proceso de construcción del snapshot distribuido para que el mecanismo no genere una mayor carga de trabajo de la que hace el sistema al utilizar alguno de estos métodos?*

Este mecanismo propone utilizar las condiciones de la red y de los dispositivos móviles con el método asíncrono para relajar en cierta forma la condición de cómo se deben generar los *snapshot* distribuidos. Asimismo, se propone utilizar las relaciones causales difusas para que las *MSS* puedan revisar internamente o a nivel de celda el estado del sistema para determinar si se está pasando de un estado consistente a uno inconsistente y envíen a los *MH* la petición de generar el *snapshot* distribuido.

Si cada *MSS* determina un *snapshot* parcial del sistema, el mecanismo generaría a partir de cada *MSS* que conforma la red móvil un *snapshot* distribuido del sistema y consistente para que se disminuya la pérdida de trabajo computacional y se evita causar el efecto dominó al momento de realizar una restauración del sistema.

## 1.3 Objetivos

### 1.3.1 General

Desarrollar un mecanismo asíncrono para la generación de *snapshot* distribuido en sistemas de cómputo móvil basado en relaciones causales difusas.

### 1.3.2 Específicos

- Seleccionar las variables de los sistemas de cómputo móvil para establecer los parámetros representativos de los dominios de la relación causal difusa.
- Desarrollar un algoritmo asíncrono para generar *snapshot* que satisfaga un orden de entrega causal difuso de los eventos del sistema.
- Diseñar un control difuso para identificar la oscilación del estado del sistema entre consistente e inconsistente y determinar si el sistema debe hacer la petición de generar el *snapshot* distribuido.

## 1.4 Propuesta de Solución

Los esquemas actuales para generar un estado global en un ambiente móvil distribuido lo llevan a cabo de manera periódica de tal manera que la carga de trabajo generada puede llegar a ser mayor a la generada por la aplicación.

El desarrollo de este mecanismo asíncrono hace que la generación de *snapshot* distribuidos para sistemas de cómputo móvil se lleve a cabo en el punto donde el sistema empiece a no tolerar más pérdida de datos de lo que se permite. Para detectar este punto el mecanismo involucró las condiciones de la red y las condiciones de los dispositivos móviles.

Los mensajes que se generan se evalúan durante su recepción en las estaciones de soporte móvil. Las estaciones al utilizar las relaciones causales difusas muestran el grado de afectación que le tienen dichos mensajes al sistema sin hacer un bloqueo parcial o total del mismo. Este grado de afectación refleja el comportamiento del estado actual del sistema y el mecanismo determina si se debe llevar a cabo la generación del *snapshot* distribuido.

Con esta propuesta surgió la siguiente hipótesis, *es posible que a partir del estado del sistema se pueda determinar con las relaciones causales difusas cuándo se debe generar un snapshot en un sistema de cómputo móvil*.

Como el mecanismo toma en cuenta las condiciones del tráfico de la red y los recursos de los dispositivos móviles, éste generaría los *snapshot* distribuidos de manera optimista porque este proceso se haría antes de que el sistema se vuelva inconsistente sin que exista el riesgo de que se presente el efecto dominó. De esta manera, si sucede algún inconveniente habría una disminución considerable de pérdida de información y de trabajo computacional cuando el sistema haga una restauración del mismo.

## 1.5 Justificación

Los mecanismos actuales requieren constantemente de recursos en los dispositivos móviles pues generan *snapshot* de acuerdo a periodos de tiempo o ajustan dichos periodos de acuerdo el estado de la red. Por lo que es necesaria una coordinación entre procesos utilizando mensajes especiales para llevar a cabo esta tarea.

Sin embargo, el uso de estos mensajes origina la existencia de cuellos de botella causando una mala recepción de los mensajes por lo que se pueden presentan inconsistencias en los *snapshot* que se generen. Además, los mecanismos actuales utilizan los recursos de los dispositivos móviles por lo que los *MH* pueden presentar un bajo rendimiento y por lo tanto esto puede causar que el sistema presente algún tipo de bloqueo.

Es por eso que a diferencia de los mecanismos desarrollados en la literatura, este mecanismo no genera *snapshot* distribuido de manera periódica sino que la decisión se toma a partir de las condiciones del estado del sistema y de su calidad de servicio. Con esto el sistema no tiene que ajustar continuamente los periodos de tiempo para llevar a cabo la generación de los *snapshot*. Esto implica que los dispositivos móviles tengan un ahorro de recursos que posiblemente puedan ser utilizados por la aplicación distribuida en un futuro (muy) cercano.

En el mecanismo, las estaciones de soporte móvil se encargan del proceso de generación del *snapshot* distribuido para no tener que realizar bloqueos de manera parcial o total porque dicha tarea se hace de manera asíncrona. Esto permite que el sistema funcione de manera continua sin que se vea afectado el rendimiento del sistema.

Asimismo, no se utilizan mensajes especiales para indicar que el sistema tiene que generar un *snapshot*, sino que las peticiones se sobrellevan en los mensajes de la aplicación ya que con esto el mecanismo evita generar *snapshot* inconsistentes.

Como se permite el uso continuo del sistema, este mecanismo fue desarrollado principalmente para utilizarse en aplicaciones distribuidas multimedia y en tiempo real además de que toleren cierta pérdida de información durante la transmisión de los mensajes. Ejemplos de algunas de estas aplicaciones pueden ser la realidad virtual, la realidad aumentada y los videojuegos.

## 1.6 Metodología

1. Seleccionar las variables en base a las condiciones del tráfico de la red y las condiciones de los dispositivos móviles para establecer los dominios de la relación causal difusa.
2. Diseñar el modelo del entorno móvil distribuido basado en relaciones causales difusas para generar estados del comportamiento de los sistemas de cómputo móvil.
3. Desarrollar el algoritmo basado en el modelo anterior para evaluar los estados a través del control difuso y determinar la construcción del *snapshot* distribuido.
4. Validación del algoritmo mediante estudio de casos de los principales problemas de inconsistencia al generar *snapshot*.

## 1.7 Alcances

La presente propuesta tiene como alcances aquellos sistemas distribuidos móviles cuyas aplicaciones se encuentre dentro de la clase *conversacional* establecida por el *3GPP*, ya que éstas



soportan pérdida de datos y retardos entre mensajes aceptables [5, 6]. Otro alcance es que el mecanismo está destinado para sistemas distribuidos móviles de un sólo grupo.

## 1.8 Limitaciones

Este mecanismo no está destinado para aplicaciones distribuidas que se encuentren fuera de la clase *conversacional* establecida por el *3GPP*, tampoco se puede aplicar para redes móviles del tipo ad hoc así como para sistemas distribuidos móviles del tipo multigrupo.



# CAPITULO II ESTADO DEL ARTE

## 2.1 Introducción

Al agregar dispositivos móviles en un sistema distribuido surgen nuevos problemas que necesitan un manejo adecuado para diseñar mecanismos que generen *snapshot* distribuidos de manera distinta a la de un sistema de cómputo tradicional. Los problemas más destacados son la movilidad, las desconexiones, la fuente de energía finita, la vulnerabilidad a daños físicos, la capacidad de almacenamiento estable.

Con la movilidad, la ubicación de un *MH* en la red cambia con el tiempo por lo que las *MSS* antes de realizar una petición envían mensajes de control para localizar a los *MH* y si el mecanismo no toma en cuenta esto puede ocurrir un *overhead* en los canales de comunicación. Debido a la vulnerabilidad de los *MH* a fallas catastróficas, no es recomendable que guarden los registros de *snapshot* locales o mensajes y por lo tanto los mecanismos deben contar con alternativas de almacenamiento [7].

La desconexión voluntaria o involuntaria de uno o más *MH* no debe impedir que se lleve a cabo el proceso de generación de un estado global por lo tanto los mecanismos deben ser capaces de reaccionar ante este suceso garantizando la consistencia del estado global. La batería en el *MH* tiene una vida limitada y corta, y aunque el *MH* apague componentes internos que no utilice, la conservación de energía y las limitaciones del ancho de banda requieren de mecanismos que minimicen el número de mensajes y el número de *snapshot* [8].

Es por eso que para diseñar un mecanismo que genere *snapshot* distribuidos en sistemas de cómputo móvil existen técnicas que se adaptan de acuerdo a la comunicación entre las *MSS* y los MH clasificándose en tres métodos fundamentales: síncrono, asíncrono y cuasi-síncrono; y de acuerdo a los mensajes: registro.

## 2.2 Síncrono o Coordinado

Es el método más utilizado, su característica principal dada su nombre es que para generar el *snapshot* global requiere de una participación coordinada de los procesos que se vean involucrados durante este suceso de tal forma que se garantice la consistencia de los *snapshot*.

Los mecanismos que utilizan este método llevan a cabo en dos fases la generación de los *snapshot* a través del uso de mensajes *especiales*. En la primera fase el proceso que inicia la creación del *snapshot* distribuido envía un mensaje de petición al resto de los participantes para que éstos guarden un *tentativo snapshot* local (se almacena en la memoria volátil) y envíen un mensaje de respuesta. En la segunda fase el proceso iniciador al recibir las respuestas envía otro mensaje de confirmación para que el *snapshot* se vuelva *permanente*, es decir, se almacena en la memoria no volátil [9].

A continuación se describen las dos formas en las que se divide el método síncrono:

### 2.2.1 Bloqueante

El mecanismo bloquea o detiene las acciones que se encuentran realizando los procesos del sistema que reciben el mensaje de petición del proceso generador, estos procesos construyen el *snapshot* tentativo, posteriormente se encuentran a la espera del mensaje de confirmación para hacerlo permanente o descartarlo y continuar con su trabajo. Si el mecanismo detiene a todos los procesos se denomina un bloqueo total y si detiene a algunos procesos se denomina bloqueo parcial [10].

Singh & Cabilic, presentan el concepto de *snapshot sucesivo* para lograr un estado global consistente, se basan en el ahorro de energía que evita una computación innecesaria cuando se generen los *snapshot* ya que éstos no se descartan y además tampoco se construyen *snapshot inútiles* [11]. El algoritmo realiza un bloqueo parcial del sistema de manera restrictiva verificando la disponibilidad de los procesos tomando en cuenta los recursos que tienen para preservar la consistencia del sistema para que no existan retardos en los canales de comunicación. Sin embargo para garantizar que no exista el efecto dominó, los *snapshot* sucesivos utilizan mensajes *bloqueadores* para aquellos procesos donde pueda ocurrir un mensaje huérfano, de tal modo que los mensajes provenientes de otros procesos tienen que esperar en una cola hasta que el mensaje *desbloqueador* le haya llegado al proceso bloqueado.

Lim, propone un esquema donde combina un agente y el algoritmo para generar *snapshot* en un ambiente móvil distribuido que le permite a la aplicación móvil ajustar la cantidad de *snapshot* recién creados que se vayan a requerir en caso de ser necesaria una restauración del sistema [12]. El uso de las comunicaciones inalámbricas se reduce con el algoritmo ya que la cantidad de información que se transmite es mínima sin embargo es necesario el bloqueo del sistema durante un periodo corto de tiempo. El agente sólo se encuentra en cada una de las *MSS* y éste lleva un registro de los *snapshot* que se van generando y decide si cada *snapshot* pertenece a una línea de recuperación.

Suri & Satiza, proponen un protocolo para generar snapshot de manera coordinada el cual no bloquea de manera total al sistema, sino que durante el transcurso del sistema se fuerza a un mínimo número de procesos a construir sus *snapshot* sin suspender las tareas que se encuentren realizando [13]. El mínimo número de procesos se obtiene a través del vector de dependencia de información del proceso que inicia la generación del *snapshot* distribuido. Otra característica es que los *snapshot* son enumerados para garantizar la consistencia del estado global, con estas ventajas se disminuye el consumo de energía por parte de los *MH*; así como la utilización de los canales de comunicación y del ancho de banda. Aun así, su desventaja recae en la necesidad de mensajes de petición y confirmación para que los *snapshot* pasen de ser tentativos a permanentes.

### 2.2.2 No Bloqueante

A diferencia del anterior, éste no detiene las acciones de los procesos, sin embargo es necesario un control de los mensajes para evitar inconsistencias al momento de generar el *snapshot* distribuido [14].

Prakash & Singhal, realizan un algoritmo donde se combina la minimización del número de mensajes de sincronización para generar el estado global y la minimización del número de *snapshot* que se toman durante este proceso sin realizar algún tipo de bloqueo al sistema [15]. Ellos manejan dos tipos de mensajes los de cómputo que se refieren a los mensajes de la aplicación distribuida y los de sistema que se refiere a los mensajes que se utilizan para llevar a cabo el proceso de generación del *snapshot*. Conforme cada proceso toma un *snapshot*, el algoritmo va enumerando para así evitar inconsistencias, además para garantizar la dependencia entre los procesos se utilizan los vectores lógicos. Sin embargo, al forzar sólo una cantidad mínima de procesos obtenida a través de los vectores lógicos pueden suceder situaciones donde exista una

inconsistencia cuando se genere el estado global al haber dependencias transitivas entre otros procesos.

Neves & Fuchs, muestran un interesante protocolo para generar *snapshot* el cual se adapta a la calidad del servicio de la configuración de la red (*ethernet*, radio, celular, satelital) que hay en el entorno móvil [16]. La calidad del servicio toma en cuenta el ancho de banda, costo, tasa de pérdida de datos y la latencia. Este protocolo es capaz de almacenar estados globales sin necesidad de intercambiar mensajes utilizando un temporizador que determina cuándo un proceso debe generar su *snapshot* local. El protocolo utiliza los *snapshot* locales en caso de alguna falla en los *MH*, pero también utiliza dos *snapshot* globales para garantizar la consistencia del sistema, el primero es guardado en un almacenamiento estable en caso de una falla permanente y el segundo es dispersado a todos los *MH* en caso de alguna falla transitoria. A pesar de tener en cuenta la calidad del servicio de la red para los dispositivos móviles, dicha calidad se emplea para cambiar el temporizador que determina la periodicidad de la generación de *snapshot* sin tomar en cuenta la consistencia del sistema lo que hace que algunos *snapshot* puedan volverse *inútiles* para el estado global.

Cao & Singhal, diseñan un algoritmo que no bloquea a los procesos y que utiliza los mínimos necesarios relajando ciertas condiciones [17, 18]. Además, introducen el concepto de *snapshot mutables*, éstos son cercanos a los tentativos pero no son permanentes ya que se pueden almacenar en cualquier lugar evitando así el *overhead* sobre la red. Con esto Cao & Singhal logran evitar el efecto dominó haciendo que este algoritmo se pueda utilizar en la mayoría de las aplicaciones móviles distribuidas. A pesar de que utilizan *snapshot mutables*, tentativos y permanentes, se necesita de un vector para enumerar los *snapshot* tomados por cada proceso y también se requiere de un recolector de basura que descarte a aquellos *snapshot* que no se utilizaron, esto se debe a la utilización de los distintos dispositivos de almacenamiento por lo que repercute en un aumento en el consumo de energía por parte de los *MH*.

Lin, *et al.*, presentan un protocolo basado en el tiempo para generar *snapshot* en sistemas de cómputo móvil sobre *IP* Móvil [19]. El protocolo reduce el número de *snapshot* que toma cada proceso cuando que se genera un estado global, siendo el más mínimo posible para disminuir el número y tamaño de los mensajes que se transmiten a través de la red y por lo tanto también se disminuye el costo del tráfico cuando se transmiten los *snapshot* sobre los canales de comunicación. Este protocolo es no bloqueante porque las inconsistencias se evitan a través de información sobrellevada por cada mensaje. Aunque este protocolo reduce el overhead en los



canales de comunicación, genera el estado global de manera periódica utilizando recursos de forma innecesaria cuando no exista diferencia entre dos o más estados globales.

Kumas & Khunteta, proponen un algoritmo que utiliza una mínima cantidad de procesos, dicho algoritmo funciona para aplicaciones distribuidas determinísticas [20]. Esta mínima cantidad se obtiene del número de procesos que se encuentran en el vector de dependencias directas y transitivas del proceso iniciador. Kumar & Khunteta eliminan la utilización de *snapshot inútiles*, el bloqueo de procesos y tratan de minimizar la pérdida de la información que se envía durante la generación del estado global a través de estos vectores de dependencias en formato de bits, los cuales son sobrellevados en los mensajes volviéndose un problema para aplicaciones móviles que puedan ser escalables con respecto al número de *MH* porque se aumentaría el tamaño de los mensajes.

Garh & Kumar, presentan un algoritmo para generar *snapshot* de manera coordinada sin realizar un bloqueo total o parcial del sistema [21]. Este algoritmo se basa en mantener las dependencias directas entre los procesos para que la *MSS* obtenga el conjunto mínimo de ellos y enviarles la petición de que construyan su *snapshot* de tal manera que se reduzcan la construcción de *snapshot inútiles*. Este algoritmo tiene la desventaja de tener los mensajes de petición y confirmación para que los *snapshot* pasen de ser tentativos a permanentes, de lo contrario no se guardarían en el almacenamiento fijo, además si ocurre una falla en algún proceso se aborta la generación de los *snapshot* y se genera un aumento en los recursos de los *MH* con respecto a sus procesos involucrados.

Panghal, *et al.*, presentan el algoritmo síncrono que utiliza una mínima cantidad de procesos para generar *snapshot* en un ambiente móvil distribuido y disminuir los costos de consumo de energía y ancho de banda [22]. La mínima interacción entre los procesos que utiliza hace que este algoritmo no realice un bloqueo parcial o total del sistema porque dicha interacción proviene del número de procesos dependientes directos o transitivos del proceso iniciador. Este algoritmo tiene la desventaja de tener los mensajes de petición y confirmación para que los *snapshot* sean permanentes, por lo que si ocurre una falla en algún proceso se aborta la generación de los *snapshot*.

Tuli & Kumar, proponen un esquema para generar snapshot en redes móviles *Ad Hoc* [23]. El esquema contiene un clúster basado en un protocolo de ruteo, el algoritmo de este protocolo genera el estado global del sistema de manera coordinada sin realizar un bloqueo de los procesos

ya que selecciona una mínima cantidad de procesos para llevar a cabo esta tarea. Aunque el esquema minimiza el uso de mensajes de control, los mensajes de petición y confirmación son necesarios para que el conjunto de *snapshot* pase de ser tentativos a permanentes y completen la generación del estado global.

Kaware & Ramteke, proponen un método para generar *snapshot* en sistemas de cómputo móvil, el cual contempla el consumo de energía que genera cada *MH* al construir y guardar su estado local, de esta manera el sistema ajusta el intervalo de tiempo para generar el siguiente *snapshot* [24]. Las *MSS* guardan los estados locales de sus respectivos *MH* para que en caso de ser necesario los retransmitan de nuevo a ellos. Sin embargo, los *snapshot* se siguen generando de manera periódica no toma en cuenta la consistencia del sistema lo que hace que algunos *snapshot* puedan volverse *inútiles* para el estado global.

## 2.3 Asíncrono o Sin Coordinador

En este método, y como su nombre lo indica, no existe una coordinación entre los procesos ya que cada uno de ellos genera su *snapshot* de manera independiente. Esto le permite a cada proceso la máxima autonomía para decidir cuándo debe generar su *snapshot* lo que elimina el *overhead* causado por la coordinación de procesos y la recuperación del sistema [4].

Para determinar un estado global consistente los procesos deben guardar las dependencias que existen entre los *snapshot* por lo que pueden existir *snapshot* que no pertenezcan a un estado global y por lo tanto es necesario de un recolector de basura [3].

Park, *et al.*, presentan un esquema de recuperación de manera asíncrona y utilizando un registro de los mensajes recibidos [25]. Este esquema toma en cuenta la movilidad de los dispositivos para controlar el costo de los recursos en transferencia de información y en la construcción del estado global. Con el registro de los mensajes y la periodicidad de generar *snapshot* hace que la recuperación de manera asíncrona sea factible en caso de que múltiples y concurrentes fallas en el sistema. Respecto a la movilidad, si un *MH* cambia de una *MSS* a otra, la información de registro y recuperación es enviada a esta última, por lo que la responsabilidad de la petición para generar el estado global recae en las *MSS*. Sin embargo, generar *snapshot* periódicamente puede ser un proceso innecesario ya que posiblemente existan un punto donde dos o más estados globales sean similares y por lo tanto se desperdicie el uso de los recursos de los *MH* involucrados.

Singh, presenta un protocolo para generar *snapshot* basado en índices y en el tiempo [26]. El algoritmo de este protocolo es no bloqueante, adaptivo y no utiliza mensajes de control ya que los índices se utilizan para que cada proceso verifique si se debe generar el *snapshot*. También, el protocolo toma el menor número de *snapshot* para generar un estado global y no necesita computar relaciones de dependencia. Esto reduce el *overhead* tanto en los procesos como en los canales de comunicación ya que el protocolo no genera ningún *snapshot* tentativo. Sin embargo, al estar basado en el tiempo los estados se generan cada cierto tiempo por lo que se puede incurrir en un uso innecesario de recursos cuando no exista diferencia entre dos o más estados globales.

Vutukuru, *et al.*, proponen un esquema de recuperación para sistemas de cómputo móvil [27]. Si un proceso falla, éste retorna a su último estado guardado y además todos los mensajes que ocurrieron antes de la falla le son retransmitidos por su *MSS*, ya que las *MSS* llevan un registro de los mensajes que han sido generados, por lo que no existe el efecto dominó ni perdida de mensajes. Las estructuras de datos que manejan generan bajos consumos de ancho de banda, energía y memoria. Este esquema por ser de naturaleza asíncrona no realiza un bloqueo parcial o total del sistema por lo que se garantiza la continuidad del sistema en caso de que falle un proceso. El manejo de la movilidad hace que la recuperación del *MH* pueda ser desde la *MSS* donde se encontraba conectado o cuando se conecte a otra *MSS*. Sin embargo, este esquema genera *overhead* sobre los canales de comunicación porque requiere de distintos mensajes de control para que la recuperación del sistema sea exitosa.

## 2.4 Cuasi-síncrono o Hibrido

Este método utiliza dos tipos de *snapshot* para generar un estado global, los *snapshot* locales y los *snapshot* forzados. Los primeros se generan de manera independiente durante el transcurso del tiempo y los segundos se generan para garantizar el progreso del sistema de tal manera que se minimicen los *snapshot inútiles* y exista menor pérdida de información [28].

El método cuasi-síncrono no intercambia mensajes de petición y confirmación para generar los *snapshot*, sino que la información es sobrellevada en el intercambio de mensajes entre los proceso. Con esta información el sistema deberá construir un estado global consistente [29].

Agbaria & Sanders, realizan una adaptación de la técnica tradicional para *snapshot* distribuidos en el entorno móvil [30]. En su algoritmo proponen una comunicación síncrona entre los procesos suponiendo que los canales de comunicación son confiables para que sean modelados como

procesos deterministas. Las *MSS* son los responsables de iniciar el proceso para generar un estado global a través de marcadores entre mensajes, envían las peticiones en los marcadores hacia los *MH* donde éstos generan su *snapshot* local y lo envían a sus *MSS*. Al final las *MSS* construyen el *snapshot* distribuido a partir de los *snapshot* locales.

Gupta, *et al.*, proponen un esquema hibrido para aplicaciones determinísticas en sistemas de cómputo móvil en donde minimizan el número de procesos que se utilizan para generar un estado global sin bloquear al sistema y sin generar *snapshot inútiles* [31]. Los procesos que realizan los *snapshot* son seleccionados de acuerdo a la dependencia directa o transitiva que tengan con el proceso iniciador y del valor del *snapshot* reciente que hayan realizado, con esto tratan de disminuir la pérdida de trabajo computacional que se haya hecho. Este esquema sobrelleva la información para los *snapshot* a través de los mensajes que se envían para determinar cuándo se deben de generar para ahorrar batería y tener un consumo bajo del ancho de banda. También se utilizan los denominados *anti-mensajes* que se utilizan para confirmar que un mensaje haya llegado a su destino y así eliminar los mensajes duplicados. Por lo que el problema reside en la saturación de los canales de comunicación al utilizar distintos tipos de mensajes.

## 2.5 Registro

Este método no toma en cuenta la coordinación entre los procesos sino la recepción de los mensajes. Los procesos guardan los mensajes que han recibido en el almacenamiento estable y con esto llevan un registro. Cuando un proceso falla, uno nuevo se crea para tomar su lugar y se le entrega el registro del proceso fallido. El nuevo proceso reproduce los mensajes contenidos en el registro tal y como le fueron llegando originalmente al proceso fallido, de esta forma se garantiza la consistencia del estado global cuando este sea generado [1].

Biswas & Neogy, proponen un protocolo para generar *snapshot* basado en patrones de la movilidad de los dispositivos ya sean del tipo intercelda, intracelda o combinación de ambas [32]. El algoritmo utilizado no bloquea al sistema por lo que los mensajes que se envían o reciben durante la generación de los *snapshot* son registrados para su posterior procesamiento. Este protocolo utiliza dos tipos de mensajes, los de ejecución para los mensajes del sistema y los de control para coordinar la construcción del estado global. Además, maneja dos tipos de *snapshot*, los permanentes para el estado global y los *migratorios* para cuando los *MH* planeen una desconexión del sistema. Sin embargo, el hecho de manejar *snapshot migratorios* y mensajes de control puede generar una carga mayor hacia los recursos de cada *MH* y del tráfico de la red.

Nagpal & Kumar, proponen un algoritmo que genera *snapshot* de manera coordinada a través de tres fases utilizando una cantidad mínima de procesos [33]. Dicha mínima cantidad proviene del número de procesos dependientes directos o transitivos del proceso la generación del snapshot. Este algoritmo no crea *snapshot inútiles*, ni bloquea parcial o totalmente a los procesos sino que se basa en el registro de mensajes que se hayan enviado durante el transcurso del sistema, por lo que tratan de minimizar la pérdida de trabajo computacional. El proceso para generar un estado global se basa en el envío de mensajes de petición y confirmación, si algún proceso falla sólo éste aborta la operación, lo que significa en un desperdicio de recursos, y los demás continúan operando por lo que es posible que pueda existir alguna inconsistencia que se haga una recuperación del sistema. Por lo que utilizar los mensajes de petición y confirmación aumentando el *overhead* hacia los canales de comunicación puede generar un cuello de botella para mensajes posteriores.

Como se pudo observar, los mecanismos expuestos anteriormente evitan algunos los problemas cuando generan *snapshot* en un ambiente móvil distribuido. El mecanismo causal difuso al utilizar el método asíncrono no realiza ningún tipo de bloqueo en el sistema. Asimismo, el mecanismo no depende de ningún periodo de tiempo para generar a los *snapshot* sino que depende del comportamiento del estado del sistema y su calidad de servicio. Además, no necesita utilizar mensajes especiales que saturen el tráfico de la red así como también no necesita utilizar recursos de los *MH*. Por lo que se asume que la carga de trabajo que origina el mecanismo es relativamente baja para el propio sistema. La figura 1 muestra la tabla comparativa entre el mecanismo causal difuso y algunos trabajos de investigación más recientes, en esta tabla se muestra los problemas que evita el mecanismo causal difuso (MCD).

| | Lim [12] | Suri & Satiza [13] | Kaware & Ramteke [24] | Vutukuru *et al.* [27] | Nagpal & Kumar [33] | MCD |
|---|---|---|---|---|---|---|
| **Periodos** | Si | Si | No | Si | Si | Si |
| **Uso de recursos** | Si | No | No | Si | No | Si |
| **Cuellos de botella** | Si | No | Si | No | No | Si |
| **Bloqueos** | No | No | Si | Si | Si | Si |
| **Carga de trabajo** | Normal | Alta | Alta | Normal | Alta | Baja |

*Figura 1: Comparación de características entre mecanismos.*

# CAPITULO III PRELIMINARES

## 3.1 Definiciones

A continuación se describen las definiciones utilizadas en este trabajo las cuales ayudarán a dar una mejor comprensión del mecanismo propuesto.

### 3.1.1 Distancia Causal

López, establece que la distancia causal entre dos mensajes causalmente dependientes es el mayor número de mensajes pares dependientes enviados entre ellos más uno [34]. Su definición formal es la siguiente:

**Definición 1**. La distancia $d(m, m')$, se define para cualquier par de mensajes $m$ y $m' \in M$ tal que $m \rightarrow m'$: $d(m, m')$ es el mayor entero $n$ tal que para cualquier secuencia de mensajes $(m_i, i=0...n)$ con $m=m_0$ y $m'=m_n$, se obtiene $m_i \downarrow m_{i+1}$ para todo $i = 0...n-1$.

### 3.1.2 Relación Causal Difusa

Morales, establece que la relación causal difusa (*Fuzzy Causal Relation*, *FCR*) denotada como $a \xrightarrow{\lambda} b$, se basa en una noción de *distancia* entre eventos [35]. La distancia se puede establecer considerando tres principales dominios: espacial, temporal y/o lógico. Usando la noción de *distancia*, la relación causal difusa establece un grado de causa-efecto indicando *hace cuánto* un evento *a* sucedió antes de un evento *b*. La definición formal de la *FCR* es la siguiente:

**Definición 2**. La relación causal difusa ($\overset{\lambda}{\rightarrow}$) en un conjunto de eventos $E$ satisface las siguientes condiciones:

$$a \overset{\lambda}{\rightarrow} b \text{ If } a \rightarrow b \wedge 0 \leq DR(a,b) < 1$$

$$a \overset{\lambda}{\rightarrow} c \text{ If } \exists \text{b} \mid a \rightarrow b \rightarrow c \wedge DR(a,b) \leq DR(a,c) : DR(a,b), DR(a,c) < 1$$

Donde:

- La primera condición establece que dos eventos (*a*, *b*) tienen una relación causal difusa si *a* sucedió antes que *b* y el valor de la relación difusa entre ellos, *DR(a, b)*, es más pequeña que la unidad.
- La segunda condición es la propiedad transitiva. Esta condición establece que dos eventos (*a*, *c*) tienen una relación causal difusa si existe un evento *b* tal que *a* sucedió antes que *b*, y *b* sucedió antes que *c*. Además, los valores para *DR(a, b)* y *DR(a, c)* crecen monótonamente y son más pequeños que la unidad.

En este trabajo, una suposición considerada para la *FCR* es *que tan consistente* es la relación de causa-efecto entre los eventos de acuerdo a los tres dominios entre un par de eventos.

### 3.1.3 Consistencia Causal Difusa

Morales, establece que la consistencia causal difusa (*Fuzzy Causal Consistency*, *FCC*) se basa en la *FCR* [35]. Su objetivo es indicar *que tan bueno* es el rendimiento del sistema en un cierto tiempo.

La *FCC* se calcula mediante el promedio ponderado de las relaciones causales difusas para cada evento contenido en el historial causal *H(a)* del evento *a* del cual se desea conocer la consistencia del sistema. El promedio ponderado es una cifra que resulta ser idéntica o más cercana a la media aritmética. La *FCC* sólo considera los eventos con una dependencia inmediata con el evento *a* dado que estos eventos tienen una afectación directa con este evento. Los valores de la *FCC* para este trabajo se normalizan dentro del intervalo [0,1].

La figura 2 muestra la manera en cómo se obtiene la *FCC* para un evento *a* sobre un proceso *p*.

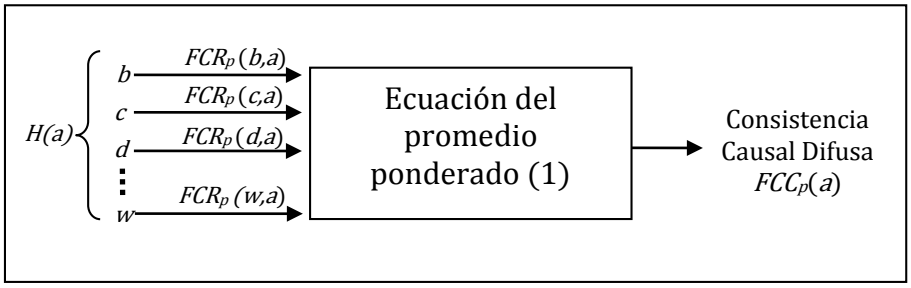


*Figura 2: Cálculo de consistencia causal difusa.*

$$FCC_p(a) = \frac{\sum_{b \in H(a)} FCR_p(a,b) * GP(b)}{\sum_{b \in H(a)} GP(b)} \ldots (1)$$

Donde:

- *GP*(*b*) es el grado de ponderación que se utiliza para determinar las prioridades o pesos para cada relación causal difusa.
- $FCR_p(a, b)$ es la relación causal difusa del par de eventos (*a*, *b*) sobre el proceso *p*.

## 3.2 Modelo del sistema

El modelo del sistema para la generación del *snapshot* distribuido considera las siguientes características:

- Un sistema de cómputo móvil (ver figura 3) está constituido de una cantidad finita de dispositivos móviles (*MH*) y estaciones de soporte móvil (*MSS*).
- Las *MSS* se interconectan a través de una conexión cableada y se asume que la comunicación entre ellas es confiable. Cada *MSS* tiene una conexión inalámbrica a la cual se conectan los *MH* dentro de un área limitada llamada celda y se asume que esta comunicación no es confiable.
- Los *MH* pueden conectarse a distintas *MSS* y son libres de moverse entre las celdas pero están asociados únicamente a una MSS. Si un *MH* quiere comunicarse con el resto del grupo lo puede hacer a través de la *MSS* a la cual esté conectado directamente.
- Se asume que el tipo de transmisión de mensajes es por el paso de mensajes mediante *broadcast* y que pueden existir retardos aleatorios durante dicha transmisión.
- Las *MSS* son las encargadas de enviar la petición para generar los *snapshot* así como de almacenarlos.

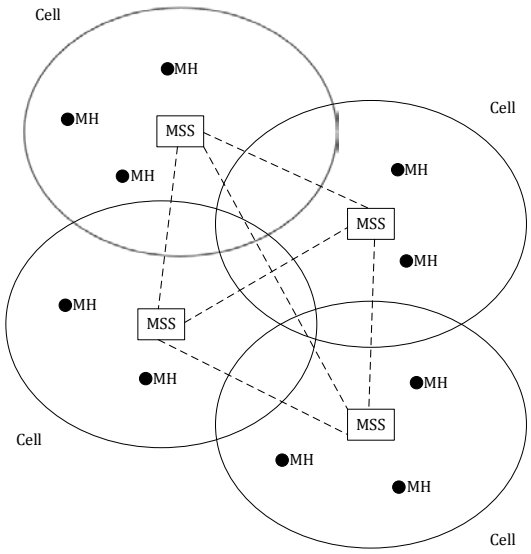


*Figura 3: Diagrama de un sistema de cómputo móvil.*

# CAPITULO IV MECANISMO CAUSAL DIFUSO

## 4.1 Introducción

Un proceso en el sistema distribuido móvil es ejecutado solamente en un *MH* o en una *MSS*. Cuando se trata de este tipo de sistemas utilizar sólo un reloj físico no es factible para generar un *snapshot* distribuido. Por eso una alternativa al reloj físico es el reloj lógico ya que al utilizarlo se puede establecer el ordenamiento que deben tomar los eventos a través de su cronología y también de las relaciones que existen entre dichos eventos.

Los eventos en un sistema distribuido se dividen en tres tipos: *internos*, los cuales ocurren dentro de un proceso y los otros procesos nunca tienen conocimiento de ellos; envío (*send*) y recepción (*receive*), los cuales utilizan los procesos para comunicarse entre ellos y llevar a cabo la cooperación, el aspecto importante de estos eventos es que sólo a través de ellos es posible modificar el estado global del sistema.

Cuando se trata de un sistema distribuido móvil, los eventos de *envío* y *recepción* son importantes porque afectan en cierto grado la consistencia de cada *snapshot* que se genere, es por eso que este trabajo hace uso del ordenamiento parcial de estos eventos para permitir una vista ordenada de ellos de tal manera que para un subconjunto de estos eventos los procesos tendrán la misma vista garantizando la consistencia del sistema. Por lo que el algoritmo de este mecanismo utiliza dichos eventos para llevar a cabo la generación de los *snapshot* distribuidos.

## 4.2 Mecanismo causal difuso

El mecanismo para la generación de *snapshot* distribuido en sistemas de cómputo móvil está conformado de tres componentes (ver figura 4). El primer componente es el estado del sistema donde se encuentra el conjunto de variables de entrada que alimentará a la entrega causal difusa. El segundo componente es la entrega causal difusa, incluye la *FCR* y la *FCC* aplicadas al momento de la entrega de los mensajes sobre las *MSS*. El tercer componente es el generador del *snapshot* distribuido en donde se le determina al sistema cuándo debe generar el *snapshot*.

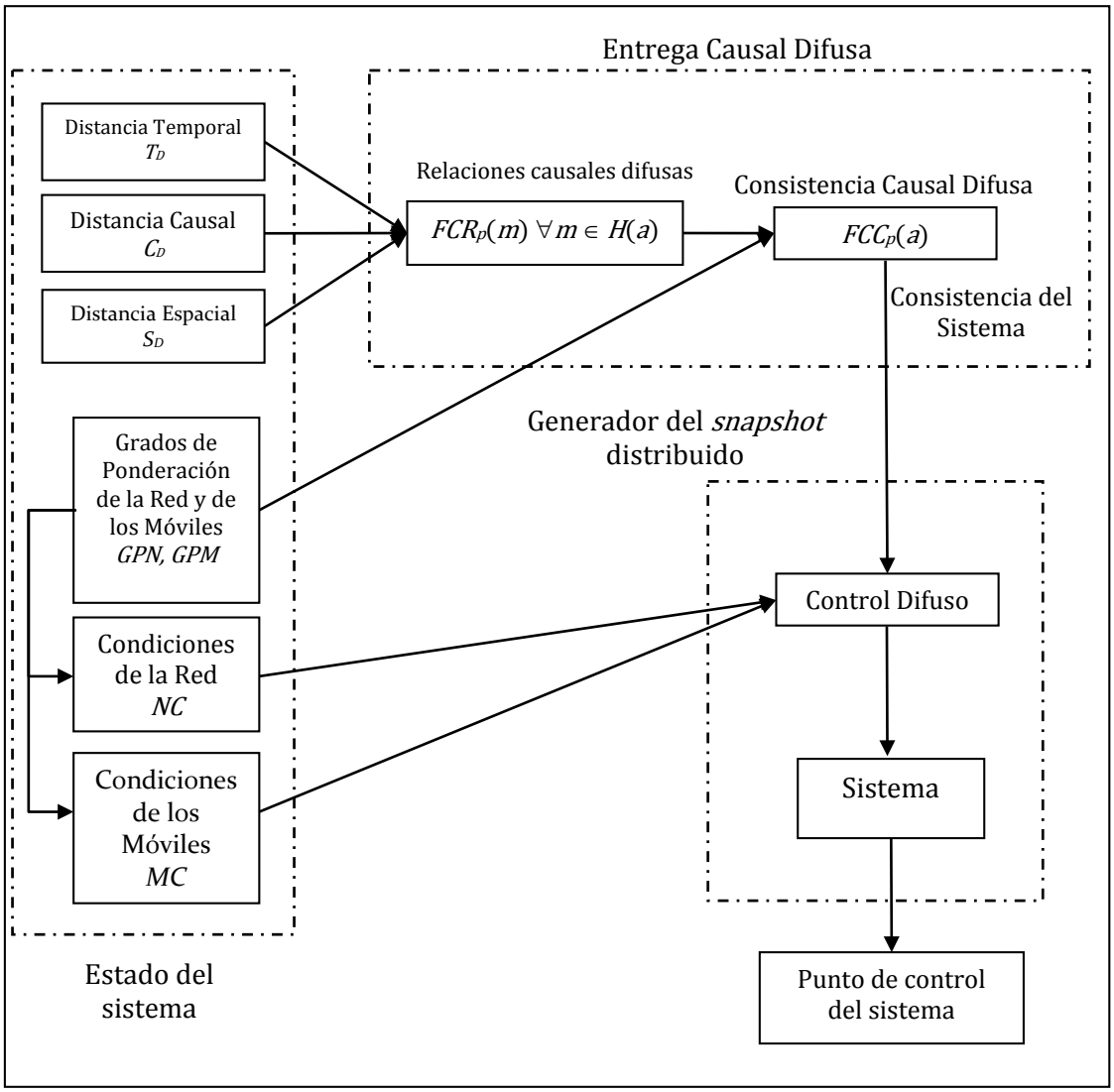


*Figura 4: Mecanismo causal difuso.*

## 4.3 Estado del sistema

Este componente contiene las variables del estado actual del sistema que se calculan cuando un mensaje *a* es recibido por una *MSS*. Estas variables provienen de las condiciones de la red y de las condiciones del dispositivo móvil de donde fue enviado el mensaje.

### 4.3.1 Distancia causal

La *distancia causal* se usa como variable de entrada para la función de membresía $C_D$. La $C_D(m)$ mide la distancia local entre cada mensaje causal contenido en el historial causal de $m$ y el último mensaje recibido por cada proceso de un móvil contenido en su historial causal. Con el fin de

conocer el valor de la distancia causal entre dos eventos se ha hecho un algoritmo para la transmisión *broadcast* en sistemas de cómputo móvil. El algoritmo se muestra en el apéndice A.

### 4.3.2 Distancia temporal

La *distancia temporal* se usa como variable de entrada para la función de membresía $T_D$. La $T_D(m)$ mide el tráfico de la red tanto de las *MSS* y de los *MH* entre cada mensaje contenido en el historial causal de $m$ y el último mensaje recibido por cada proceso de un móvil contenido en su historial causal.

### 4.3.3 Distancia espacial

La *distancia espacial* se usa como variable de entrada para la función de membresía $S_D$. La $S_D(m)$ mide la intensidad de la señal del dispositivo móvil que envió el mensaje $m$ y la de cada mensaje contenido en su historial causal de $m$.

### 4.3.4 Grados de ponderación

Los grados de ponderación determinan la degradación de un canal (proceso) conforme al mejor que exista en el sistema. Estos grados se utilizan en la fórmula que determina la consistencia causal difusa.

#### 4.3.4.1 De la red (*GPN*)

La variable *GPN* es el grado de ponderación de la red y se utiliza en la fórmula que determina las condiciones de la red, ver fórmula 2. Para calcular la *GPN* de los mensajes relacionados causalmente al mensaje *a*, es necesario primero calcular el canal con las mejores condiciones de la red (*BNChannel*), ver fórmula 3.

$$GPN_i = \left(\frac{RTT_i + RTT_j + DR_i + DR_j}{BNChannel}\right) \ldots (2)$$

$$BNChannel = \min_{i=1}^{n}\left[RTT_i + RTT_j + DR_i + DR_j\right] \ldots (3)$$

A continuación se describe el significado de cada una de las variables utilizadas en las ecuaciones:

- $i$ es el identificador del proceso que se ejecuta sobre el dispositivo móvil.

- *j* es el identificador del proceso que se ejecuta sobre la estación de soporte móvil asociada con el proceso *i*.
- *RTT* es el tiempo de ida y vuelta para un mensaje.
- *DR* es el porcentaje de la perdida de datos para un mensaje.
- *n* es el número de canales (procesos) de los *MH* que están relacionados causalmente al evento *a*.

A continuación se describen los valores del $GPN_i$:

- Si el valor del $GPN_i$ tiende a cero significa que el canal en el $MH_i$ junto con el canal de la $MSS_j$ al que está asociado tiene una *pequeña* degradación con respecto al mejor canal.
- Si el valor del $GPN_i$ tiende a uno significa que el canal en el $MH_i$ junto con el canal de la $MSS_j$ al que está asociado tiene una *gran* degradación con respecto al mejor canal.

#### 4.3.4.2 De los móviles (*GPM*)

La variable *GPM* es el grado de ponderación de los móviles y se utiliza en la fórmula que determina las condiciones de los móviles, ver fórmula 4. Para calcular la *GPM* de los mensajes relacionados causalmente al mensaje *a*, es necesario primero calcular el canal con las mejores condiciones de la red, *BMChannel*, ver fórmula 5.

$$GPM_i = \left(\frac{CPU_i + RAM_i}{BMChannel}\right) \ldots (4)$$

$$BMChannel = \ min_{i=1}^{n}[CPU_i + RAM_i] \ldots (5)$$

A continuación se describe el significado de cada una de las variables utilizadas en las ecuaciones:

- *i* es el identificador del proceso que se ejecuta sobre el dispositivo móvil.
- *CPU* es la capacidad de procesamiento con la que cuenta el $MH_i$.
- *RAM* es la capacidad de memoria con la que cuenta el $MH_i$.
- *n* es el número de canales (procesos) que están relacionados causalmente al evento *a*.

A continuación se describen los valores del $GPM_i$:

- Si el valor del $GPM_i$ tiende a cero significa que el canal en el $MH_i$ tiene una *pequeña* degradación con respecto al mejor canal.

- Si el valor del $GPN_i$ tiende a uno significa que el canal en el $MH_i$ tiene una *gran* degradación con respecto al mejor canal.

### 4.3.5 Condiciones de la red

La variable *NC* determina de una manera indirecta las condiciones de la red que hay en el sistema de cómputo móvil. La *NC* representa de manera cualitativa los cambios en las condiciones de la red, por ejemplo los retardos entre mensajes, la perdida de mensajes, la congestión en la red y el ancho de banda disponible. Los valores de la *NC* se normalizan dentro del intervalo [0,1] conforme a la función de membresía triangular, ver figura 5.

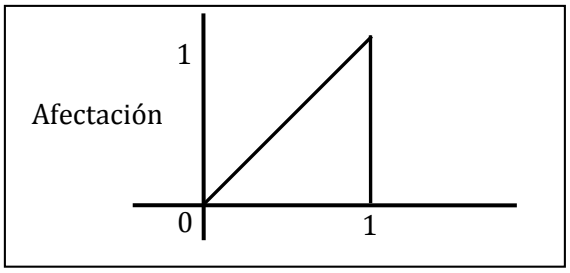


*Figura 5: Función de membresía de la NC.*

Las condiciones de la red se calculan cuando un mensaje *a* es recibido por parte de una *MSS* sin necesidad de agregar información extra ha dicho mensaje. La *NC*(*a*) se calcula mediante el promedio ponderado de la condición de red de cada canal (proceso) que se incluye en el historial causal del mensaje *a*. Recordando que el valor del $GPN_i$ para cada canal indica su degradación con respecto al mejor canal. Utilizando estos valores se calculan las condiciones de la red para el sistema mostrando que los canales con las peores condiciones tienen la mayor afectación al sistema.

A continuación se describe el significado para los valores de la *NC*(*a*):

- Si *NC*(*a*)→0 significa que las condiciones de la red son *buenas*, esto indica que el comportamiento de la red se encuentra dentro los parámetros de la calidad de servicio.
- Si *NC*(*a*)→1 significa que las condiciones de la red son *malas*, esto indica que el comportamiento de la red se encuentra fuera de los parámetros de la calidad de servicio.

La variable *NC* se calcula utilizando la ecuación 6.

$$NC(a) = \frac{\sum_{i=1}^{n}\left[\frac{2B_j}{RTT_j} + \frac{2B_i}{RTT_i} + DR_j + DR_i\right] * GPN_i}{S_{GPN}} \ldots (6)$$

A continuación se describe el significado de cada una de las variables utilizadas en la ecuación:

- *i* es el identificador del proceso que se ejecuta sobre el dispositivo móvil.
- *j* es el identificador del proceso que se ejecuta sobre la estación de soporte móvil asociada con el proceso *i*.
- *B* es el ancho de banda disponible en el $MH_i$ y la $MSS_j$ al que está asociado.
- *RTT* es el tiempo de ida y vuelta para un mensaje en el $MH_i$ y la $MSS_j$ al que está asociado.
- *DR* es el porcentaje de la pérdida de datos para un mensaje para el $MH_i$ y la $MSS_j$ al que está asociado.
- $S_{GPN}$ es la suma de los grados de ponderación de los canales.
- *n* es el número de canales (procesos) de los *MH* que están relacionados causalmente al evento *a*, recordando que estos se encuentran el historial causal de *a*.

Para una mejor compresión de la ecuación, el factor $\frac{2B}{RTT}$ mide el retraso de la transmisión que existe a través de la red.

### 4.3.6 Condiciones de los móviles

La variable *MC* determina de una manera indirecta las condiciones de las capacidades de cada dispositivo móvil que hay en el sistema. La *MC* representa de manera cualitativa las capacidades que tiene cada dispositivo, por ejemplo la batería, el almacenamiento fijo, el procesador y la memoria *RAM*. Los valores de la *MC* se normalizan dentro del intervalo [0,1] conforme a la función de membresía triangular, ver figura 6.

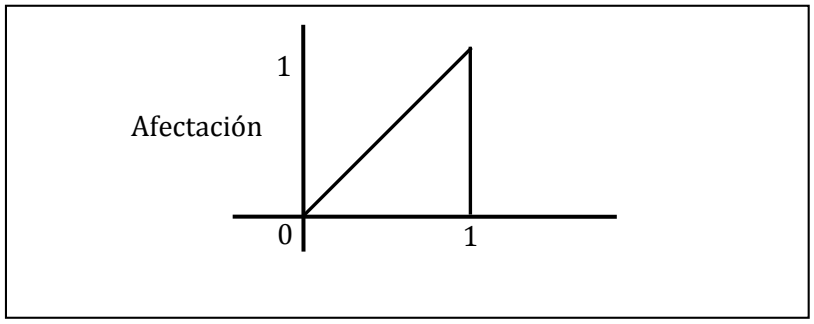


*Figura 6: Función de membresía de la MC.*

Las condiciones de los móviles únicamente incluyen a los *MH* y se calculan cuando un mensaje *a* es recibido por parte de una *MSS* sin necesidad de agregar información extra ha dicho mensaje. La *MC*(*a*) se calcula mediante el promedio ponderado de la condición móvil de cada canal que se incluye en el historial causal del mensaje *a*. Recordando que el valor del $GPM_i$ para cada canal indica su degradación con respecto al mejor canal. Utilizando estos valores se calculan las condiciones de la red para el sistema mostrando que los canales con las peores condiciones tienen la mayor afectación al sistema.

A continuación se describe el significado para los valores de la *NC*(*a*):

- Si *MC*(*a*)→0 significa que las condiciones de los móviles son *buenas*, esto indica que las capacidades de los dispositivos móviles involucrados en el envío de *a* están por encima del *MH* con las más bajas capacidades.
- Si *MC*(*a*)→1 significa que las condiciones de los móviles son *malas*, esto indica que las capacidades de los dispositivos móviles involucrados en el envío de *a* están cercanas del *MH* con las más bajas capacidades.

La variable *MC* se calcula utilizando la ecuación 7.

$$MC(a) = \frac{\sum_{i=1}^{n} \left[\frac{BT_i + ROM_i}{CPU_i + RAM_i}\right] * GPM_i}{S_{GPM}} \ldots (7)$$

A continuación se describe el significado de cada una de las variables utilizadas en la ecuación:

- *i* es el identificador del proceso que se ejecuta sobre el dispositivo móvil.
- *BT* es la capacidad de la batería con la que cuenta el $MH_i$.
- *ROM* es la capacidad de almacenamiento con la que cuenta el $MH_i$.
- *CPU* es la capacidad de procesamiento con la que cuenta el $MH_i$.
- *RAM* es la capacidad de memoria con la que cuenta el $MH_i$.
- $S_{GPM}$ es la suma de los grados de ponderación de los canales.
- *n* es el número de canales (procesos) de los *MH* que están relacionados causalmente al evento *a*, recordando que estos se encuentran el historial causal de *a*.

## 4.4 Entrega causal difusa

Este componente tiene como objetivo realizar en cierta forma la entrega de mensajes en el sistema de cómputo móvil. Con el fin de mostrar cómo se lleva a cabo este proceso la figura 7 presenta un escenario móvil distribuido entre dos dispositivos móviles y dos estaciones de soporte móvil.

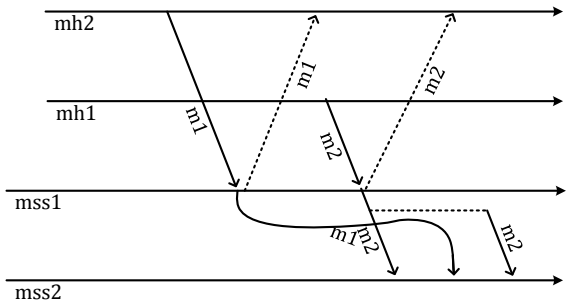


*Figura 7: Ejemplo de un escenario móvil distribuido.*

### 4.4.1 Aplicación de la *FCR* a la entrega de mensajes

En este trabajo se consideran los tres dominios de la relación causal difusa para resolver los problemas cuando se genera un *snapshot* distribuido en los sistemas de cómputo móvil. Hay que recordar que estos dominios se utilizan como parámetros de entrada para la $DR(a,b)$. Para cada dominio, una función de membresía se define, $C_D$, $T_D$ y $S_D$ respectivamente. El dominio lógico se considera porque a través de él es posible establecer la causalidad entre los eventos, con el dominio temporal es posible establecer el tráfico existente de la red durante la transmisión de dichos eventos y el dominio espacial establece las *posiciones geográficas* de los *MH* con respecto a las *MSS*, los cuales están conectados.

El valor de la *FCR* para un par de mensajes $FCR(g,h)$ viene determinada de la unión de las tres funciones de membresía, $C_D \cup T_D \cup S_D$, dominios lógico, temporal y espacial respectivamente. El operador difuso elegido como operador de unión es la función máximo, ver ecuación 8. La *FCR* se define formalmente por la ecuación 9 como:

$$DR(g,h) = \max(R_D, R_N, R_S) \ldots (8)$$

$$FCR(g,h) = DR(g,h) \text{ if } g \xrightarrow{\lambda} h \ldots (9)$$

Las funciones de membresía se normalizan dentro del intervalo [0,1]. El significado de los valores obtenidos por la $FCR(g,h)$ son los siguientes:

- Cuando $FCR(g,h)\rightarrow 0$ indica que los mensajes se están entregando en condiciones por encima de lo normal y existe un menor grado de afectación para la consistencia del sistema.
- Cuando $FCR(g,h)\rightarrow 1$ indica que los mensajes se están entregando en condiciones por debajo de lo normal y existe un mayor grado de afectación para la consistencia del sistema.

La normalización del intervalo [0,1] para los valores de las funciones de membresía, $C_D$, $T_D$ y $S_D$ se pueden calcular a través de la función triangular que se encuentra en la ecuación 10.

$$R(x) = \begin{cases} 0, if\ x < f \\ \dfrac{x-f}{h-f}, if\ f \leq x < h \\ 1, if\ x \geq h \end{cases} \ldots (10)$$

### 4.4.2 Aplicación de la *FCC* a la entrega de mensajes

Al aplicar la *FCC* para resolver los problemas cuando se genera un *snapshot* distribuido en los sistemas de cómputo móvil se obtiene una medida cualitativa del estado del sistema de acuerdo a las dependencias temporal, lógica y espacial en un cierto tiempo con respecto a la vista parcial que tiene cada proceso. Los valores de la *FCC* se normalizan dentro del intervalo [0,1] y se usan para asignar etiquetas lingüísticas del estado del sistema (*bueno*, *regular* o *malo*) de acuerdo a las condiciones en las que fueron enviados los eventos a través de red, ver figura 8.

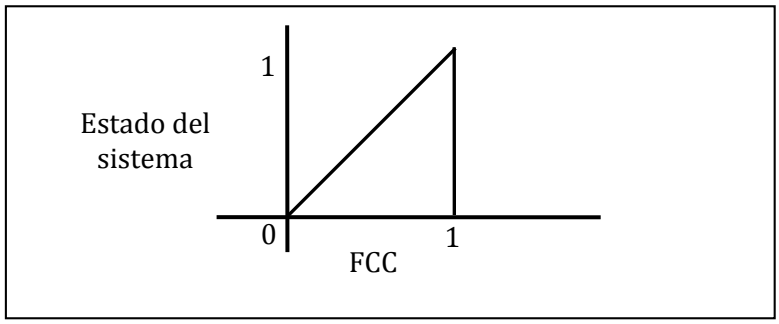


*Figura 8: Función de membresía de la FCC.*

Al utilizar la ecuación 1 se puede conocer el estado del sistema. La *FCC* se calcula mediante el promedio ponderado de las relaciones causales difusas para cada evento que se encuentra en el historial causal $H(a)$ del evento $a$.

Los significados de los valores obtenidos por la $FCC_p(a)$ son los siguientes:

- Cuando $FCC_p(a) \rightarrow 0$, indica que el estado del sistema es *bueno*, lo que significa que se satisfacen las dependencias temporal, lógica y espacial en el momento de la entrega de los mensajes manteniendo un buen estado consistente del sistema y por lo tanto no hay necesidad de enviar la petición para generar el *snapshot* distribuido.
- Cuando $FCC_p(a) \rightarrow 1$, indica que el estado del sistema es *regular* o *malo*. En este caso las dependencias temporal, lógica y espacial no se satisfacen completamente en el momento de la entrega los mensajes lo que significa que el sistema está cambiando a un estado inconsistente y por lo tanto es necesario enviar la petición para generar el *snapshot* distribuido.

## 4.5 Generador del *snapshot* distribuido

Este componente tiene como objetivo indicarle al sistema si debe o no generar un *snapshot* distribuido y si la generación del *snapshot* implica eliminar *snapshot* anteriores o mantenerlos. La figura 9 muestra el diseño del control difuso, sus variables de entrada son: las condiciones de la red (*NC*), las condiciones de los móviles (*MC*) y la consistencia causal difusa (*FCC*). La variable de salida se utiliza para indicar el punto de control del sistema y determinar si debe generar el *snapshot* distribuido.

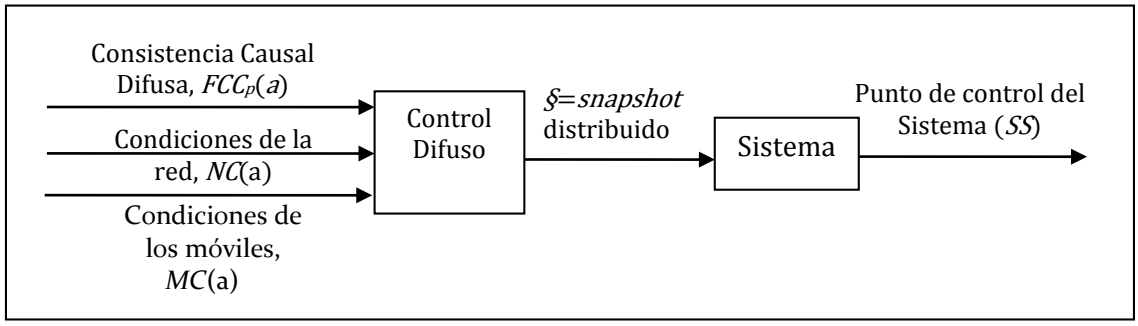


*Figura 9: Diagrama del control difuso*

El punto de control del sistema (*SS*) es un factor que representa el nuevo punto de partida del sistema después de aplicar la salida del control difuso cuando sucede la entrega del mensaje *a*.

El conjunto de reglas que utilizan en el control difuso en la entrega del mensaje *a* tiene la forma:

$$\text{If } NC(a) \in A_i, \text{MC}(a) \in B_i \text{ y } FCCp(a) \in C_i, \text{ then } \S \in D_i$$

Donde:

- *NC(a)*, *MC(a)*, *FCCp(a)* y *§* son los parámetros del sistema de control difuso.
- $A_i$ = *{buenas, regulares, malas}* son las etiquetas lingüísticas de las *NC(a)*, representan el tráfico en los canales con respecto a las condiciones de la red.
- $B_i$ = *{buenas, regulares, malas}* son las etiquetas lingüísticas de las *MC(a)*, representan la disponibilidad de los recursos con respecto a los dispositivos móviles.
- $C_i$ = *{buena, regular, mala}* son las etiquetas lingüísticas de la *FCCp(a)*, representan la consistencia del sistema de acuerdo a las dependencias físicas, lógicas y espaciales.
- $D_i$= *{continuar, generar, generar y eliminar}* son las etiquetas lingüísticas de *§* para determinar si el sistema debe generar el *snapshot* distribuido.

### 4.5.1 Reglas del control difuso

A continuación se describe el conjunto de reglas que el generador distribuido considera para llevar a cabo el envío de petición del *snapshot* distribuido:

| | | | |
|---|---|---|---|
| IF *NC(a)= malas* | and *MC(a)= malas* | and $FCC_p(a)$= *mala,* | then *§=generar y eliminar* |
| IF *NC(a)= malas* | and *MC(a)= malas* | and $FCC_p(a)$= *regular*, | then *§=generar y eliminar* |
| IF *NC(a)= malas* | and *MC(a)= malas* | and $FCC_p(a)$= *buena*, | then *§=generar* |
| IF *NC(a)= malas* | and *MC(a)= regulares* | and $FCC_p(a)$= *mala*, | then *§=generar* |
| IF *NC(a)= malas* | and *MC(a)= regulares* | and $FCC_p(a)$= *regular*, | then *§=generar* |
| IF *NC(a)= malas* | and *MC(a)= regulares* | and $FCC_p(a)$= *buena*, | then *§=continuar* |
| IF *NC(a)= malas* | and *MC(a)= buenas* | and $FCC_p(a)$= *mala*, | then *§=generar* |
| IF *NC(a)= malas* | and *MC(a)= buenas* | and $FCC_p(a)$= *regular*, | then *§=generar* |
| IF *NC(a)= malas* | and *MC(a)= buenas* | and $FCC_p(a)$= *buena*, | then *§=continuar* |
| IF *NC(a)= regulares* | and *MC(a)= malas* | and $FCC_p(a)$= *mala*, | then *§=generar y eliminar* |
| IF *NC(a)= regulares* | and *MC(a)= malas* | and $FCC_p(a)$= *regular*, | then *§=generar y eliminar* |
| IF *NC(a)= regulares* | and *MC(a)= malas* | and $FCC_p(a)$= *buena*, | then *§=generar* |
| IF *NC(a)= regulares* | and *MC(a)= regulares* | and $FCC_p(a)$= *mala*, | then *§=generar* |
| IF *NC(a)= regulares* | and *MC(a)= regulares* | and $FCC_p(a)$= *regular*, | then *§=continuar* |
| IF *NC(a)= regulares* | and *MC(a)= regulares* | and $FCC_p(a)$= *buena*, | then *§=continuar* |
| IF *NC(a)= regulares* | and *MC(a)= buenas* | and $FCC_p(a)$= *mala*, | then *§=generar* |
| IF *NC(a)= regulares* | and *MC(a)= buenas* | and $FCC_p(a)$= *regular*, | then *§=continuar* |
| IF *NC(a)= regulares* | and *MC(a)= buenas* | and $FCC_p(a)$=*buena*, | then *§=continuar* |
| IF *NC(a)= buenas* | and *MC(a)= malas* | and $FCC_p(a)$= *mala*, | then *§=generar y eliminar* |
| IF *NC(a)= buenas* | and *MC(a)= malas* | and $FCC_p(a)$= *regular*, | then *§=generar y eliminar* |
| IF *NC(a)= buenas* | and *MC(a)= malas* | and $FCC_p(a)$=*buena*, | then *§=generar* |

| | | | |
|---|---|---|---|
| IF *NC(a)= buenas* | and *MC(a)= regulares* | and $FCC_p(a)$*= mala*, | then *§=generar* |
| IF *NC(a)= buenas* | and *MC(a)= regulares* | and $FCC_p(a)$*= regular*, | then *§=continuar* |
| IF *NC(a)= buenas* | and *MC(a)= regulares* | and $FCC_p(a)$*= buena*, | then *§=continuar* |
| IF *NC(a)= buenas* | and *MC(a)= buenas* | and $FCC_p(a)$*= mala*, | then *§=generar* |
| IF *NC(a)= buenas* | and *MC(a)= buenas* | and $FCC_p(a)$*= regular*, | then *§=continuar* |
| IF *NC(a)= buenas* | and *MC(a)= buenas* | and $FCC_p(a)$*= buena*, | then *§=continuar* |

## 4.6 Algoritmo

En primer lugar el algoritmo traduce los eventos de envío y recepción de cada mensaje de acuerdo al escenario que se presenta en un sistema de cómputo móvil (ver figura 10).

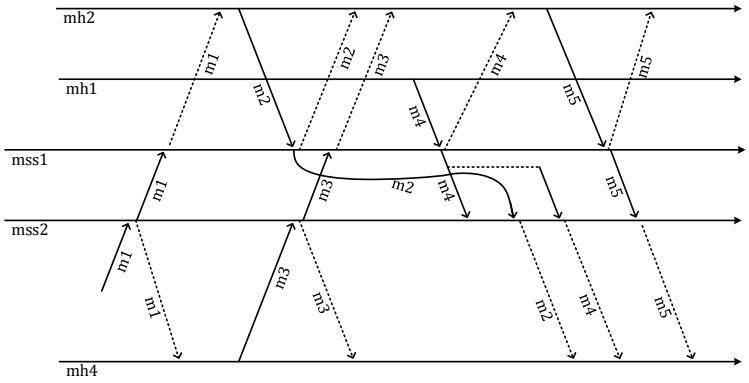


*Figura 10: Ejemplo de escenario de un sistema de cómputo móvil.*

En segundo lugar, el algoritmo verifica si el mensaje recibido por una *MSS* contiene la petición para generar el *snapshot* distribuido teniendo dos posibles casos.

1. Si contiene la petición, el algoritmo procesa el mensaje y posteriormente le indica a la *MSS* que debe transmitirlo.
2. Si no contiene la petición, el algoritmo procesa el mensaje, posteriormente calcula las variables de entrada utilizadas por el control difuso y la consistencia causal difusa. Enseguida determina las relaciones causales difusas y la consistencia causal difusa del mensaje recibido. Finalmente, el algoritmo evalúa mediante el control difuso el estado del sistema y las condiciones de la red para tomar la decisión de si la *MSS* debe enviar transmitir el mensaje con la petición para que se genere el *snapshot* distribuido.

### 4.6.1 Codificación del algoritmo

El algoritmo (ver anexo C) se encuentra dividido en secciones: la sección principal, la sección de *MSS*, la sección de *MH*, la sección de *snapshot*; en este apartado se describen las estructuras de datos de la sección de *MSS*, recordando que las *MSS* son las encargadas de enviar la petición para generar el *snapshot* distribuido.

#### 4.6.1.1 Estructura de datos

A continuación se describen las estructuras de datos utilizadas por el algoritmo.

*Estructura de los mensajes.* Formalmente, un mensaje $m$ recibido por una *MSS* en el algoritmo es una túpla $m = (s, i, t, j, H(m), \text{data})$, donde:

- $s$ es el identificador que muestra si el mensaje contiene la petición para generar el *snapshot* y el tipo de generación.
- $i$ es el identificador del proceso del *MH* que envió el mensaje $m$.
- $t = VT(p)[k]$ es el valor del reloj local del proceso $k$ cuando el mensaje $m$ fue trasmitido.
- $H(m)$ es el historial causal del mensaje $m$, contiene identificadores $(i, t, j)$ de los mensajes que le preceden causalmente al mensaje $m$. La información en $H(m)$ asegura la entrega causal del mensaje $m$. Esta estructura se construye antes de que el mensaje $m$ sea enviado por un *MH*.
- $j$ es el identificador del proceso de la *MSS* en el cual está asociado el *MH*.
- *data* es la estructura de la información interna del sistema.

*Estructura de los datos causales.* Éstas son las principales estructuras causales: $VT(p)$, $VFCR(p)$ y $CI(p)$. El tamaño de las estructuras $VT(p)$ y $VRCD(p)$ depende del número de *MH* mientras que el tamaño de la estructura $CI(p)$ depende del número de *MSS*.

- $VT(p)$ es el vector de tiempo. Cada proceso $p$ tiene un elemento $VT(p)[i]$ donde $i$ es un identificador del proceso ejecutado por el *MH*. El $VT(p)[i]$ representa el número más grande de mensajes que el $MH_i$ ha *visto* en un orden causal. La estructura $VT(p)$ contiene la vista local del historial causal del sistema por el proceso $p$.
- $CI(p)$ es la estructura de información de control donde $p$ es el identificador de la *MSS*, la estructura está compuesta por el conjunto de entradas $(k, t)$ las cuales identifican a los mensajes que han sido transmitidos por el proceso $k$ con valor $t$ de su reloj lógico. Además,

contiene información acerca del historial causal de cada proceso *k*. Cada entrada en *CI*(*p*) denota un mensaje cuya entrega conforme al orden causal no está garantizada.

- *VFCR*(*p*) es el vector que contiene las relaciones causales difusas. Cada proceso *p* tiene un elemento *VFCR*(*p*)[*i*] donde *i* es el identificador del proceso del *MH*. La estructura *VFCR*(*p*) se utiliza durante el cálculo de la *FCC* para determinar el estado del sistema conforme a la entrega de los eventos.

*Estructura de los datos auxiliares.* Hay trece estructuras de datos auxiliares que almacenan los parámetros que usan para implementar el mecanismo causal difuso. El tamaño de cada estructura es igual al número de *MH*. A continuación se describe la función de cada una.

- *VABw*(*p*) almacena el ancho de banda de cada proceso *p* que reside en una *MSS*.
- *VARtt*(*p*) almacena el retardo de cada proceso *p* que reside en una *MSS*.
- *VADr*(*p*) almacena el porcentaje de la tasa de perdida de datos de cada proceso *p* que reside en una *MSS*.
- *VIBw*(*p*) almacena el ancho de banda de cada proceso *p* que reside en un *MH*.
- *VIRtt*(*p*) almacena el retardo de cada proceso *p* que reside en un *MH*.
- *VIDr*(*p*) almacena el porcentaje de la tasa de perdida de datos de cada proceso *p* que reside en un *MH*.
- *VSp*(*p*) almacena la intensidad de la señal de cada proceso *p* que reside en un *MH*.
- *VBt*(*p*) almacena la capacidad de batería de cada proceso *p* que reside en un *MH*.
- *VRom*(*p*) almacena la capacidad de almacenamiento de cada proceso *p* que reside en un *MH*.
- *VCpu*(*p*) almacena la capacidad de procesamiento de cada proceso *p* que reside en un *MH*.
- *VRam*(*p*) almacena la capacidad de memoria de cada proceso *p* que reside en un *MH*.
- *GPN*(*p*) almacena el grado de ponderación de red de cada proceso *p* que reside en un *MH* incluyendo los valores de la *MSS* al que este asociado.
- *GPM*(*p*) almacena el grado de ponderación móvil de cada proceso *p* que reside en un *MH*.

*Estructura de datos del control difuso.* Estas estructuras son las variables de entrada para el control difuso.

- *T* almacena el valor de la función de membresía utilizada para calcular la distancia temporal para la *FCR*.



- *C* almacena el valor de la función de membresía utilizada para calcular la distancia causal para la *FCR*.
- *S* almacena el valor de la función de membresía utilizada para calcular la distancia espacial para la FCR.
- *FCC* almacena la consistencia causal difusa para ser utilizada como parámetro de entrada del control difuso.
- *NC* almacena las condiciones de la red para ser utilizada como parámetro de entrada del control difuso.
- *MC* almacena las condiciones de los móviles para ser utilizada como parámetro de entrada del control difuso.

### 4.6.2 Propiedades del algoritmo

A continuación se describen de manera general las propiedades del algoritmo para garantizar la generación de *snapshot* consistentes.

#### 4.6.2.1 Orden en la entrega de mensajes

Antes de que un mensaje sea enviado por algún dispositivo móvil, éste se etiqueta de acuerdo a un vector lógico (línea 102, ver Anexo C) con el objetivo de conocer las precedencias causales que tiene con otros mensajes.

| | |
|---|---|
| 102. | $m = (0, i, VT(p)[i]+1, j, H, data)$ |

El algoritmo mantiene esta condición de precedencia causal cuando la *MSS* recibe diferentes mensajes, ya que con la línea 40 del Anexo C, se puede comprobar si cada mensaje recibido debe ser procesado o debe mantenerse en espera hasta que la condición de entrega se cumpla.

| | |
|---|---|
| 40. | IF ($t == VT(p)[i] +1$) THEN |
| 41. | *delivery*($m$) |
| … | |
| 133. | ELSE |
| 134. | *wait*() |

Asimismo, los dispositivos móviles pueden comprobar si cada mensaje recibido debe ser procesado (línea 108, ver Anexo C) o debe mantenerse en espera para su posterior procesamiento (línea 120, ver Anexo C).

| | |
|---|---|
| 108. | ELSE IF ($t ==$ *VT*($p$)[$k$] + 1 AND $t' \leq$ *VT*($p$)[$k'$] $\forall$ ($k'$, $t'$) $\in$ *H*($m$))/* |
| 109. | *delivery*(m) |
| … | |
| 120. | ELSE |
| 121. | *wait*() |

De lo contrario, si los mensajes no se ordenan durante su recepción en cualquier proceso ocasionaría que cada *snapshot* generado sea inconsistente cumpliendo con la siguiente afirmación.

$$IF \exists\, m, m' \epsilon \S_k \mid Send(m) \rightarrow Send(m') \; and \; Recv(m') \rightarrow Recv(m)$$
$$THEN \S_k \; es \; inconsistente$$

Un ejemplo de este problema puede verse reflejado en la figura 11, ya que si $m_4$ se procesa antes que $m_2$ en la $MSS_2$, generar el *snapshot* posterior a esta situación seria inadecuado pues esto sería inconsistente para el sistema.

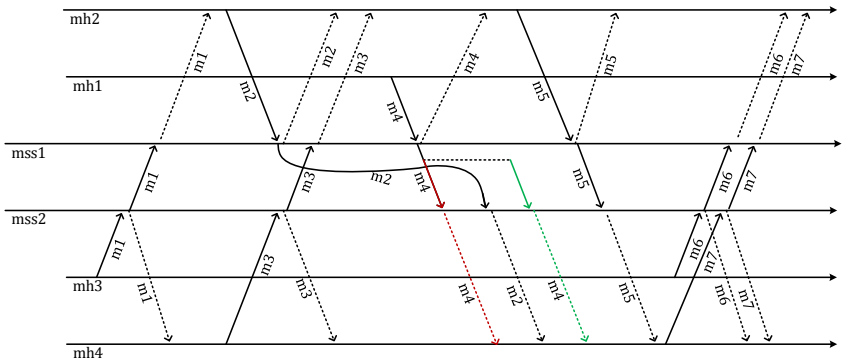


*Figura 11: Ejemplo del orden en la entrega de mensajes.* La flecha en guinda muestra una violación al orden sobre la entrega de mensajes, mientras que la verde mantiene este orden.

### 4.6.2.2 *Snapshot* libres de mensajes huérfanos

El mecanismo genera *snapshot* sin que alguno contenga un mensaje huérfano que origine el fenómeno del efecto dominó. Un mensaje huérfano es aquel donde su evento de recepción pertenece a un *snapshot* pero su evento de envío pertenece a algún sucesor de dicho *snapshot*, lo que causa que ambos *snapshot* sean inconsistentes, es decir, se cumple la siguiente afirmación.

$$IF \exists\, m \mid Recv(m) \epsilon \S_k \; and \; Send(m) \epsilon \S_{k+n} \; THEN \; m \; es \; un \; mensaje \; huérfano$$

Para evitar la presencia de mensajes huérfanos como en la figura 12, el algoritmo se apoya de la propiedad del orden en la entrega de mensajes. Si la *MSS* recibe un mensaje y éste puede contener una petición (línea 39, ver Anexo C) pero el mensaje aun no puede ser procesado este se mantiene en espera.

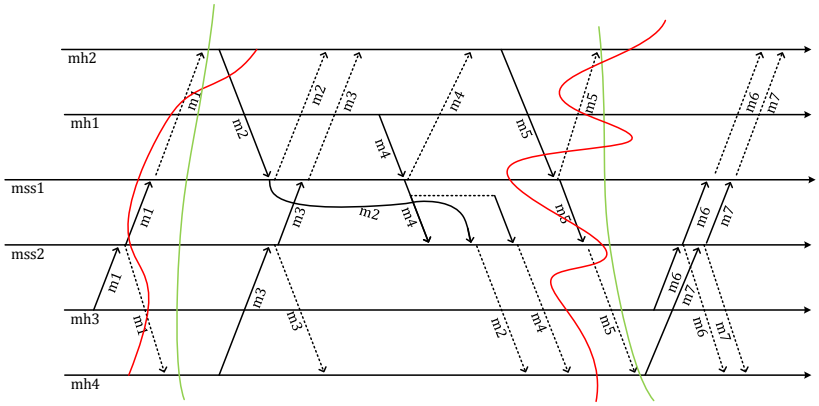


*Figura 12: Ejemplo de snapshot libre de mensajes huérfanos.* Las líneas rojas presenta la aparición de mensajes huérfanos que originan el efecto dominó. Las líneas verdes por otro lado presentan *snapshot* sin mensajes huérfanos.

Una vez cumplida la condición anterior y en caso de ser necesario, el algoritmo se asegura con las líneas 79 – 84 del Anexo C que las *MSS* generen su *snapshot* antes de retransmitir el mensaje que han procesado.

| | |
|---|---|
| 39. | IF $s = \{$*continuar*, *generar*, *generar y eliminar*$\}$ THEN |
| … | |
| 79. | IF $s == \{$*generar*, *generar y eliminar*$\}$ THEN |
| 80. | *snapshot*($s$) |
| 81. | IF $i \notin mss_j$ |
| 82. | ~~*m*~~ ”= ($-s$, $I$, $t$, $H(m)$) |
| 83. | *sending*(~~*m*~~ ”)/* return to $mss_k$ |
| 84. | END IF |
| … | |

Sin embargo, si una *MSS* recibe un mensaje proveniente de otra *MSS* y detecta que debe generar un *snapshot*, ésta se encarga de retornarle a la *MSS* que envió el mensaje para indicarle dicha decisión. De tal manera que con las líneas 96 – 100 del Anexo C, una *MSS* puede generar el *snapshot* aun después de haber procesado el mensaje.

| | |
|---|---|
| 96. | ELSE/* message returned |
| 97. | *snapshot*($s$) |

Asimismo, con las líneas 108 y 109 del Anexo C los *MH* generan su *snapshot* antes de procesar el mensaje que han recibido y con las líneas 105 y 106 del mismo anexo, los *MH* generan su *snapshot* después de haber procesado dicho mensaje.

| | |
|---|---|
| 105. | IF ($t \leq VT(p)[k]$ AND $t' \leq VT(p)[k'] \forall (k', t') \in H(m)$ AND $s > 0$) THEN/* message returned |
| 106. | *snapshot*($s$) |
| 107. | ELSE IF ($t == VT(p)[k] + 1$ AND $t' \leq VT(p)[k'] \forall (k', t') \in H(m)$)/* |
| 108. | IF $s ==$ {*generar*, *generar y eliminar*} THEN |
| 109. | *snapshot*($s$) |

Si el sistema hace una restauración y encuentra consecutivamente dos o más *snapshot* inconsistentes, la restauración se haría en cascada (efecto dominó) hasta que el sistema encuentre un *snapshot* consistente. Una situación menos favorable de este fenómeno es que la restauración del sistema llegue al estado inicial del mismo.

### 4.6.2.3 Manejo de mensajes en tránsito

Los mensajes en tránsito son la contraparte de los mensajes huérfanos, pues aquí su evento de envío pertenece a un *snapshot* y el evento de recepción pertenece a algún sucesor de dicho *snapshot*, de tal manera que se cumple la siguiente afirmación.

$$IF\ \exists\ m \mid Send(m)\ \epsilon\ \S_k\ and\ Recv(m)\ \epsilon\ \S_{k+n}\ THEN\ m\ es\ un\ mensaje\ en\ tránsito$$

Si estos mensajes no son reenviados inmediatamente después de haber realizado una restauración pueden llegar a generar información inconsistente al sistema. Para evitar este problema, el algoritmo nuevamente se apoya de la propiedad del orden en la entrega de mensajes, ya que al mantener dicho orden las *MSS* pueden conocer a los mensajes que aún se encuentran en tránsito.

Este proceso puede ser conocido tanto a nivel intercelda como a nivel intracelda de los *snapshot* que se hayan generado (líneas 124 – 132, ver Anexo C) a través de la variables *SCI* y *SVT* que utiliza el algoritmo.

| | |
|---|---|
| 124. | IF $dms > 1$ THEN |
| ... | |
| 127. | $SVT(p)$[cs] ← $VT(p)$ |
| 128. | $SCI(p)$[cs] ← $CI(p)$ /* only applicable to mss */ |
| 129. | ELSE/* |
| 130. | $SVT(p)[cs]$ ← $VT(p)$ |
| 131. | $SCI(p)[cs]$ ← $CI(p)$ /* only applicable to mss */ |

| | |
|---|---|
| 132. | END IF |

#### 4.6.2.4 Manejo de concurrencia de mensajes

El mecanismo es capaz de manejar la concurrencia de mensajes tanto a nivel intercelda como a nivel intracelda. Ya que de lo contrario se presentaría una inhibición de mensajes que causaría que el *snapshot* generado se vuelva inconsistente, es decir, se cumple la siguiente afirmación.

$$IF\ \exists\ m, m' \mid \left(Send(m) \parallel\ Send(m')\right)\ and\ \left(Recv(m) \rightarrow Recv(m')\ or\ Recv(m') \rightarrow Recv(m)\right)$$
$$THEN\ existe\ una\ inhibicion\ entre\ m\ y\ m$$

La inhibición de mensajes es un fenómeno que se produce cuando una *MSS* recibe dos mensajes concurrentes, es decir, mensajes que tienen en común un mensaje predecesor causal a ellos (ver figura 13). La *MSS* los procesa conforme a su vista local provocando que un mensaje preceda al otro al momento de ser retransmitidos. Para garantizar la concurrencia de los mensajes, el algoritmo se apoya de la propiedad del orden en la entrega de mensajes, de tal manera que las otras *MSS* al recibir el mensaje precedido lo mantienen en espera hasta que su predecesor sea recibido.

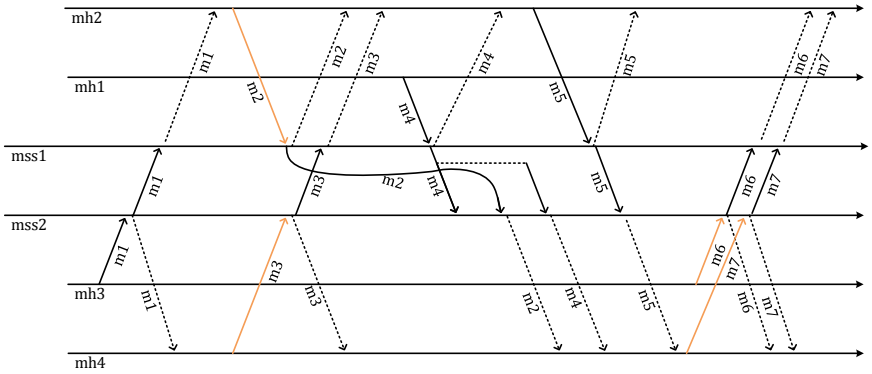


*Figura 13: Ejemplo de concurrencia de mensajes.* Las flechas naranja muestran la concurrencia a nivel intercelda (lado izquierdo) y a nivel intracelda (lado derecho).

## 4.7 Evaluación del rendimiento

A continuación se describen las siguientes notaciones para comparar el algoritmo del mecanismo causal difuso con respecto a otros mecanismos.

- *m*: número de *MSS*.
- *n*: número de *MH*.

- *N*: número de procesos.
- *b*: tamaño en bits.
- *B*: tamaño en Bytes.

### *4.7.1 Overhead* durante la generación de *snapshot*

Cuando alguna *MSS*, detecta que debe iniciar el proceso de generación de snapshot, ésta en el mismo mensaje de aplicación adjunta la petición de dicho proceso. Una vez hecho lo anterior la *MSS* transmite el mensaje hacia las otras *MSS* y estas a su vez retransmiten el mensaje a los *MH* que están asociados a ellas. Durante este proceso se asume que el *overhead* generado por el mecanismo es de aproximadamente $m * n * \log(n)_B$.

El overhead por mensaje que se propone en este mecanismo es menor en comparación con los que se han propuesto en [12, 13, 27, 33] dado que el esquema del algoritmo es de una sola fase ya que la petición se adjunta al mensaje que ha recibido la MSS, mientras que otros mecanismos utilizan *mensajes especiales* para completar el proceso de generación de snapshot. Asimismo, el vector de dependencia de procesos que utiliza el mecanismo causal difuso proviene del historial causal ($H(m)$) del mensaje a enviar, en los algoritmos que se han propuesto utilizan uno o más vectores de tamaño *N*. La figura 14 muestra la comparación entre los mecanismos que se han propuesto y el mecanismo causal difuso.

| | Lim [12] | Suri & Satiza [13] | Vutukuru *et al.* [27] | Nagpal & Kumar [33] | Mecanismo Causal Difuso |
|---|---|---|---|---|---|
| **Tipo de transmisión** | *Broadcast* | *Broadcast* | *Broadcast* | *Broadcast* | *Broadcast* |
| **Numero de fases para generar *snapshot*** | 1 | 2 | 1 | 3 | 1 |
| **Tamaño promedio de *overhead* utilizado** | $m * n * 3N_B$ | $m * n * (N_b + N_B)$ | $m * n * 3N_B$ | $4 * m * n * N_b + 3 * m * N_b$ | $m * n * \log(n)_B$ |

*Figura 14: Comparación del rendimiento del sistema.*

# CAPITULO V CONCLUSIONES Y TRABAJO A FUTURO

## 5.1 Conclusiones

A partir del estado del sistema y la aplicación de la lógica difusa fue posible detectar cuándo se debe llevar a cabo la generación del *snapshot* sin recurrir a una generación periódica y sin tener que hacer un uso innecesario de recursos tanto en los dispositivos móviles como en las estaciones de soporte móvil. Por lo que éstas últimas son las encargadas de llevar a cabo dicha proceso para enviarle las peticiones a los dispositivos móviles.

Además, con el algoritmo se evita la presencia de mensajes huérfanos para que no se origine el efecto dominó cuando se haga la restauración del sistema. Asimismo, se evita la inhibición de mensajes para que las estaciones de soporte móvil procesen los mensajes de acuerdo a la vista de los dispositivos móviles y no de acuerdo a su vista. También, el algoritmo le permite al sistema manejar a aquellos mensajes que se encontraban en tránsito para pueda retransmitirlos durante el proceso de restauración.

En comparación con otros trabajos, el mecanismo causal difuso requiere de una fase y de un *overhead* de tamaño $m * n * \log(n)$ para el proceso de generación de *snapshot* distribuidos sin recurrir a *mensajes especiales* para completar dicho proceso por lo que el *overhead* generado es menor en comparación los trabajos [12, 13, 27, 33].

En conclusión, el mecanismo causal difuso es capaz de generar *snapshot* de manera optimista sin interferir con las operaciones que el sistema se encuentre realizando, ya que a través del uso de las relaciones causales difusas pueden asociarse la consistencia causal difusa y la calidad de servicio para determinar el grado de afectación del estado del sistema y el momento en que se debe generar un *snapshot* distribuido.

## 5.2 Trabajo a futuro

Como trabajo a futuro se propone realizar una extensión del mecanismo causal difuso para su aplicación en sistemas distribuidos dentro de una red *Ad Hoc* (*MANET*), donde este tipo de sistemas están compuestos únicamente por dispositivos móviles, ver figura 14.

También se propone realizar una extensión del mecanismo causal difuso para su aplicación en sistemas de cómputo móvil del tipo multigrupo, donde los dispositivos móviles estén agrupados aun si estos pertenecen a una misma celda o a distintas celdas.

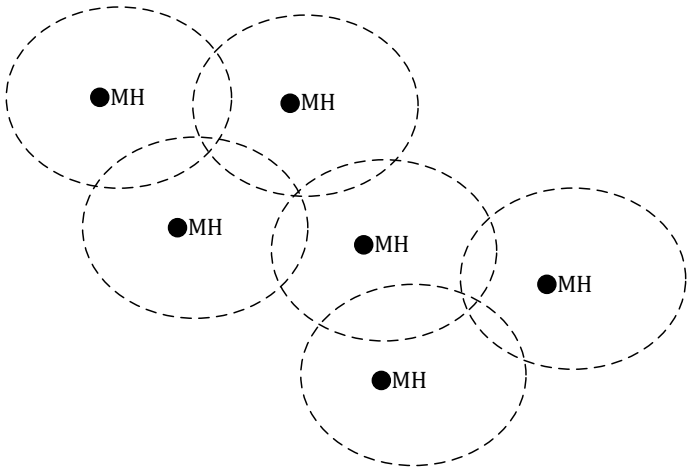


*Figura 15: Diagrama de una MANET.*

# REFERENCIAS


[1] R. Garg and P. Kumar, "A Review of Checkpointing Faul Tolerance Techniques in Distributed Mobile Systems," *International Journal on Computer Science and Engineering, 2(4),* pp. 1052-1063, 2010.

[2] P. Gupta, P. Kumar and A. K. Solanki, "Review of Some Minimum-procces Synchronous Checkpointing Schemes for Mobile Distributed Systems," *(IJCSE) International Journal on Computer Science and Engineering, 2(4),* pp. 1406-1410, 2010.

[3] R. Tuli and P. Kumar, "Analysis of recent checkpointing techniques for mobile computing systems," *International Journal of Computer Science & Engineering Survey (IJCSES), 2(3),* 2011.

[4] A. Khunteta and P. Kumar, "An Analysis of Checkpointing Algorithms for Distributed Mobile Systems," *(IJCSE) International Journal on Computer Science and Engineering, 2(4),* pp. 1314-1326, 2010.

[5] 3GPP, "Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); LTE; Quality of Service (QoS) concept and arcnitecture (3GPP TS 23.107 version 11.0.0 Release 11)," ETSI, 2012.

[6] 3GPP, "Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); LTE; End-to-end Quality of Service (QoS) concept and arcnitecture (3GPP TS 23.207 version 11.0.0 Release 11)," ETSI, 2012.

[7] P. Gahlan and P. Kumar, "Review of Some Checkpointing Algorithms in Distributed and Mobile Systems," *International Journal of Engineering Science and Technology, 2(6),* pp. 1594-1602, 2010.

[8] S. Kumar, R. K. Chauhan and P. Kumar, "Real Time Snapshot Collection Algorithm for Mobile Distributed Systems with Minimum Number of Checkpoints," *International Journal of Computer Applications, 2(9),* Junio 2010.

[9] S. Kumar, R. K. Chauhan and P. Kumar, "Minimum Process Error Recovery Algorithms for Mobile Distributed System using Global Checkpoint," *International Journal of Information Technology and Knowledge Management, 1(1),* pp. 23-33, 2008.

[10] P. Kumar, P. Gupta and A. Kumar Solanki, "Dealing with Frequent Aborts in Minimum-process Coordinated Checkpointing Algorithm for Mobile Distributed Systems," *International Journal of Computer Applications (0975 – 8887) 3(10),* pp. 7-12, 2010.

[11] P. Singh and G. Cabilic, "Successive Checkpointing Approach for Mobile Computing Environment," *Proceedings of the International Conference on Wireless Networks, ICWN '03,* 23-26 Junio 2003.

[12] S. Lim, "A Tunable Checkpointing Algorithm for the Distributed Mobile Environment," *IJCSI International Journal of Computer Science Issues, 8(6-3),* Noviembre 2011.

[13] P. K. Suri and M. Satiza, "An Efficient Checkpointing Protocol for Mobile Distributed Systems," *International Journal of Latest Research in Science and Technology, 1(2),* pp. 109-114, Julio-Agosto 2012.

[14] P. Kumar, R. Setiya and P. Gahlan, "Checkpointing Algorithms for Distributed Systems," *International Journal of Computing Science and Communication Technologies, 2(1),* 2009.

[15] R. Prakash and M. Singhal, "Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems," *IEEE Transaction on Parallel and Distributed Systems, 7(10),* pp. 1035-1048, Octubre 1996.

[16] N. Neves and W. K. Fuchs, "Adaptive Recovery for Mobile Environments," *Communications of the ACM, 40(1),* Enero 1997.

[17] G. Cao and M. Singhal, "Mutable Checkpoints: A New Checkpointing Approach for Mobile Computing Systems," *IEEE Transactions on Parallel and Distributed Systems, 12(2),* Febrero 2001.

[18] G. Cao and M. Singhal, "Checkpointing with mutable checkpoints," *Theoretical Computer Science, 290,* pp. 1127-1148, 2003.

[19] C.-Y. Lin, S.-C. Wang and S.-Y. Kuo, "An Efficient Time-Based Checkpointing Protocol for Mobile Computing Systems over Mobile IP," *Mobile Networks and Applications 8,* pp. 687-697, 2003.

[20] P. Kumar and A. Khunteta, "Anti-message Logging based Coordinated Checkpointing Protocol for Deterministic Mobile Computing Systems," *International Journal of Computer Applications (0975 – 8887), 3(1),* Junio 2010.

[21] R. Garg and P. Kumar, "A Nonblocking Coordinated Checkpointing Algortihm for Mobile Computing Systems," *IJCSI International Journal of Computer Science Issues, 7(3-3),* Mayo 2010.

[22] A. K. Panghal, M. K. Rana and P. Kumar, "Minimum-Process Synchronous Checkpointing in Mobile Distributed Systems," *International Journal of Computer Applications (0975-8887), 17(4),* Marzo 2011.

[23] R. Tuli and P. Kumar, "Minimum Proccess Coordinated Checkpointing Sheme for Ad Hoc Networks," *International Journal on AdHoc Networking Systems (IJANS), 1(2),* Octubre 2011.

[24] S. R. Kaware and P. L. Ramteke, "An Optimal Checkpoint Interval – a Novel Checkpointing Approach for mobile Consumer Devices," *International Journal of Advanced Research in Computer Science and Software Engineering, 4(3),* pp. 575-577, Marzo 2014.

[25] T. Park, N. Woo and H. Y. Yeon, "An Efficient Recovery Scheme for Mobile Computing Environments," *ICPADS 2001. Proceedings. Eighth International Conference on Parallel and Distributed Systems, 2001.,* pp. 53-60, 2001.

[26] A. K. Singh, "On Mobile Checkpointing using Index and Time Together," *World Academy of Science, Engineering and Technology, 8,* 2007.

[27] A. Vutukuru, B. Gupta, N. Mogharreban, S. Rahimi and L. Tallam, "A Domino-Effect Free Recovery Mechanism for Mobile Computing Environment," *Int'l Conf. Wireless Networks, ICWN'0,* pp. 361-367, 2009.

[28] C.-M. Lin and C.-R. Dow, "Efficient Checkpoint-based Failure Recovery Techniques in Mobile Computing Systems," *Journal of Information Science and Engineering, 17,* pp. 549-573, 2001.

[29] S. Kumar, R. K. Chauhan and P. Kumar, "Design and Performance Analysis of Coordinated Checkpointing Algortihms for Distributed Mobile Systems," *International Journal of Distributed and Parallel Systems (IJDPS), 1(1),* 2010.

[30] A. Agbaria and W. H. Sanders, "Distributed Snapshots for Mobile Computing Systems," *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications, 14(17),* pp. 177-186, Marzo 2004.

[31] S. K. Gupta, R. K. Chauhan and P. Kumar, "A Hybrid Coordinated Checkpointing Protocol for Deterministic Mobile Distributed Systems," *Journal of Applied Computer Science & Mathematics, 9(4),* 2010.

[32] S. Biswas and S. Neogy, "A Mobility-Based Checkpointing Protocol for Mobile Computing System," *International Journal of Computer Science & Information Technology (IJCSIT), 2(1),* Febrero 2010.

[33] M. Nagpal and P. Kumar, "Anti-message Logging based Check pointing Algorithm for Mobile Distributed Systems," *International Journal of Computer Applications (0975-8887), 69(14),* Mayo 2013.

[34] J. E. J. F. P. S. Lopez E., "A Fault-tolerant Causal Broadcast Algorithm to be Applied to Unreliable Networks," *Proc. 17th International Conference on Parallel and Distributed Computing and Systems,* pp. 465-470, 2005.

[35] L. A. Morales Rosales, Distributed Multimedia Synchronization Based on Fuzzy Causal Relations, Sta. Ma. Tonantzintla, Puebla: Instituto Nacional de Astrofísica, Óptica y Electrónica, 2009.

[36] M. Friedman, "Virtual Time and Global States of Distributed Systems," *Proceedings of International Workshop on Parallel and Distributed Algorithms,* Octubre 1988.

[37] A. Kiehn, P. Raj and P. Singh, "A Causal Checkpointing Algorithm for Mobile Computing Environments," *ICDCN 2014, LNCS 8314,* pp. 134-148, Enero 2014.

[38] A. Chaturvedi, S. S. Hussain and V. Kumar, "A Study of Mutable Checkpointing Approach to Reduce Overheads Associated with Coordinated Checkpointing," *The SIJ Transactions on Computer Networks & Communication Engineering (CNCE), 1(3),* 2013.

[39] P. Sharma and A. Khunteta, "A Survey of Checkpointing Algorithms in Mobile Ad Hoc Network," *Global Journal of Computer Science and Technology Network, Web & Security, 12(12),* 2012.

[40] S. Kalaiserlvi and V. Rajaraman, "A survey of checkpointing algortihms for parallel and distributed computers," *Sadhand, 25(5),* pp. 489-510, 2000.

[41] P. Kumar and P. Kumar, "Analysis of Some Minimum-process Coordinated Algorithms for Mobile Computing Systems," *TECHNIA – International Journal of Computing Science and Communication Technologies, 4(1),* 2011.

[42] S. Kumar, R. K. Chauhan and P. Kumar, "A Low Overhead Minimum Process Global Snapshot Collection Algoritms for Mobile Distributed Systems," *The International Journal of Multimedia & Its Applications (IJMA), 2(2),* Mayo 2010.

[43] K. M. Chandy and L. Lamport, "Distributed Snapshots: Determining Global States of Distributed Systems," *CM Transactions on Computer Systems, 3(1),* pp. 63-75, Febrero 1985.

[44] R. H. B. Netzer and J. Xu, "Necessary and Sufficient Conditions for Consistent Global Snapshots," *IEEE Transactions on Parallel and Distributed Systems, 6(2),* pp. 165-169, Febrero 1995.

[45] G. Li and L. Shu, "A Low-Latency Checkpointing Scheme for Mobile Computing Systems," Mayo 2005.

[46] P. Kumar and P. Gahlan, "A Low-Overhead Minimum Process Coordinated Checkpointing Algorithm for Mobile Distributed System," *International Journal of Computer Applications (0975 – 8887), 3(1),* Junio 2010.

[47] B. Gupta, Z. Liu and S. Koneru, "A Low-Overhead Non-Block Check Pointing and Recovery Approach for Mobile Computing Environment," *Advances and Applications in Mobile Computing,* Marzo 2012.

[48] B. Gupta, S. Rahimi and Z. Liu, "A New High Performance Checkpointing Approach for Mobile Computing Systems," *IJCSNS International Journal of Computer Science and Network Security, 6(5B),* Mayo 2006.

[49] P. K. Jaggi and A. K. Singh, "Log Based Recovery with Low Overhead for Large Mobile Computing Systems," *Journal of Information Science and Engineering 29,* pp. 969-984, 2013.

[50] J. Ahn, "Low-cost Checkpointing-based Rollback Recovery Algorithm Considering Scalability," *International Journal of Multimedia and Ubiquitous Engineering, 7(2),* Abril 2012.

[51] P. S. Mandal and K. Mukhopadhyaya, "Mobile Agent Based Checkpointing with Concurrent Inititations," *International Journal of Foundations of Computer Science, 18(5),* pp. 1107-1122, 2007.

[52] P. Kumar and R. Garg, "Soft-Checkpointing Based Coordinated Checkpointing Protocol for Mobile Distributed Systems," *IJCSI International Journal of Computer Science Issues, 7(3-5),* Mayo 2010.

[53] R. Tuli and P. Kumar, "The Performance of Soft Chekpointing Approach in Mobile," *Global Journal of Computer Science and Technology, 11(9) Version 1.0,* 15 Mayo 2011.

[54] S. I. Rodriguez Cardemil, Modelo de Calidad de Servicio para una Red de Datos HSDPA (High Speed Downlink Packet Access) para el Entorno Local, Santiago de chile: Universidad de Chile - Facultad de Ciencias Físicas y Matemáticas, 2009.

[55] R. C. Gass and B. Gupta, "An Efficient Checkpointing Scheme For Mobile Computing Systems," *In Proc. ISCA 13th Int. Conf. Computer Applications in Industry and Engineering,* pp. 323-328, 2000.

[56] E. Lopez Domínguez, S. E. Pomares Hernández and G. Rodríguez Gomez, "MOCAVI: An Efficient Causal Protocol for Cellular Networks," *JCSNSI nternational Journal of Computer Science and Network Security 8(1),* pp. 136-144, 2008.

[57] K.-F. Ssu, B. Yao, W. K. Fuchs and N. F. Neves, "Adaptive Checkpointing with Storage Management for Mobile Environments," *IEEE Transactions on Reliability 48(4),* pp. 315-324, 1999.

[58] A. Khunteta and P. Kumar, "A non-blocking minimum-process checkpointing protocol for deterministic mobile computing systems," *Journal of Theoretical and Applied Information Technology 17(1),* 2010.

[59] N. Sridhar and P. A. G. Sivilotti, "Lazy Snapshots," *PDCS,* 2002.

[60] G. Cao, "Designing Efficient Fault-Tolerant Systems on Wireless Networks," *Proc. of the Third IEEE Information Survivability Workshop (ISW-2000),* 2000.

[61] B. Gupta and S. Rahimi, "A Fast And Efficient Non-Blocking Coordinated Checkpointing Approach For Distributed Systems," *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications & Conference on Real-Time Computing Systems and Applications, PDPTA 2006,* 2006.

# ANEXOS

## *A.* Algoritmo de distancia causal para la transmisión de mensajes en sistemas móviles distribuidos (caso *broadcast*)

Con el fin de conocer el valor de la distancia causal entre dos mensajes ocurridos en un sistema móvil distribuido, en este trabajo se desarrolló el algoritmo que cumple con este objetivo.

### Estructura de datos

- $VT(p)$ es el vector time. El tamaño del $VT(p)$ es igual al número de procesos. Cada proceso $p$ tiene un elemento $VT(p)[i]$ donde $i$ es el identificador del proceso que reside en algún dispositivo móvil.
- $VT(p)[i]$ representa el número más grande de mensajes que el identificador $i$ ha visto en el orden causal de $p$. La estructura $VT(p)$ contiene la vista local del historial causal del sistema del proceso $p$.
- $H(m)$ es la información causal del mensaje $m$. Ésta estructura contiene identificadores de los mensajes $(k, t)$ que preceden causalmente al mensaje $m$. La información en $H(m)$ asegura la entrega causal del mensaje $m$.
- $CI(p)$ es la estructura de información de control. Ésta posee un conjunto de entradas $(m, m', m'', d)$, dónde:
    - $m = (i, t)$, representa un mensaje transmitido por el proceso $p_i$ en el tiempo local lógico $t = VT(p_i)[i]$.
    - $m' = IDR(m) = (i', t')$ representa un mensaje, transmitido por un proceso $i'$ en el tiempo local lógico $t'$, el cuál es recibido inmediatamente por el proceso $i$ después de recibir a $m$. tal que $m \rightarrow m'$.
    - $m'' = (i'', t'')$ representa un mensaje, transmitido por un proceso $i''$ en el tiempo local lógico $t''$, el cuál es el último mensaje recibido por el proceso $i$, tal que $m \rightarrow m''$.
    - $d$ es una variable que contiene la distancia causal entre $m$ y $m''$.

| | |
|---|---|
| 1. | **Initially** |
| 2. | $VT(p)[i] = 0 \forall\ i = 1 \ldots n \mid i \in$ mh /* Vector clock */ |
| | /* Data structure maintained for any mobile support station */ |
| 3. | $CI(p) \leftarrow \emptyset$ |

| | |
|---|---|
| | /* Data structure maintained for any mobile host */ |
| | |
| 4. | $H \leftarrow \emptyset$ |
| | /* Function receive for any mobile support station */ |
| 5. | **For each reception** *receive*(~~$m$~~) **at** $p$, ~~$m$~~ = ($i$, $t$, $H$(~~$m$~~), *content*) |
| 6. | **if not** ($t ==$ $VT(p)[i] + 1$ **and** $t' \leq VT(p)[i] \; \forall \; (i', t') \in H$(~~$m$~~)) **then** |
| 7. | *wait*() |
| 8. | **else** |
| 9. | **Delivery:** *delivery*(~~$m$~~) |
| 10. | $VT(p)[i] = VT(p)[i] + 1$ |
| 11. | **for all** ($m$, $m'$, $m''$, $d$) $\in CI(p_j)$ |
| 12. | **if** $m' == null$ **and** $m \in$ H(~~$m$~~) **then** |
| 13. | $m'$, $m'' = (i, t)$ |
| 14. | $d = d + 1$ |
| 15. | **end if** |
| 16. | **if** $m'' \in H$(~~$m$~~) **then** |
| 17. | $m'' = (i, t)$ |
| 18. | $d = d + 1$ |
| 19. | **end if** |
| 20. | **end for** |
| 21. | **for all** ~~$m'$~~ $\in H$(~~$m$~~) |
| 22. | **if** $\exists$($m$, $m'$, $m''$, $d$) $\in CI(p)$ \| ~~$m'$~~ $== m$ **then** |
| 23. | $CI(p_j) \leftarrow CI(p_j) \cup$ (~~$m'$~~, ($i$, $t$), ($i$, $t$), 1) |
| 24. | **for all** ($m$, $m'$, $m''$, $d$) $\in CI(p_j)$ \| ($IDR$(~~$m'$~~) $\neq$ ~~$m$~~) $== m''$, ~~$m'$~~ $\neq m$ |
| 25. | $m'' =$ ~~$m'$~~ |
| 26. | $d = d - 1$ |
| 27. | **end for** |
| 28. | **end if** |
| 29. | **end for** |
| 30. | $CI(p_j) \leftarrow CI(p_j) \cup$ (($i$, $t$), *null*, *null*, 0) |
| 31. | **if** $i \in mss_j$ **then** |
| 32. | ~~$m''$~~ = ($i$, $t$, $j$, $H$(~~$m$~~), *content*) |
| 33. | **Diffusion**: *send*(~~$m''$~~) /* to other mobile support station */ |
| 34. | **end if** |
| 35. | ~~$m'$~~ = ($i$, $t$, $H$(~~$m$~~), *content*) |
| 36. | **Diffusion**: *send*(~~$m'$~~) /* to all local mobile host including i */ |
| 37. | **end if** |
| | /* Function send for any mobile host */ |
| 38. | **For each diffusion of message** *send*(~~$m$~~) **at** $p_i$ |
| 39. | ~~$m$~~ = ($i$, $VT(p)[i]$+1, $H$, *content*) |
| 40. | **Diffusion**: *send*(~~$m$~~) /* to local MSS |
| | /* Function receive for any mobile host */ |
| 41. | **For each reception** *receive*(~~$m$~~) **at** $p$, ~~$m$~~ = ($i$, $t$, $H$(~~$m$~~), *content*) |
| 42. | **if not** ($t ==$ $VT(p)[i] + 1$ **and** $t' \leq VT(p)[i] \; \forall \; (i', t') \in H$(~~$m$~~)) **then** |
| 43. | *wait*() |

| | |
|---|---|
| 44. | **else** |
| 45. | **Delivery**: *delivery*($m$) |
| 46. | $VT(p)[i] = VT(p)[i] + 1$ |
| 47. | **if** $H == H(\cancel{m})$ **then** |
| 48. | $H \leftarrow (i, t)$ |
| 49. | **else** |
| 50. | $H \leftarrow H \cup (i, t)$ |
| 51. | **end if** |
| 52. | **end if** |

## B. Calidad de servicio para aplicaciones móviles

La calidad de servicio para aplicaciones móviles se establece de acuerdo a la norma establecida por la *3GPP* para sistemas digitales de telecomunicación celular en redes *UMTS* y *LTE*. La *3GPP* establece cuatro clases de servicio para definir las restricciones y limitaciones de las comunicaciones móviles, estas son: conversacional, afluente, interactiva y diferida.

Las aplicaciones distribuidas en tiempo real para nuestro caso particular como videojuegos, realidad aumentada, y realidad virtual entran en la clase conversacional debido a que esta clase preserva la relación o variación del tiempo entre las entidades que transmiten la información. Además los retardos estrictos y bajos (conocidos como patrones conversacionales) le sirven a este mecanismo para definir las condiciones de la red necesarias para los sistemas de cómputo móvil.

| Clase de servicio | **Aplicación** | **Tasa de datos (kbps)** | **Retraso** | **Perdidas (%)** |
|---|---|---|---|---|
| *Conversacional* | *Voz, videoconferencia, juegos interactivos* | *32 - 384* | *< 400 ms* | *< 5 FER* |
| Afluente | Audio, video, transferencia de datos | 32 - 384 | < 100 ms | < 3 FER |
| Interactiva | Mensajes de voz, navegación web, correo electrónico | 4 - 32 | < 4 s | < 1 FER |
| Diferida | Servicio de mensajes cortos, fax | < 32 | < 30 s | 0 |

## C. Codificación formal del algoritmo

A continuación se presenta la codificación formal del algoritmo.

| | |
|---|---|
| 1. | **Initially** |
| 2. | $VT(p)[i] = 0 \; \forall \; i = 1 \ldots n \mid$ i ∈ mh /* main section */ |
| 3. | $SVT(p)[l] = \emptyset \; \forall \; l = 1 \ldots x \mid x$ is nth snapshot taken by $p$ |
| 4. | $dms = 1$ |
| 5. | $cs = 0$ |
| | **Data structures maintained for any mobile support station** |
| 6. | $CI(p) \leftarrow \emptyset$ |
| 7. | $VFCR(p)[i] = 0 \; \forall \; i = 1 \ldots n \mid i \in$ mh |
| 8. | $VABw(p)[j] = 0 \; \forall \; j = 1 \ldots n \mid j \in$ mss |
| 9. | $VARtt(p)[j] = 0 \; \forall \; j = 1 \ldots n \mid j \in$ mss |
| 10. | $VADr(p)[j] = 0 \; \forall \; j = 1 \ldots n \mid j \in$ mss |
| 11. | $VIBw(p)[i] = 0 \; \forall \; i = 1 \ldots n \mid i \in$ mh |
| 12. | $VIRtt(p)[i] = 0 \; \forall \; i = 1 \ldots n \mid i \in$ mh |
| 13. | $VIDr(p)[i] = 0 \; \forall \; i = 1 \ldots n \mid i \in$ mh |
| 14. | $VSp(p)[i] = 0 \; \forall \; i = 1 \ldots n \mid i \in$ mh |
| 15. | $VBt(p)[i] = 0 \; \forall \; i = 1 \ldots n \mid i \in$ mh |
| 16. | $VRom(p)[i] = 0 \; \forall \; i = 1 \ldots n \mid i \in$ mh |
| 17. | $VCpu(p)[i] = 0 \; \forall \; i = 1 \ldots n \mid i \in$ mh |
| 18. | $VRam(p)[i] = 0 \; \forall \; i = 1 \ldots n \mid i \in$ mh |
| 19. | $VDc(p)[i] = 0 \; \forall \; i = 1 \ldots n \mid i \in$ mh |
| 20. | $GPN(p)[i] = 0 \; \forall \; i = 1 \ldots n \mid i \in$ mh |
| 21. | $GPM(p)[i] = 0 \; \forall \; i = 1 \ldots n \mid i \in$ mh |
| 22. | $SCI(p)[l] = \emptyset \; \forall \; l = 1 \ldots x \mid x$ is nth snapshot taken by $p$ |
| | **Data structure maintained for any mobile host** |
| 23. | $H \leftarrow \emptyset$ |
| | **First snapshot for all process** |
| 24. | $snapshot(0) \; \forall \; p$ |
| | **Checking network status for any mobile support station *p* with identifier of the process *j*** |
| 25. | ***status***$(p)$ /* |
| 26. | $VABw(p)[j] = get\_ABw()$ |
| 27. | $VARtt(p)[j] = get\_ARtt()$ |
| 28. | $VADr(p)[j] = get\_ADr()$ |
| | **Checking network and mobile status for any mobile host *p* with identifier of the process *i* when connecting or reconnecting to mobile support station** |
| 29. | ***status***$(p)$ |
| 30. | $VIBw(p)[i] = get\_IBw()$ |
| 31. | $VIRtt(p)[i] = get\_IRtt()$ |
| 32. | $VIDr(p)[i] = get\_IDr()$ |
| 33. | $VSp(p)[i] = get\_Sp()$ |
| 34. | $VBt(p)[i] = get\_Bt()$ |
| 35. | $VRom(p)[i] = get\_Rom()$ |
| 36. | $VCpu(p)[i] = get\_Cpu()$ |
| 37. | $VRam(p)[i] = get\_Ram()$ |
| | **For any message *m* received by a mobile support station *p* with identifier of the process *j*** |
| 38. | **receive**(m) in $p$ with $m = (s, i, t, k, H(m), data)$ /* MSS section */ |

| | |
|---|---|
| 39. | IF $s = \{continuar \mid generar \mid generar\ y\ eliminar\}$ THEN |
| 40. | IF ($t == VT(p)[i] + 1$) THEN |
| 41. | $delivery(m)$ |
| 42. | $VT(p)[i] = VT(p)[i] + 1$ |
| 43. | $CI(p_j) \leftarrow CI(p_j) \cup (i, t)$ |
| 44. | IF s == {*continuar*} THEN |
| 45. | $VDc(p)[i] = get\_Dc()$ |
| 46. | $C = max(VDc(p)[i], VBt(p)[i], VRom(p)[i])$ |
| 47. | $T = max(VIBw(p)[i] + VABw(p)[k], VIRtt(p)[i] + VARtt(p)[k], VIDr(p)[i] + VADr(p)[k])$ |
| 48. | $S = VSp(p)[i]$ |
| 49. | $VFCR(p)[i] = max(C, T, S)$ |
| 50. | $\forall (i', t') \in H(m)$ |
| 51. | $nGPN = 0$ |
| 52. | $nGPM = 0$ |
| 53. | $\forall (i', t', k') \in H(m)$ |
| 54. | $GPN(p)[i'] = VIRtt(p)[i'] + VIDr(p)[i'] + VARtt(p)[k'] + VADr(p)[k'])$ |
| 55. | $GPM(p)[i'] = VICpu(p)[i'] + VIRam(p)[i']$ |
| 56. | $nGPN = nGPN + GPN(p)[i']$ |
| 57. | $nGPM = nGPM + GPM(p)[i']$ |
| 58. | $GPN(p)[i] = VIRtt(p)[i] + VIDr(p)[i] + VARtt(p)[k] + VADr(p)[k]$ |
| 59. | $GPM(p)[i] = VCpu(p)[i] + VRam(p)[i]$ |
| 60. | $nGPN = nGPN + GPN(p)[i']$ |
| 61. | $nGPM = nGPM + GPM(p)[i']$ |
| 62. | $BNChannel = max(GPN(p)[i'] \forall (i', t') \in H(m), GPN(p)[i])$ |
| 63. | $BMChannel = max(GPM(p)[i'] \forall (i', t') \in H(m), GPM(p)[i])$ |
| 64. | $FCC = 0$ |
| 65. | $NC = 0$ |
| 66. | $MC = 0$ |
| 67. | $\forall (i', t', k') \in H(m)$ |
| 68. | $FCC = FCC + (VFCR(p)[i'] * (GPN(p)[i'] + GPM(p)[i']))$ |
| 69. | $NC = NC + ((2*VIBw(p)[i']/VIRtt(p)[i']) + (2*VABw(p)[k']/VARtt(p)[k']) + VIDr(p)[i'] + VADr(p)[k']) * (GPN(p)[i']/BNChannel)$ |
| 70. | $MC = MC + ((VBt(p)[i'] + VRom(\mathrm{p})[i']) / (VCpu(\mathrm{p})[i'] + VRam(\mathrm{p})[i']) * (GPM(p)[i']/BMChannel)$ |
| 71. | $FCC = FCC + (VFCR(p)[i] * (GPN(p)[i] + GPM(p)[i]))$ |
| 72. | $NC = NC + ((2*VIBw(p)[i]/VIRtt(p)[i]) + VIDr(p)[i]) * (GPN(p)[i]/BNChannel)$ |
| 73. | $MC = MC + ((VBt(p)[i] + VRom(p)[i]) / (VCpu(p)[i] + VRam(p)[i]) * (GPM(p)[i]/BMChannel)$ |
| 74. | $FCC = FCC/(nGPN + nGPM)$ |
| 75. | $NC = NC/nGPN$ |
| 76. | $MC = MC/nGPM$ |

| | |
|---|---|
| 77. | $s$ = /* Fuzzy Control ($\S$) */ |
| 78. | END IF |
| 79. | IF $s$ == {*generar* \| *generar y eliminar*} THEN |
| 80. | *snapshot*($s$) |
| 81. | IF $i \notin mss_j$ |
| 82. | ~~$m$~~" = (-$s$, $I$, $t$, $H$($m$)) |
| 83. | *sending*(~~$m$~~") /* return to $mss_k$ |
| 84. | END IF |
| 85. | ELSE |
| 86. | $dms = dms + 1$ |
| 87. | END IF |
| 88. | ~~$m$~~' = ($s$, $i$, $t$, $j$, $H(m)$, *data*) |
| 89. | *sending*(~~$m$~~') /* to all local mh including i |
| 90. | IF $i \in mss_j$ THEN |
| 91. | ~~$m$~~" = ($s$, $i$, $t$, $j$, $H$($m$), *data*) |
| 92. | *sending*(~~$m$~~") /* to other mss |
| 93. | END IF |
| 94. | ELSE |
| 95. | *wait*() |
| 96. | ELSE/* message returned |
| 97. | *snapshot*($s$) |
| 98. | ~~$m$~~' = (–$s$, $i$, $t$, $H$(~~$m$~~)) |
| 99. | *sending*(~~$m$~~') /* to all local mh |
| 100. | END IF |
| | **For any message *m* sent by a mobile host *p* with identifier of the process *i* associated to mobile support station with identifier of the process *j*** |
| 101. | **send()** |
| 102. | $m$ = (0, $i$, $VT(p)[i]$+1, $j$, $H$, *data*) /* MH section */ |
| 103. | *sending*($m$) |
| | **For any message *m* received by a mobile host *p* with identifier of the process *i* associated to mobile support station with identifier of the process *j*** |
| 104. | **receive**($m$) in $p$ with $m$ = ($s$, $k$, $t$, $j$, $H$($m$), *data*") |
| 105. | IF ($t \leq VT(p)[k]$ AND $t' \leq VT(p)[k'] \forall (k', t') \in H(m)$ AND $s > 0$) THEN/* message returned |
| 106. | *snapshot*($s$) |
| 107. | ELSE IF ($t == VT(p)[k] + 1$ AND $t' \leq VT(p)[k'] \forall (k', t') \in H(m)$)/* |
| 108. | IF $s$ == {*generar* \| *generar y eliminar*} THEN |
| 109. | *snapshot*($s$) |
| 110. | ELSE |
| 111. | $dms = dms + 1$ |
| 112. | END IF |
| 113. | *delivery*(m) |
| 114. | $VT(p)[k] = VT(p)[k] + 1$ |
| 115. | IF $H == H(m)$ THEN |
| 116. | $H \leftarrow (k, t, j)$ |
| 117. | ELSE |

| | |
|---|---|
| 118. | $H \leftarrow H \cup (k, t, j)$ |
| 119. | END IF |
| 120. | ELSE |
| 121. | *wait*() |
| 122. | END IF |
| | **For a snapshot taken by any process *p*** |
| 123. | ***Snapshot***(*s*) in *p*: /* snapshot section */ |
| 124. | IF $dms > 1$ THEN |
| 125. | $cs = cs + 1$ |
| 126. | $dms = 1$ |
| 127. | $SVT(p)[cs] \leftarrow VT(p)$ |
| 128. | $SCI(p)[cs] \leftarrow CI(p)$ /* only applicable to mss */ |
| 129. | ELSE/* |
| 130. | $SVT(p)[cs] \leftarrow VT(p)$ |
| 131. | $SCI(p)[cs] \leftarrow CI(p)$ /* only applicable to mss */ |
| 132. | END IF |
| 133. | IF *s* == {*generar y eliminar*} THEN /* only applicable to mh */ |
| 134. | $\forall vt$ in $SVT(p) \mid vt \neq SVT(p)[cs]$ |
| 135. | *delete*(*vt)* |
| 136. | END IF |