PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE INGENIERÍA

MAESTRÍA EN REDES DE COMUNICACIÓN

TESIS:

"IMPLEMENTACIÓN DE UN ALGORITMO PARA LA SELECCIÓN DE UN NODO LÍDER EN REDES DE SENSORES INALÁMBRICAS"

AUTOR:

PATIÑO ALBUJA DIANA IVETTE

TRABAJO PREVIO LA OBTENCION DEL TÍTULO DE:

MAGISTER EN REDES DE COMUNICACIONES

Quito, Noviembre 2016

RESUMEN

El estudio y diseño de algoritmos tolerantes a fallos dentro del campo de las aplicaciones distribuidas incluyen múltiples tareas complejas. Uno de los problemas más importantes y que requiere bastante capacidad de análisis es el consenso, el cual hace referencia a distintos procesos procurando alcanzar un acuerdo común. Este tipo de problema no puede ser resuelto fácilmente en un sistema donde existe la posibilidad que los procesos fallen. Los profesores Tushar Deepak Chandra y Sam Toueg con la finalidad de brindar una solución a este tipo de inconveniente plantearon los detectores no fiables de fallos.

El presente trabajo investigativo realiza un análisis y un estudio del algoritmo detector no fiable de fallos Omega planteado por los profesores Tushar Deepak Chandra y Sam Toueg, el mismo que incluye un modelo fallo/recuperación (Crash-Recovery), cuya característica principal se centra en mantener la disponibilidad y el correcto funcionamiento de la red cuando un nodo falla sin que esto afecte a todo el sistema.

Específicamente el enfoque se centra en el desarrollo de un simulador que permita observar el funcionamiento del algoritmo detector de fallos Omega en sistemas en donde los procesos puedan fallar y posteriormente recuperarse, en dichos casos se ha demostrado que se puede resolver mediante un consenso.

Se realiza un estudio del algoritmo detector de fallos Omega aplicando el modelo de fallo y recuperación.

Luego de un exhaustivo análisis y estudio del algoritmo, se procede con la implementación del mismo, mediante un simulador desarrollado en lenguaje C#.

Como parte final de la investigación se presenta las conclusiones y recomendaciones, en las cuales se expone los resultados obtenidos de toda la investigación, y recomendaciones para futuros estudios o implementaciones de esta tecnología.

ABSTRACT

The design and the algorithm verification and tolerant distributed applications to shortcomings are complex tasks. Inside their study, several types of problems have been identified.

One of the most important is the Consent, the problem of several processes trying to agree a common decision. The problem of the Consent cannot be resolved in asynchronous systems where the processes can fail. To solve, Chandra and Toueg proposed the non reliable detectors of shortcomings.

The present investigation is centered in the study of the non reliable detector of shortcomings Omega proposed by Chandra and Toueg, in the pattern of failure-and-recovery system (Crash-Recovery).

Plus specifically we center ourselves in the design of an algorithm that implements this detector of shortcomings in models where the processes can fail and then to recover, for which it has been demonstrated that the Consent can be solved.

The detector of shortcomings Omega is studied for the failure-and-recovery pattern.

After an exhaustive analysis and study of the algorithm, you proceed with the implementation of the same one by means of a pretender developed in language C #.

As final part of the investigation it is presented the conclusions and recommendations, in which it is exposed the obtained results of the whole investigation, and recommendations for future studies or implementations of this technology.

Índice de Contenidos

AGRADECIMIENTO

Agradezco a Dios por todas las bendiciones recibidas y por haberme permitido terminar con éxito una etapa más en mi vida.

A mi padre, por todo su amor, preocupación y apoyo durante este largo camino tanto en mis estudios como en mi vida, por ser ejemplo de perseverancia, dedicación, responsabilidad y sobre todo porque ni un solo instante he dejado de escuchar su voz de aliento que me ha permitido levantarme y enfrentar los retos con fortaleza y valentía.

De igual forma un agradecimiento muy especial a ese ángel que me dio la vida, que siempre será la luz de mi vida y que desde el cielo estará a mi lado cuidándome, mi madre.

Agradezco también a mis amigos que a lo largo de toda la trayectoria de la vida pude conocer, a aquellos que a través de los años me han acompañado y han compartido conmigo momentos únicos que los guardaré en mi mente y mi corazón.

De manera especial al Ing. Carlos Egas, Director del Proyecto, por el tiempo brindado, por la gestión realizada y por toda la orientación recibida para la culminación de esta meta.

DEDICATORIA

La presente investigación lo dedico a toda mi familia, que gracias a su apoyo incondicional logré concluir la culminación de la misma y de manera muy especial a mis padres por toda su paciencia, y que gracias a sus sabios consejos han sabido formarme como una persona con buenos sentimientos, hábitos y valores.

A mi hermana que ha estado junto a mí brindándome su apoyo en todo momento.

De igual forma a aquellas personas que aunque no estén físicamente conmigo, seguro han estado acompañándome y guiándome a través de los años.

Mi esfuerzo y dedicación es para todas aquellas personas que me han brindado su confianza y que día a día me han impulsado a seguir adelante.

CAPITULO I: PROBLEMA

1. INTRODUCCIÓN

Las redes WSN incluyen aplicaciones para la recolección de datos confiables, reduciendo en forma significativa la intervención del hombre. Dentro de este tipo de redes se pueden encontrar diferentes tipos de problemas como la comunicación, la medición, errores de la fuente de energía, choques entre otros. La redundancia de nodos sensores, permite proporcionar a la red un cierto nivel de tolerancia a fallos. (Saavedra Cano, 2015)

La información del sensor es receptada por algunos nodos de la red denominados agregadores. El objetivo de este proyecto es el estudio y la simulación de un algoritmo para la selección de un nodo líder en una red WSN “Wireless Sensor Network” dicho nodo será el encargado de realizar la recolección de datos. Para la selección del nodo líder se utiliza el algoritmo detector de fallos Omega propuesto por los profesores [Tushar Deepak Chandra, Hadzilacos, Sam Toueg 1996] el mismo que permite que la red siga en funcionamiento cuando un nodo falla. El algoritmo proporciona un mecanismo para la selección de un nodo líder en una red WSN con tolerancia a fallos.

1.1 PLANTEAMIENTO DEL PROBLEMA

En la última década ha habido una mayor motivación en desarrollar redes inalámbricas mediante sensores. Sus usos son diversos entre ellos podemos citar los siguientes: evolución y análisis de tornados, monitoreo, seguridad de hogares, centros comerciales y edificios públicos, domótica, entre otros. A medida que existe un incremento de estas aplicaciones el ser humano va dependiendo más de su uso día a día.

Las redes wsn están compuestas por múltiples nodos de sensores los mismos que se encuentran dispersos, cada dispositivo incluye componentes de comunicación y procesamiento, cuya principal función es el monitoreo del ambiente, en espacios cerrados como en abiertos, para luego realizar un envío de la data coleccionada a una base principal en donde posteriormente será procesada.

Uno de los principales problemas con este tipo de redes es el aprovechar de una forma óptima el consumo de energía en los nodos para alargar el mayor tiempo posible la vida de la red, tomando en cuenta que cada uno de los nodos utiliza como principal fuente de energía la batería.

Economizar la distancia de las comunicaciones es un factor primordial a tomar en cuenta dentro de una red. [5]

Debido a que el concepto de redes wsn es nuevo y no existe un estándar que permita responder a las necesidades planteadas, resulta beneficioso analizar un algoritmo que

permita efectuar un ahorro de energía y que de igual forma pueda ser usado en aplicaciones como un nodo agregador de datos.

Una red WSN está compuesta de múltiples nodos inalámbricos y de un Gateway, cuya característica principal es la conexión con el Internet o red remota.

En el momento que exista un fallo en el Gateway, toda la red perdería la conexión con el Internet, de igual manera si un nodo empieza a fallar el rendimiento y desempeño de la red se verá afectada, es por esta razón que es de indispensable importancia tener varios nodos Gateway de respaldo, para que cuando exista un fallo del nodo principal cualquiera de ellos entre en funcionamiento.

Existen varios mecanismos que pueden utilizarse para seleccionar el nodo gateway de respaldo sin embargo después de un estudio y análisis de los algoritmos de enrutamiento se determinó que el Algoritmo Omega es una buena opción por cuanto es independiente de la topología lo cual ahorra tiempo y optimiza los recursos de la red, en la presente investigación se realizó una adaptación del algoritmo Omega reemplazando los procesos por nodos, este algoritmo permite seleccionar de los nodos Gateway de respaldo un líder en caso que el nodo inicial falle y su función principal será la transmisión de la información, logrando que la red permanezca funcionando y evitando pérdida de los datos.

El criterio con el cual se seleccionará el nodo líder de respaldo será aquel cuyo estado de la batería sea el más alto de todos los nodos que conforman la red.

1.2 OBJETIVOS

1.2.1 Objetivo General

- Implementación de un algoritmo para la selección de un Nodo líder en Redes de Sensores Inalámbricas

Objetivos Específicos

- Implementar el algoritmo OMEGA para la selección del nodo líder
- Realizar un estudio del funcionamiento del algoritmo OMEGA y proceder a adaptarlo a IPv6.
- Implementar el algoritmo OMEGA para la selección de un nodo líder mediante un simulador desarrollado en lenguaje de programación C#
- Elaborar pruebas de desempeño del algoritmo basado en el detector de fallos Omega.
- Probar que el algoritmo con las funciones adaptadas a IPv6 funcionen correctamente.
- Realizar un informe con los resultados obtenidos de las pruebas.

1.3 JUSTIFICACIÓN

El vertiginoso mundo de la tecnología ha influido en el número de aplicaciones desarrolladas para diversas aéreas de la ciencia como: monitoreo de eventos aplicados en la agricultura, monitoreo permanente sin necesidad de la intervención hombre, medicina, biología, monitoreo de construcciones civiles, prevención de desastres naturales, estudio de los estados del tiempo meteorológico, detección de incendios forestales en tiempo real, domótica entre otros. Las redes wsn facilitan el estudio y el trabajo en dichas áreas, de igual forma permite prevenir accidentes futuros.

Por todas estas razones han sido identificadas como una de las tecnologías más prometedoras por revistas especializadas y diferentes analistas tecnológicos, entre los cuales se puede citar al observatorio tecnológico del MIT [1].

Esto se debe al desarrollo del cual han sido objetos dichas plataformas inalámbricas en lo relacionado a desempeño, asequibilidad, optimización, disponibilidad e interoperabilidad.

La principal finalidad de las redes WSN radica en sustituir sensores de alta complejidad - (que incluyen costos elevados, limitados en su número e infraestructura de comunicaciones) por un conjunto de dispositivos sensores más sencillos y que resultan mucho más económicos. [3]

Las aplicaciones WSN usan diversos tipos de algoritmos de enrutamiento. Un algoritmo de enrutamiento es un elemento del software encargado de elegir la línea de salida por la cual será transmitido un paquete de entrada. [4] Existen 2 tipos de enrutamiento: Estático y Dinámico.

El enrutamiento estático basa sus decisiones de enrutamiento en la decisión de que ruta se usará para llegar de I a J. Cambia sus decisiones de enrutamiento para reflejar los cambios de topología. [4]

El enrutamiento estático por la trayectoria más corta construye un grafo de la subred, en el cual cada nodo está representado por un enrutador y cada arco del grafo por una línea de comunicación denominada enlace. El algoritmo busca en el grafo la trayectoria más corta entre todos. [4]

En el algoritmo estático de inundación, cada paquete de entrada es transmitido por cada una de las líneas de salida, menos por la que llegó. Es evidente que esto genera una gran cantidad de paquetes duplicados. [4]

En el algoritmo dinámico por vector de distancia cada enrutador contiene una tabla de enrutamiento indizada que tiene un registro de cada enrutador de la subred. En la práctica posee un problema serio; si bien es cierto que existe una convergencia en la respuesta correcta, puede hacerlo en forma muy lenta. [4]

El algoritmo dinámico por estado de enlace detecta a sus vecinos, relaciona sus direcciones de red, mide el tiempo de retardo de cada uno, elabora un paquete con todo su aprendizaje, envía el paquete a los demás enrutadores y calcula la trayectoria más corta. Las tablas no sólo consumen memoria sino que necesitan mayor tiempo

de procesamiento y ancho de banda para transmitir un informe del estado entre enrutadores. [4]

La correcta selección de un algoritmo de enrutamiento permitirá reducir el consumo de energía, encontrar la ruta más óptima para la transferencia de datos minimizando el tiempo y la velocidad de transmisión de los mismos.

La importancia de este proyecto radica en el estudio y la simulación del algoritmo detector de fallos Omega, considerando que es una buena alternativa por cuanto es independiente de la topología, no utiliza una tabla de enrutamiento, optimizando de esta forma los recursos de la red, su principal función es la selección de un nodo líder de respaldo el cual será el encargado de transmitir los datos cuando el nodo inicial empiece a fallar, impidiendo así una pérdida de datos y una pérdida de conectividad en la red.

CAPITULO II: MARCO TEORICO

2.1 Algoritmo de Fallos Omega

El objetivo principal de este proyecto es el estudio e implementación del algoritmo para la selección de un Nodo Líder en Redes de Sensores Inalámbricas basado en el detector de fallos Omega propuesto por el profesor Tushar Deepak Chandra y su tolerancia y recuperación frente a fallos del sistema.

El detector de fallos omega (Ω) proporciona funciones para la elección de un líder tomando en cuenta únicamente los nodos correctos. Este algoritmo garantiza que todos los sensores de una región estén de acuerdo en un nodo común.

El Detector de Fallos Omega permite seleccionar un nodo líder común dentro de una región tomando en cuenta para este proceso la fiabilidad de los sensores. Un nodo sensor es un aspirante a ser seleccionado como líder si no contiene errores en el transcurso de sus operaciones. Este algoritmo garantiza que de un grupo de sensores será elegido como líder uno solo. Este nodo tendrá como función principal la recolección de los datos dentro de su área de cobertura.

Un nodo es elegido como el líder para actuar como un controlador centralizado de la red. El propósito de elección del líder es de elegir un nodo que se encargará de coordinar las actividades del sistema, el líder es elegido en base a algún criterio. La condición para la elección del nodo líder requiere que un solo nodo sea elegido y, finalmente, convertirse en el líder del sistema distribuido.

La información es enviada a los diferentes nodos mediante la transmisión de mensajes hasta que se llegue a un acuerdo. Una vez que se toma la decisión, un nodo es elegido como el líder.

2.2 Introducción

El detector de fallo Omega (Ω), ofrece una funcionalidad de elección de un proceso líder eventual, es decir, conforme avance en el tiempo todos los procesos correctos confían permanentemente en el mismo proceso correcto. [16][17] En otras palabras, el algoritmo Omega está enfocado a procesos correctos de los cuales se elige a uno de ellos como líder.

Se dice que un algoritmo posee una propiedad de comunicación eficiente **(communication- eficient),** si existe un número de enlaces limitado por n, siendo n el número de procesos en el sistema, por lo tanto un algoritmo es tolerante a fallos si eventualmente deja de enviar mensajes a causa de procesos fallidos. [16][17]

El Algoritmo Omega Ω proporciona una funcionalidad relacionada a la elección de un proceso líder eventual [16]. En este sentido, se ha demostrado que no es factible la ejecución del algoritmo Omega Ω con recuperación a fallos, sin la existencia de una mayoría de procesos correctos. En base a lo mencionado anteriormente, el número de procesos se desconoce, cada proceso ***pi*** sólo conoce su propia identidad y el número total de procesos en el sistema, es decir, ***i*** y ***n*** respectivamente; pero ***pi*** no conoce la identidad del resto de procesos del sistema.

Es necesario señalar también que un algoritmo distribuido tiene la propiedad “Quietud de Choque” o en inglés denominada “Crash-Quiescence”, la cual no envía mensajes a procesos dañados o caídos. [18] [17].

Un sistema con desconocimiento de los procesos que lo conforman, permite ejecutar a cada proceso el algoritmo Omega Ω sin tener que conocer inicialmente los identificadores de todos los procesos; sólo conociendo su propio identificador y el número total de procesos en el sistema. Por lo tanto, se puede ejecutar el mismo código del algoritmo Omega Ω en escenarios con diferentes procesos sin tener que cambiar de forma estática el conjunto de direcciones de procesos que participan en cada ejecución. Esto es posible porque los procesos aprenden dinámicamente sobre la existencia de otros procesos mediante la recepción de mensajes. Se debe tener en cuenta que esta reducción de conocimiento inicial supone también que si el mismo proceso se bloquea o falla antes de enviar un mensaje, el resto de procesos no serán capaces de hallar a su existencia como miembro del sistema.

Por ejemplo, supóngase que se tiene un entorno P2P (Peer to Peer) en Internet, donde cada servidor es identificado por una dirección IP (Internet Protocol). También se conoce que el número de servidores asciende a 10, en los sistemas donde se conoce el número de miembros, cada proceso tiene que conocer cada una de las direcciones IP de los distintos servidores, es decir, las 10 direcciones IP previamente asignadas a cada servidor, antes de iniciar la ejecución del algoritmo Omega Ω para elegir a algunos de ellos como líder. En los sistemas con desconocimiento de los miembros que los conforman, el líder puede ser elegido sin

este conocimiento, aprendiendo las direcciones IP de los mensajes recibidos de otros procesos [17].

En **“Communication-efficient and crash-quiescent Omega with unknown membership” [17]** se muestra que el algoritmo Omega Ω se puede implementar sin el conocimiento de la identidad del resto de procesos del sistema, lo que hace posible implementar un detector de fallos de tal manera que cada proceso correcto finalmente solo confíe en procesos correctos, en lugar de procesos fallosos.

2.3 Detectores de fallo poco fiables

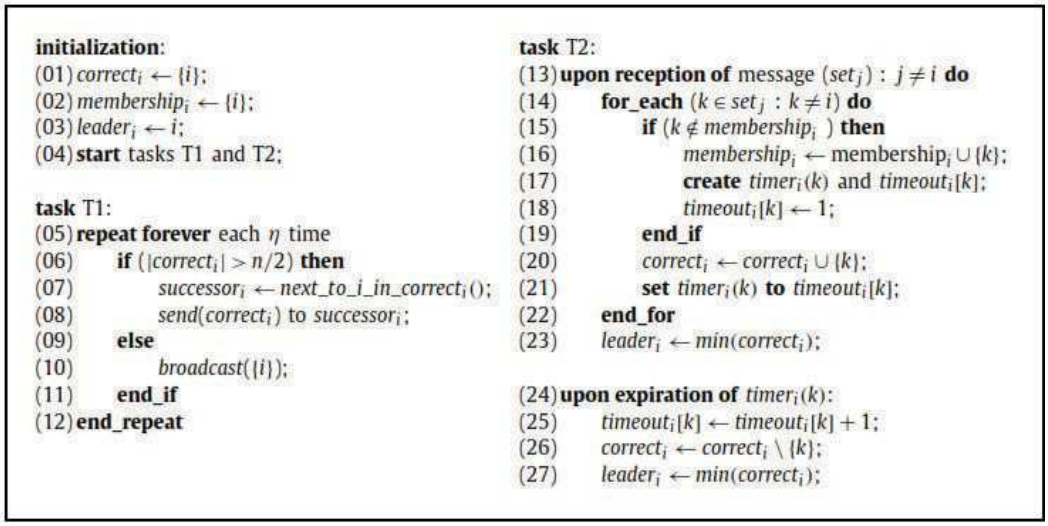
Un detector de fallos poco fiable es un mecanismo que proporciona información (posiblemente errónea) sobre las fallas de un proceso. Cuando se consulta, el detector de fallos devuelve una lista de procesos fallidos (presuntos procesos). Los detectores de fallos se caracterizan por tener dos propiedades: **la integridad y exactitud (Completeness and Accuracy). Integridad** se caracteriza por la capacidad que tiene el detector de fallos para sospechar de cada proceso incorrecto (procesos que en realidad fallaron), mientras que la **precisión o exactitud** se caracteriza por la capacidad de no sospechar de procesos correctos.

2.4 Algoritmo Omega *P ◇

El profesor Chandra en **“Communication-efficient and crash-quiescent Omega with unknown membership” [17]**, planteó el algoritmo detector de fallos Omega el cual asegura que de todo el conjunto de procesos correctos se seleccione como líder siempre un proceso correcto. [16].

Los profesores Tushar Deepak Chandra y Sam Toueg determinan al algoritmo detector de fallos Omega como un sistema casi perfecto por cuanto hay un tiempo después del cual cada uno de los procesos correctos únicamente desconfía de los procesos defectuosos [17]. En este modelo el descubrimiento de los procesos se va complementando poco a poco por cuanto al inicio no se conoce el número ni los elementos del sistema, estos datos son desconocidos, es por este motivo que es imposible señalar los procesos defectuosos. Si existiera un conocimiento del número de procesos ***pi*** fallosos al inicio, antes del envío o recepción de cualquier mensaje, no existiría la posibilidad que otros procesos ***pj*** durante el transcurso puedan aprender acerca de los demás procesos ***pi*** en el sistema, imposibilitando a los proceso ***pj*** distinguir a los procesos ***pi*** dudosos [16].

2.5 Modelo del Algoritmo Omega



```
initialization:
(01) correct_i ← {i};
(02) membership_i ← {i};
(03) leader_i ← i;
(04) start tasks T1 and T2;

task T1:
(05) repeat forever each η time
(06)     if (|correct_i| > n/2) then
(07)         successor_i ← next_to_i_in_correct_i();
(08)         send(correct_i) to successor_i;
(09)     else
(10)         broadcast({i});
(11)     end_if
(12) end_repeat

task T2:
(13) upon reception of message (set_j) : j ≠ i do
(14)     for_each (k ∈ set_j : k ≠ i) do
(15)         if (k ∉ membership_i ) then
(16)             membership_i ← membership_i ∪ {k};
(17)             create timer_i(k) and timeout_i[k];
(18)             timeout_i[k] ← 1;
(19)         end_if
(20)         correct_i ← correct_i ∪ {k};
(21)         set timer_i(k) to timeout_i[k];
(22)     end_for
(23)     leader_i ← min(correct_i);

(24) upon expiration of timer_i(k):
(25)     timeout_i[k] ← timeout_i[k] + 1;
(26)     correct_i ← correct_i \ {k};
(27)     leader_i ← min(correct_i);
```

Figura 2.1 Algoritmo Omega en un sistema donde cada proceso inicialmente solo conoce su identidad y n (código para pi procesos).

Se resumirá todo el proceso del algoritmo Omega en la Figura 2.1 y se explicará a detalle su funcionamiento. Para esto se considera un sistema ***S*** compuesto por un conjunto finito de ***n*** procesos. Los identificadores de cada uno de los procesos están totalmente ordenados, pero no tienen que ser consecutivos.

Se denotan como ***pi, pl, pr,*** etc., se supone que los procesos sólo conocen su propio identificador y cuál es el número de procesos ***n*** dentro del sistema ***S***, sólo tienen que conocer la mayoría, es decir, n/2. Por ejemplo, el proceso ***pi*** inicialmente sólo sabe acerca de ***i*** y ***n***. Los procesos se comunican únicamente mediante el envío y recepción de mensajes. Se asume que los procesos tienen una primitiva de difusión denominada **"broadcast primitivo",** misma que será utilizada para enviar mensajes (llamada de broadcast (m)).

Esta primitiva permite a un proceso enviar simultáneamente el mismo mensaje m al resto de procesos en el sistema (es decir, el mismo mecanismo utilizado en Ethernet o en IP-Multicast). Además, los procesos también tienen la posibilidad de utilizar la primitiva send (llamada send (m) a q), la cual permite enviar el mensaje ***m*** para que lo procese solamente ***q.*** [17] [20].

Un proceso puede fallar por colisionar o estar errado de forma permanente. Se dice que un proceso es correcto si no falla, y que un proceso es defectuoso si falla. Se utiliza ***C*** para denotar el conjunto de procesos correctos y ***F*** para denotar el conjunto de procesos defectuosos. Se considera que en un sistema ***S*** puede existir cualquier número de procesos fallidos y que este número no se conoce a priori.

La Figura 2.1 permite efectuar el algoritmo Omega Ω en un sistema ***S,*** donde se desconoce el número de miembros. Si la mayoría de los procesos son correctos, se demostrará que este protocolo es de comunicación eficiente.

Al inicio cada uno de los procesos ***pi*** realiza un broadcast en repetidas ocasiones para mostrar que el proceso se encuentra activo como se muestra en la línea número 10 de la Figura 2.1. Con el objetivo de saber qué procesos se encuentran en el sistema, los procesos ***pi*** forman parte de un conjunto de miembros ***i,*** que al inicio únicamente poseen su mismo proceso. El conjunto ***Correct i*** contiene los procesos que considera que están activos, siempre por lo menos se contiene a sí mismo. Cuando la mayor parte de los procesos están en estado activo, es decir **correcti|** > n/2, el proceso ***pi*** realizará un broadcast cada cierto tiempo a su sucesor (línea número 08). La variable ***sucesor i*** incluye el proceso resultante de la función **next_to_i_in_correcti ()** (línea número 07). [17] [16].

La función mencionada anteriormente guarda el identificador del proceso más cercano al identificador del proceso ***pi*** en una secuencia conformada por los elementos del conjunto ***correct i***, en orden consecutivo y ascendente. Ejemplo: si un proceso denotado como p3 tiene el conjunto de procesos correctos correct3 igual a {1, 3, 5, 8}, entonces la función **next_to_i_in_correct3 ()** dará como resultado el número 5. Otro ejemplo: si el proceso denotado como p3 tiene el conjunto correct3 igual a {1, 2, 3}, entonces la función **next_to_i_in_correct3 ()** obtendrá como resultado el valor 1. Si el conjunto |correcti| > n/2, cada uno de los procesos correctos ***pi*** trasmitirá mensajes únicamente a un proceso correcto, formando un conjunto que incluye los procesos correctos. La variable ***i*** contiene el identificador del proceso ***pi*** que supone como su líder. Para ello se toma en cuenta el valor más pequeño del conjunto ***correct i***. [16] [17]. Esto se realiza en la Tarea 1 la misma que se muestra en la Figura 2.1.

En la tarea 2 de la Figura 2.1, se observa el envío y recepción de procesos correctos mediante mensajes, estos procesos se van agregando al conjunto de miembros ***membership i***, el cual guardará los procesos correctos del sistema ***S.*** Para llevar a cabo lo mencionado, en la línea 14 se ejecuta el lazo repetitivo for_each (), con la finalidad de añadir los procesos correctos, de igual forma se utiliza temporizadores los cuales servirán para la respectiva verificación de los procesos correctos mediante la difusión de mensajes. Este proceso se efectúa para que los procesos tengan el conocimiento del proceso que fue seleccionado como líder, la condición para dicha elección es el valor más pequeño del conjunto ***correct i***, el cual será guardado en la variable ***leader i***.

2.6 Diagrama de flujo del algoritmo omega y explicación

A continuación, en la Figura 2.2 se presenta el diagrama de flujo del algoritmo Omega, cabe recalcar que este diagrama de flujo está realizado en base al análisis y explicación de las tareas presentadas en la Figura 2.1.

2.6.1 Explicación del diagrama de flujo del algoritmo omega

A continuación se presenta la explicación detallada de cada una de las tareas que el algoritmo debe proceder a realizar:

1. **DECLARACIÓN DE VARIABLES**

   - **CORRECTO[n].-** Es un vector que almacenará los procesos correctos.
   - **MIEMBROS[n].-** Es un vector que almacenará todo el conjunto de los miembros (procesos) del sistema. Almacena los identificadores de los diferentes procesos descubiertos hasta el momento.
   - **LIDER.-** Es una variable que almacenará el líder escogido del sistema.
   - **NTIEMP.-** Es una variable que almacenará un intervalo de tiempo para la elección del sucesor.
   - **SUCESOR.-** Es una variable que almacenará el sucesor que pertenezca a los procesos correctos.
   - **TIEMPO[n].-** Es el tiempo local del sensor.

- **TIEMPO_SALIDA[n].-** Es un temporizador con respecto a la variable líder.
- **CONST.-** Es una variable que almacena la división de N, siendo ***n*** el número de procesos para 2. Este valor es comparado para elegir el sensor sucesor.

2. TAREAS:

La Tarea 1 inicializa las variables, se procede con la lectura de los identificadores de los diferentes procesos descubiertos y se asignan al vector ***MIEMBROS***. De igual forma estos elementos se asignan al vector ***CORRECTO***. Una vez realizado esta operación compara cada elemento del vector ***CORRECTO*** con el valor de la variable ***CONST*** que contiene el valor resultante de dividir el número de procesos del sistema para 2; ya que esta condición permite que el algoritmo se pueda ejecutar, esta división debe realizarse para que los demás procesos correctos, envíen sus mensajes y reconozcan de entre estos procesos correctos, a uno de ellos como el proceso sucesor (líder) elegido.

Si el elemento del vector correcto es mayor que la variable ***CONST*** entonces a la variable ***SUCESOR*** se le asigna el siguiente elemento del vector ***CORRECTO***.

En caso contrario efectúa la operación de broadcast por medio de un mensaje. Todo este proceso se lo realiza en un lazo infinito cada η tiempo.

Cada elemento del vector **MIEMBROS** a su vez puede contener sub elementos. En la Tarea 2 compara cada elemento de este vector con los sub elementos, si los

procesos son diferentes entonces incorpora el identificador del nodo al vector ***MIEMBROS***, en esta tarea, la función fundamental es el traspaso de mensajes entre nodos.

Posteriormente se redimensiona el tamaño de la variable ***TIEMPO*** y **TIEMPO DE SALIDA,** estos timers son utilizados para la expiración de los procesos que fallen, la variable ***LIDER*** contiene el proceso con el valor más pequeño de todos los elementos del vector ***CORRECTO***, de esta forma queda el elegido el proceso o nodo líder.

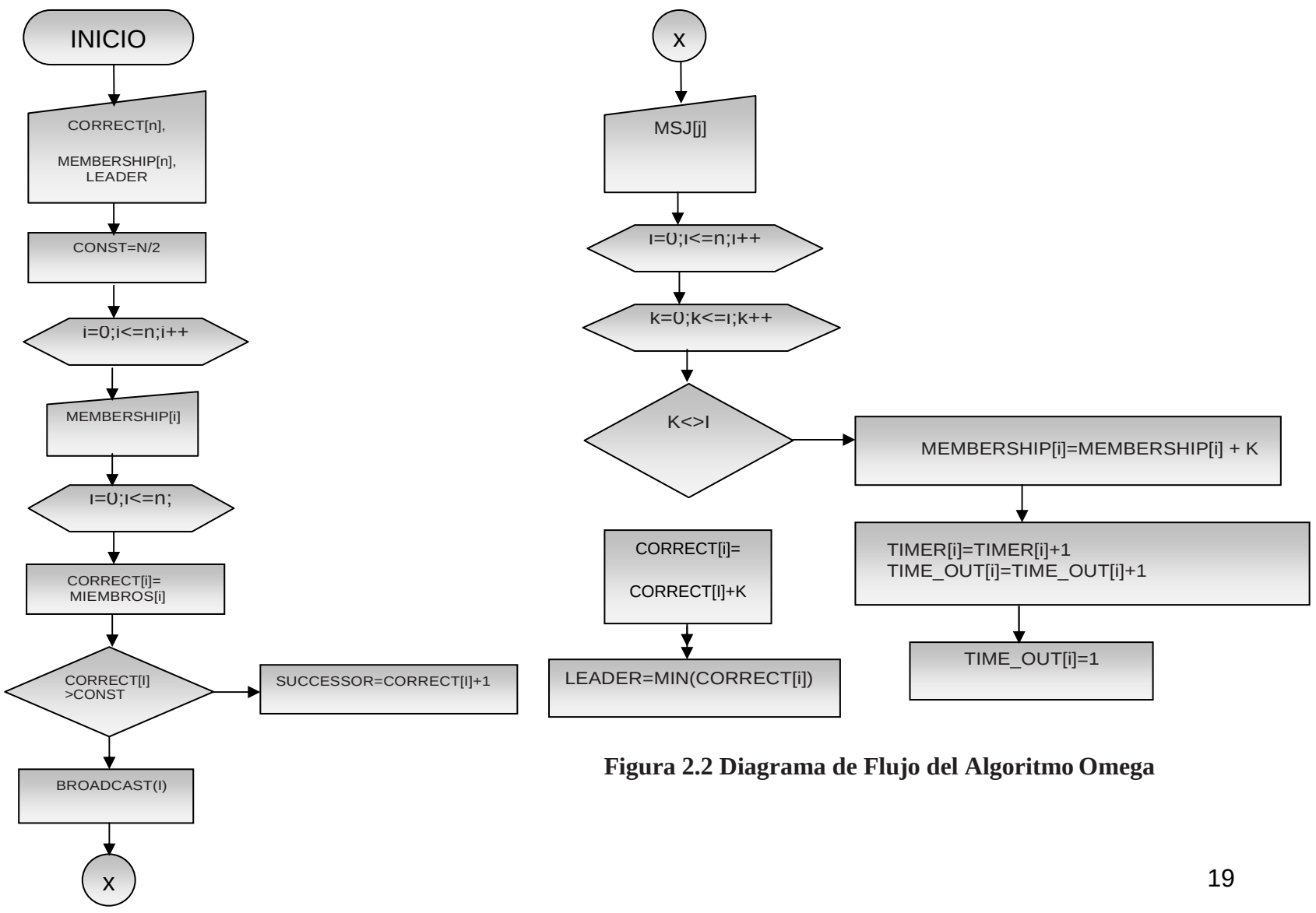


Figura 2.2 Diagrama de Flujo del Algoritmo Omega

2.7 Principios

Para implementar el algoritmo se va a tomar en cuenta un sistema ***S*** el mismo que incluye un conjunto finito de sensores cuya comunicación es realizada mediante el envío de mensajes a través de la red inalámbrica. Es considerado un modelo **crash & recovery (fallo - recuperación),** por cuanto si un nodo empieza a fallar inmediatamente será reemplazado por otro logrando de esta forma que la red se encuentre disponible y garantizando su correcto funcionamiento.

El detector de fallos omega (Ω) permite la elección de un nodo líder de todos los procesos correctos. Este algoritmo tiene una comunicación eficiente debido a que el número de procesos que envían mensajes está limitado por ***n.*** Para implementar un sistema Ω con recuperación a fallos debe existir una mayoría de procesos correctos.

En este algoritmo el número de miembros se desconoce por cuanto cada proceso ***pi*** sólo conoce su propia identidad y el número de procesos en el sistema. Los profesores Tushar Deepak Chandra y Sam Toueg propusieron los detectores de fallo no fiables con la finalidad de resolver el problema de consenso en entornos propensos de fallos asincrónicos.

Un sistema con miembros desconocidos permite a cada proceso ejecutar el algoritmo Omega sin necesidad de saber al principio los identificadores de todos los procesos (sólo sabiendo su propio identificador y el número total de procesos en el sistema). Esto es posible porque los nodos dinámicamente aprenden acerca de la existencia de otros procesos mediante la recepción de mensajes.

El modelo está compuesto de un conjunto finito de procesos. Los identificadores de los nodos están totalmente ordenados, pero no necesitan ser consecutivos. Están denotamos por ***pi, pl, pr,. . . .*** Los procesos sólo conocen acerca de su propio identificador y del número de procesos. En realidad, sólo tienen que conocer la mayoría, es decir, ***n / 2.***

Los procesos se comunican enviando y recibiendo mensajes a esto se le denomina ***broadcast***.

Esta primitiva permite a un proceso enviar simultáneamente el mismo mensaje con el resto de los procesos en el sistema.

Un proceso puede fallar de forma permanente. Si no existe falla es correcto, y si falla (es un proceso defectuoso). Utilizamos ***C*** para denotar el conjunto de procesos correctos, y ***F*** para denotar el conjunto de procesos defectuosos. Consideramos que en un sistema puede haber cualquier número de procesos fallidos, y este número no se conoce a priori.

Cada proceso ***Pi*** inicialmente envía un ***broadcast*** repetidamente para indicar que está activo, esto es realizado para conocer que procesos se encuentran en el sistema, los procesos ***Pi*** están dentro del conjunto ***miembros i*** (al inicio únicamente está formado por sí mismo). El conjunto ***Correct i*** contiene los procesos activos (siempre tiene como elemento por lo menos a sí mismo).

Si al menos la mayor cantidad de procesos están activos ***(|correct i |>n/2)*** cada proceso ***Pi*** envía mensajes cada cierto tiempo a su sucesor en lugar de hacer

broadcast a sí mismo. La variable ***successor i*** contiene los procesos retornados por la función ***next_to_i_in_correct***. Esta función obtiene como resultado el identificador del proceso más cercano en la secuencia formada por los elementos del conjunto ***correct i*** ordenado en forma ascendente. La variable leader ***i*** guarda el identificador del proceso ***Pi*** que piensa que será su líder. Hace referencia al valor más pequeño del conjunto correct ***i.***

La Tarea 1 inicializa las variables, se procede con la lectura de los identificadores de los diferentes procesos descubiertos y se asignan al vector ***MIEMBROS***. De igual forma estos elementos se asignan al vector ***CORRECTOS***. Una vez realizado esta operación compara cada elemento del vector ***CORRECTO*** con el valor de la variable ***CONST*** que contiene el valor resultante de dividir el número de procesos del sistema para 2; ya que esta condición satisface que el algoritmo se pueda ejecutar, esta división debe realizarse para que los demás procesos correctos, envíen sus mensajes y reconozcan de entre estos procesos correctos, a uno de ellos como el proceso sucesor (líder) elegido. Si el elemento del vector correcto es mayor que la variable ***CONST*** entonces a la variable ***SUCESOR*** se le asigna el siguiente elemento del vector ***CORRECTO***.

En caso contrario efectúa la operación de broadcast por medio de un mensaje. Todo este proceso se lo realiza en un lazo infinito cada **ŋ** tiempo.

Cada elemento del vector ***MIEMBROS*** a su vez puede contener sub elementos. En la Tarea 2 se envía un mensaje a cada proceso, por cada elemento del vector con los sub elementos y compara con los elementos del vector ***MIEMBROS***, si los

elementos son diferentes entonces incorpora el identificador del nodo al vector ***MIEMBROS***, en esta tarea, la función fundamental es el traspaso de mensajes.

Posteriormente se redimensiona el tamaño de la variable ***TIEMPO*** y ***TIEMPO DE SALIDA,*** estos timers son utilizados para la expiración de los procesos que fallen, que a su vez incorpora el identificador del nodo aumentado en el vector ***MIEMBROS***; además a la variable ***LIDER*** se le asigna el proceso con el valor más pequeño de todos los elementos del vector ***CORRECTOS***, para que de esta forma quede elegido el proceso líder.

2.8 Redes WSN

Las redes de sensores WSN se componen de dispositivos de bajo costo y consumo estas extraen información del entorno que les rodea, realizan un procesamiento en forma local, y posteriormente comunican mediante enlaces inalámbricos hasta llegar a un nodo principal de coordinación. Poseen una característica de auto-restauración que permite buscar nuevas rutas para el encaminamiento de los paquetes de datos en el caso que un nodo falle en la red. [13]

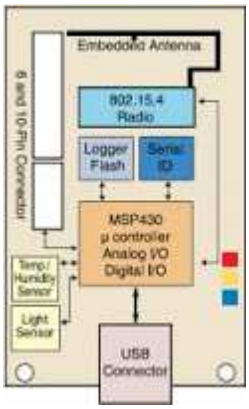
Mediante este mecanismo, la red continuará funcionando, aunque haya nodos individuales que pierdan potencia o se dañen. [13]

2.9 Funcionamiento de las redes de sensores inalámbricas WSN

Se componen de dispositivos de bajo costo y consumo capaces de adquirir datos de su entorno, procesarlos en forma local, y enviarlos mediante enlaces inalámbricos hacia un nodo central de coordinación. Los nodos son elementos que forman parte de la infraestructura de comunicaciones y que permiten el envío de los mensajes por nodos más distantes hacia el nodo central de coordinación. La red de sensores inalámbricos está compuesta por una gran cantidad de dispositivos que se encuentran distribuidos espacialmente y que permiten verificar distintas condiciones en diferentes puntos, entre las cuales se puede citar a las siguientes: el sonido, la temperatura, la presión, la vibración, entre otros. Los sensores se pueden clasificar en fijos o móviles. [13]

Figura 2.3 Componentes de un nodo de sensor

Las redes WSN se componen de los siguientes elementos:



- **Sensores:** Existen diferentes tecnologías y tipos, estos receptan la información del medio físico y posteriormente es transformada en señales eléctricas.
- **Nodos De Sensor:** Interceptan la información del sensor mediante sus puertas de datos y la trasmiten a la estación base.
- **Gateway:** Es un componente que permite conectar la red WSN y una red TCP/IP.
- **Estación Base:** Es un elemento cuya finalidad es la recolección de datos.
- **Red Inalámbrica:** Se basa en el estándar 802.15.4 ZigBee. [14]

Las redes WSN están formadas por diferentes nodos, los cuales se pueden clasificar según las funciones que realicen en la red. Los nodos están ordenados de tal forma que la comunicación se pueda localizar de forma correcta entre todos los componentes de la red.

En la siguiente figura, se visualiza un esquema de la distribución de los nodos en una red.

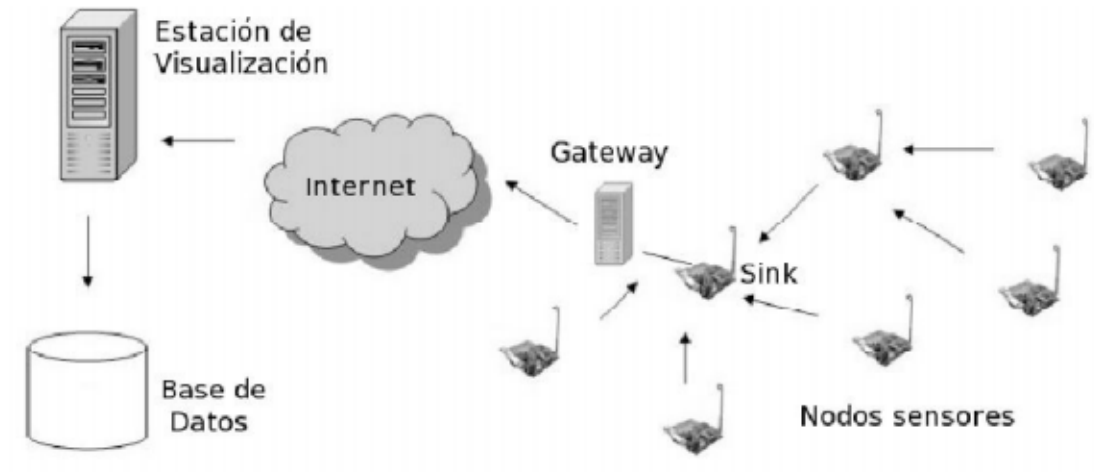


Figura 2.4 Esquema de componentes de una WSN

Nodo Sensor.- Para la definición de un nodo sensor, nos remitiremos a una cita textual presentada por John A. Stankovic, que define de forma completa y concisa las características de un nodo:

"Cada nodo consiste en: capacidad de procesamiento (uno o varios micro controladores o microprocesadores), puede contener varios tipos de memoria (de programa, de datos y memorias flash), tiene un transceptor de RF (normalmente con una sola antena omnidireccional), tiene una fuente de energía (por ejemplo, baterías o células solares), y dan cabida a diversos sensores y actuadores."

Los nodos sensores realizan las siguientes tareas: primero registran el estímulo externo (bien sea de temperatura, humedad, etc.). Después de esto hace un pre-procesamiento de la información para prepararla para ser enviada. Al tener la información lista, se establece la conexión por 802.15.4 con el coordinador al cual se

debe enviar, y finalmente según la programación que se tenga, guarda registros de todas las mediciones realizadas en su memoria. En la figura 2.5 se muestra una fotografía de un nodo MicaZ.

Figura 2.5 Fotografía de un nodo MicaZ.

Estación base

La estación base está conectada a algún dispositivo que permita intercambiar información entre la WSN y otras redes. Su función principal es recoger la información que llega de cada uno de los sensores, por lo que comúnmente tiene algunas características que lo distinguen como una antena más grande para lograr mejor alcance y está conectada a una salida a otras redes, a las cuales envía la información.

Gateway

El Gateway es el elemento que se encarga traducir los datos recogidos por la red para que sean compatibles con los protocolos de las redes hacia los cuales serán enviados.

En varias ocasiones el Gateway es la estación base conectada a un servidor con acceso a internet.

2.10 Características de las WSN:

Las WSN tienen como característica principal la auto restauración, es decir, si se avería un nodo, la red buscará vías alternas para encaminar los paquetes de datos. Así de esta forma, la red continuará funcionando aunque existan nodos que pierdan potencia o se dañen. Las características de autoconfiguración, reparación entre otras se han desarrollado para estas redes con la finalidad de solucionar problemas que anteriormente no era posible con otras tecnologías. Entre las principales funcionalidades destacan las siguientes [13]

- Existe una buena integración con otro tipo de tecnologías:

  Por ejemplo: biología, medicina, agricultura, minería. [14]

- Permite interactuar al hombre con el medio. [14]

  Por ejemplo: Redes vehiculares, entre otros. [14]

- Realiza una menor utilización de recursos. [14]

2.11 Modelo de un diseño de una red WSN:

Las redes WSN brindan diferentes ventajas para aquellos usuarios que necesitan elaborar sistemas cableados e inalámbricos y hacen uso de la mejor tecnología para este tipo de aplicaciones.

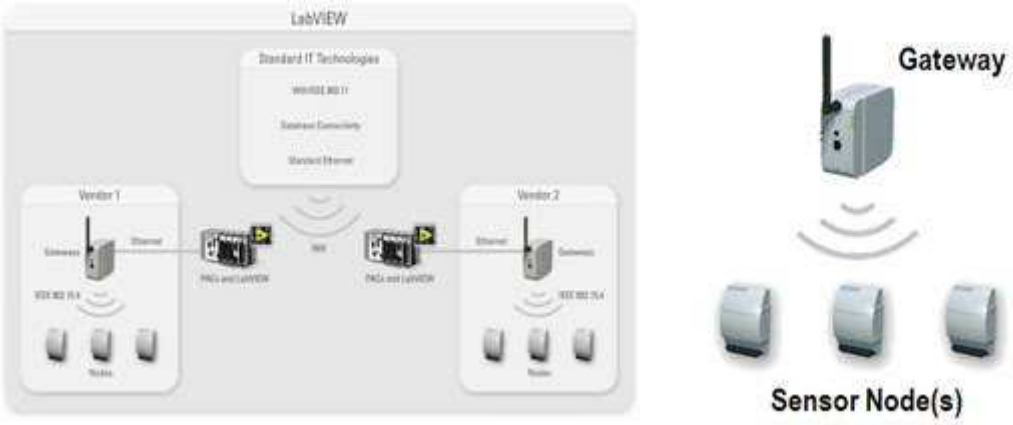


Figura 2.6 Modelo de una WSN

- En la Figura 2.6 se muestra el modelo de una red inalámbrica de sensores. Esta red está compuesta por nodos cuya principal finalidad es adquirir determinados datos del entorno para ser enviados hacia un líder común. El líder a su vez será el responsable de dirigir los datos almacenados hasta un gateway el cual permitirá el acceso desde el exterior.

2.12 Ventajas de los sensores inalámbricos:

Entre las principales ventajas de este tipo de redes se puede detallar las siguientes [14]:

1. Tiempo de vida de la red, menor energía mayor tiempo de vida para la red.
2. Bajo costo y fácil instalación, una red cableada puede llegar a alcanzar un alto costo no solo en lo económico y en tiempo.
3. Cualquier dispositivo puede estar situado en cualquier punto dentro del área de cobertura sin la necesidad de cables.
4. Bajo consumo de energía, los nodos son alimentados por medio de una batería
5. Precisión al momento de realizar las mediciones

2.13 Parámetros de una WSN:

Los parámetros más importantes de una red WSN se detallan a continuación [13]:

- Costo es más bajo en relación de una red cableada
- La instalación es más sencilla con relación a una red cableada

Entre las principales características de un nodo sensor podemos citar las siguientes [13]:

- Conexión de diferentes dispositivos
- Tolerancia antes fallos
- Permiten la comunicación

2.14 Sistema de adquisición de datos:

Existen nodos sensores de diferente tecnología y naturaleza. La información es extraída del medio y convertida posteriormente en señales eléctricas. En el mercado existen nodos sensores que incluyen diversos parámetros dependiendo del campo al cual va a ser aplicado como: sensores de luz, humedad en el suelo, humedad en el aire, medición de temperatura entre otros.

2.15 802.15.4 ZIGBEE. Norma 802.15.4

ZigBee es una asociación de 25 empresas la mayor parte de ellas fabricantes de semiconductores sin fines de lucro que se unieron con la finalidad de patrocinar el desarrollo de una tecnología para redes inalámbricas de bajo costo. [14]

Características [14]:

Se utiliza principalmente en lo relacionado con el monitoreo y control.

La memoria está comprendida de 4KB a 32KB.

Posee una velocidad de transmisión de 250 Kbps y una cobertura de 10 a 75 metros.

- A pesar que utiliza la misma frecuencia que redes como WiFi o Bluetooth su desempeño no se ha visto afectado, esto se debe a su baja tasa de transmisión y a sus características propias del estándar IEEE 802.15.4.

- Cada red ZigBee tiene un identificador único de red lo cual permite que coexistan varias redes en un mismo canal de comunicación sin generar ningún tipo de problema.

En teoría pueden existir hasta 16.000 redes diferentes en un mismo canal y cada una puede estar constituida hasta por 65.000 nodos, evidentemente estos límites se ven truncados por algunas restricciones físicas (memoria disponible, ancho de banda, entre otros.).

- Gracias a su topología de malla (MESH) permite que la red se recupere de problemas en su comunicación aumentando de esta manera su confiabilidad.

2.16 Aplicaciones actuales y posibles usos de los sensores inalámbricos WSN:

USOS:

Estas se pueden aplicar en diferentes entornos ya sea un entorno adverso o en oficinas, empresas, fábricas, hogar durante un período de tiempo con la finalidad de detectar cambios y recolectar información suficiente para poder generar algún cambio o intervención [14].

Por ejemplo:

- En la agricultura, tecnología militar entre otros.

Figura 2.7 Aplicaciones de una WSN

MOTAS:

- Los nodos pueden contener: sensores de temperatura, humedad, biológicos, químicos entre otros. Algo que diferencia una mota de otra es que permita conectar cualquier sensor de forma sencilla .[22]

- Los sensores son cada vez más potentes, más pequeños y consumen menor cantidad de energía.

- El software que generalmente se utiliza es el lenguaje NESC (parecido al lenguaje C) sobre el sistema operativo TinyOS. Sin embargo existen también nuevas plataformas que están surgiendo.

2.17 Componentes de un nodo WSN:

Un nodo WSN incluye varios componentes como la batería, el radio, micro controlador, circuito analógico y una interfaz del sensor. En sistemas que son alimentados mediante batería con tasas de datos altas y uso de radio consumen una mayor cantidad de energía. Generalmente la vida útil de una batería es de tres años de vida, por este motivo muchos de de los sistemas WSN utilizan como base ZigBee por cuanto ayuda a obtener un bajo consumo de potencia. [22]

Para alargar la vida de la batería, cada cierto tiempo un nodo WSN se enciende y transfiere los datos alimentándose del radio y apagándose para conservar la energía.

La tecnología de radio WSN debe trasmitir una señal en forma eficiente y permitir al sistema regresar al modo sleep con un mínimo uso de energía. Es decir el procesador debe despertar, encenderse y volver a sleep. [22]

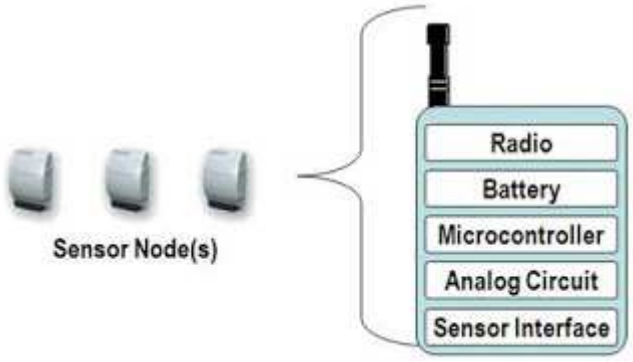


Figura 2.8 Componentes de un nodo sensor WSN

Gateway

Es un dispositivo que interconecta redes usando protocolos y arquitecturas completamente distintas. [22]

Los Gateway se usan como dispositivos universales en una red empresarial la misma que está compuesta por redes de diversos tipos. [22]

Los Gateway garantizan que los datos de una red que transportan son compatibles con los de la otra red. Conectan redes de distintas arquitecturas resolviendo sus protocolos y permitiendo que los elementos de una red puedan comunicarse con otros dispositivos de otra red diferente. [22]

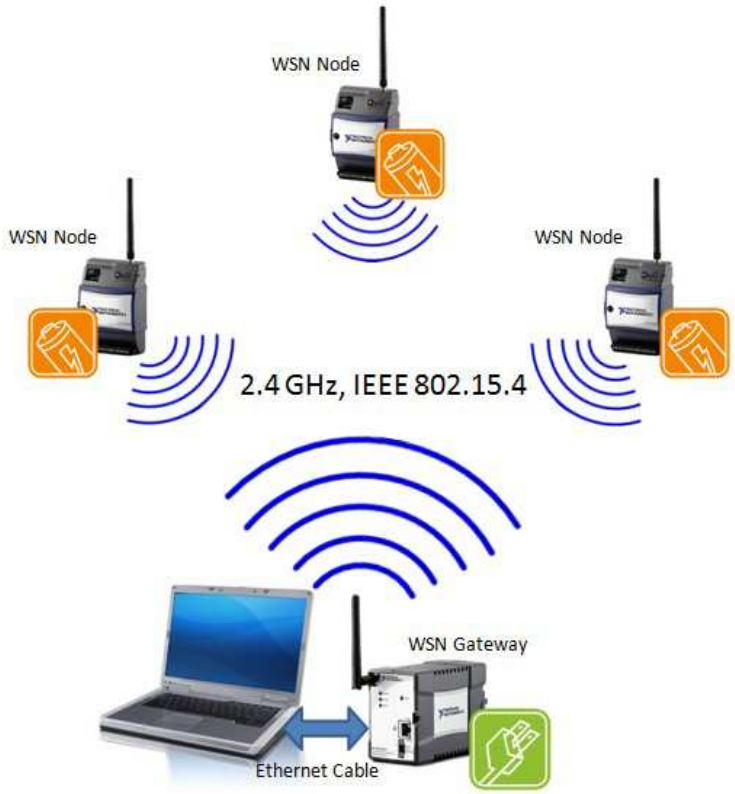


Figura 2.9 Gateway de una WSN

2.18 Porque es importante el ahorro de energía

Los nodos en una red de Sensores están limitados por la capacidad de energía en la batería. Por ende, la vida útil de la misma se ve afectada. De esta manera, la administración de la energía, es un factor de suma importancia. En estas redes el flujo de información, pasa de un nodo a otro de manera subsiguiente. [15]

La optimización en el consumo de energía tiene como finalidad prolongar la vida de cada nodo de la misma forma que inferirá por completo en toda la red.

Además de incorporar algoritmos eficientes en el consumo de energía a nivel de un nodo, debemos considerar a lo ancho de la red la cooperación entre los nodos para incrementar la vida de la red. [15]

2.19 Ahorro de energía en WSN

Dadas las características específicas de las WSN, se hace indispensable la aplicación de técnicas que permitan el máximo ahorro de energía en los nodos de la red, sin comprometer las necesidades de alcance.

La finalidad efectuar un uso eficiente de la energía es extender el tiempo de vida de la red y cumplir con los requerimientos de la Calidad de Servicios. Los avances tecnológicos que permiten incrementar la capacidad de las baterías se desarrollan en forma paulatina. Es decir que en lo relacionado con la eficiencia energética seguirá siendo un desafío para este tipo de redes en el futuro. [13]

Realizar un diseño de los nodos que permita obtener un bajo consumo representa seleccionar elementos de baja potencia. Las consideraciones a tomar en cuenta es el consumo de energía del CPU, el sensor, el radio transceptor, la memoria externa entre otros durante el modo normal de operación. [13]

Optimizar el consumo de energía en los nodos para alcanzar el máximo tiempo de vida de la red es uno de los objetivos más importantes a ser tomados en cuenta. Los componentes a considerar son los siguientes:

La transmisión es el primer aspecto que consume energía. En un sistema distribuido varios sensores necesitan comunicarse mediante largas distancias, lo que se traduce en un mayor consumo de energía. Por tal motivo, es una buena recomendación procesar en forma local la mayor cantidad de energía, para reducir el número de bits transferidos.

El CPU queda en un estado “sleep” mientras “no tenga tareas asignadas” [13]. La transmisión de información desde los nodos puede ser de tres formas: modo continuo en los intervalos señalados, encaminado por eventos (se envía cuando se cumple determinada condición) o por consulta (solo cuando existe una solicitud). Hay también sistemas híbridos que combinan los métodos mencionados [13].

- Ahorrar la distancia de las comunicaciones.
- Programación eficiente de líneas de código.
- Protocolos de enrutamiento

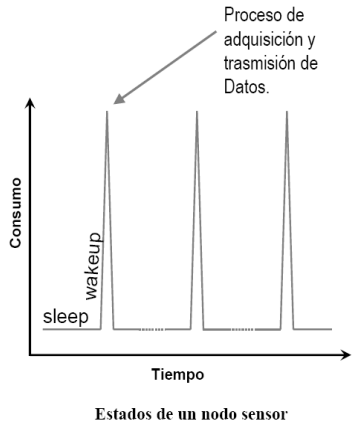


Figura 2.10 Estados de un nodo sensor

A continuación se detallan algunas estrategias de hardware de ahorro de energía [13].

Para ahorrar energía los nodos pasan por los siguientes estados:

- **Sleep:** El nodo pasa la mayor cantidad de tiempo en este estado es decir sin ninguna actividad que realizar.
- **Wakeup:** Es necesario reducir este tiempo para regresar rápidamente al estado de trabajo.
- **Active:** Significa que debe estar el menor período de tiempo trabajando y regresar de forma inmediata al estado **sleep**

La capacidad, tiempo de vida, y desempeño de los sensores están dirigidos por la energía. Los sensores deben ser capaces de estar activos durante un tiempo largo razonable sin tener que recargar la batería ya que el mantenimiento es costoso.

La importancia del ahorro de energía radica en prolongar el tiempo de vida de la red y prevenir la degradación en la conectividad, usando técnicas agresivas de administración de la energía.

2.20 Fuente de alimentación

Debido a que el punto más desafiante de las redes de sensores es la limitada provisión de energía, muchos esfuerzos de investigación tienen como objetivo el mejorar el rendimiento energético en todos los aspectos de la red. En WSN, la energía es consumida principalmente para tres propósitos: transmisión de datos, procesamiento de la señal, y la operación de hardware. Es deseable desarrollar técnicas de proceso de ahorro de energía que reducen al mínimo los requerimientos de energía a través de todos los niveles de la pila de protocolo (protocolo stack) y, al mismo tiempo, reducir al mínimo los procesos usados para el control y la coordinación de la red.

2.21 IPV6

Introducción

Internet es una red de redes que permite conectar diversos dispositivos que se encuentran repartidos en diferentes partes del mundo. Se ha transformado en un instrumento importante para las labores diarias tanto para el trabajo como para el entretenimiento, renovando la calidad de vida del ser humano, permitiendo acceder de forma fácil y rápida a una gran cantidad de aplicaciones e información en cualquier punto geográfico.

El protocolo IP forma parte del conjunto de protocolos de comunicaciones TCP/IP usado en Internet. Una de las funciones más importantes de este protocolo incluye el encapsulamiento de paquetes en un datagrama y el enrutamiento de los mismos mediante la red.

Hasta el momento la utilización de IPv4 ha sido favorable sin embargo el mundo de las telecomunicaciones ha ido evolucionando cada vez más y con ello ha surgido nuevas necesidades que con la actual versión no eran soportadas como la interconexión de una mayor cantidad de dispositivos y la confluencia de distinto tipo de tráfico en la misma red. El uso del protocolo IPv4 después de poco tiempo podría restringir las redes distribuidas, servicios móviles, entre otros. [26]

2.22 Direccionamiento IPV6

IPV6 surgió por el acelerado crecimiento del internet y por la falta de direcciones que cubran estas necesidades. Una solución planteada por los proveedores de servicios Internet para resolver dichos inconvenientes, es facilitar a los clientes direcciones NAT. Esto significa que se utilizará una única dirección IP pública para una red privada. Sin embargo en los dispositivos móviles este método no funciona. Para proporcionar una solución definitiva con el direccionamiento IPv6 el número de direcciones se extiende a 128 bits.

En una dirección IPv6 los 128 bits identifican interfaces únicas o un grupo de las mismas. Una interfaz posee varias direcciones IPv6: **unicast** (una sola interfaz), **anycast** (una interfaz selecciona una entre varias posibilidades), **multicast** (múltiples interfaces en forma simultánea). No existen las direcciones **broadcast** que en IPv4 ha causado problemas en rendimiento de la red. En el direccionamiento IPv6 los bits iniciales indican el tipo de dirección al que pertenece, a este campo se denomina prefijo y determina la ruta por la cual va a ir el paquete. Una dirección IPv6 contiene un prefijo y a continuación el identificador del nodo. [26]

2.23 Diferencias del protocolo TCP/IPv4 y TCP/IPv6

El protocolo TCP/IP (Transfer Control Protocol/Internet Protocol) es usado con el objetivo de administrar el tráfico de la red. Este protocolo se compone de dos protocolos: TCP el cual se encarga de controlar la transmisión de la información y el protocolo IP que se encarga de identificar la máquina en toda la red. [25]

TCP/IPv4

A partir de 1.981 se usa la versión 4 de este protocolo. Posee 32bits y se compone de cuatro grupos de 8 bits cada uno (8 x 4 = 32). Utilizan el siguiente formato binario 11000000.10101000.00000000.00000001, o 192.168.0.1 en traducido a formato decimal.

Esta combinación puede producir 4.000 millones de combinaciones en términos generales. Esto puede parecer una cantidad suficiente pero sin embargo no lo es. Actualmente está ocupado en forma aproximada unas 2 de 3 partes de estas combinaciones. [25]

TCP/IPv6

El protocolo TCP/IP versión 6 tiene 128 bits, lo cual genera aproximadamente 34 trillones de direcciones. La configuración de este direccionamiento es mejor organizado que el actual por cuanto se compone de ocho grupos de 16 bits (que van de 0 a ffff), separados por el signo dos puntos (:). El valor 0 puede ser reemplazado por dos puntos seguidos (::). Una dirección IP en el protocolo TCP/IP versión 6 sería la que se detalla a continuación: [25]

2005:205:0:1:175:0:bafd:14
o lo que sería lo mismo

2005:205::1:175::bafd:14 (se puede ver que se ha reemplazado los grupos que tienen valor 0 por los dos puntos seguidos ::).

2.24 IP Pública

Cualquier dispositivo que se conecte en forma directa al internet corresponde a este tipo de dirección. Un ejemplo puede ser los servidores que proporcionan alojamiento a sitios web como Google, los routers que facilitan el acceso a internet entre otros. Las direcciones IP públicas son únicas no pueden repetirse. [24]

2.25 IP Privada

Permite identificar dispositivos que se encuentran dentro de una red privada. Los rangos se detallan para IPv4 e IPv6 se detallan a continuación: [24]

Para IPv4

De 10.0.0.0 a 10.255.255.255

172.16.0.0 a 172.31.255.255

192.168.0.0 a 192.168.255.255

169.254.0.0 a 169.254.255.255

Para IPv6

En IPv6 un término equivalente a las direcciones IP privadas es denominado (ULA *Unique Local Address*) "Dirección Local Única".

Las ULAs son aquellas direcciones IP que inician con FD. Como por ejemplo: FD03:2880:2110:CF01:0ACE:0000:0000:0009.

Las direcciones IP privadas pueden ser las mismas pero en redes diferentes. Si se desea conectar una red privada con otra red distinta o con el Internet es necesario un traductor. NAT *Network Address Translation* (Traducción de Dirección de Red) es una funcionalidad del router que se utiliza de puente para poder acceder desde un equipo que tiene una dirección IP privada hacia el servidor de internet que tiene una dirección IP pública. [24]

CAPITULO III: ANALISIS Y DISEÑO

3.1 Requerimientos

Dentro de las aplicaciones distribuidas las tareas enfocadas con el diseño y verificación de algoritmos son bastante complicadas, para poder realizar un estudio se ha procedido a identificar diferentes tipos de problemas.

Uno de los problemas que más significado tiene es el consenso en el cual varios procesos intentan llegar a un acuerdo común. Este tipo de inconveniente no es resuelto en forma fácil en sistemas donde en cualquier momento los procesos pueden fallar. Para solucionar esto los profesores Tushar Deepak Chandra y Sam Toueg plantearon los detectores no fiables de fallos.

En esta tesis se realizará un estudio del detector no fiable de fallos Omega aplicando el modelo de fallo y recuperación (Crash-Recovery). Nos enfocaremos en la simulación de dicho algoritmo en donde los procesos pueden ser propensos a fallar y luego volver a recuperarse, en donde se ha llegado a demostrar que el consenso puede resolverse.

Previo un análisis se determinó los componentes principales que debe incluir el simulador, los mismos que se detallan a continuación:

1. El simular deberá permitir 2 tipos de ejecuciones:

- Ejecución paso a paso la cual permita visualizar en un panel los valores que van tomando las variables y de esta forma simular el intercambio de mensajes para un mejor entendimiento del algoritmo.

- Ejecución automática la cual permita visualizar los resultados finales del algoritmo al usuario.

2. Debe ser parametrizable permitiendo trabajar bajo Ipv4 o Ipv6,

3. Tener una interfaz gráfica amigable, interactiva y fácil de usar

4. Permitir interactuar con una base de datos para el guardado de la información.

5. Un requerimiento importante a tomar en cuenta es que el sistema deberá permitir simular la recuperación de la red cuando un nodo falle determinando de todos los nodos un sustituto.

Para el desarrollo del simulador es importante tomar en cuenta las siguientes consideraciones:

- El sistema se aplica a una red WSN.

- El algoritmo Omega utiliza procesos, para el sistema se hace referencia a nodos.

- Un proceso fallido será el equivalente a un nodo fallido.

- El término recuperación del sistema hace referencia a la recuperación de la red.

3.2 Análisis de los requerimientos

Después de realizar un análisis previo del Algoritmo Omega, se llegó a determinar algunas consideraciones importantes que deben tomarse en cuenta con la finalidad que éste pueda ser adaptado a redes de sensores inalámbricas posterior a su implementación. Las consideraciones se detallan a continuación, para que así su funcionamiento sea el adecuado.

- El algoritmo Omega como se detalló anteriormente está orientado a procesos, mediante la ejecución del mismo permite elegir un proceso líder, esta es la primera modificación que se debe adaptar a dicho algoritmo. Los procesos van a ser reemplazados por nodos, para la elección del nodo líder el algoritmo seleccionará uno en base a algún criterio.

- Al referirnos al término recuperación del sistema se hace referencia a la recuperación de la red.

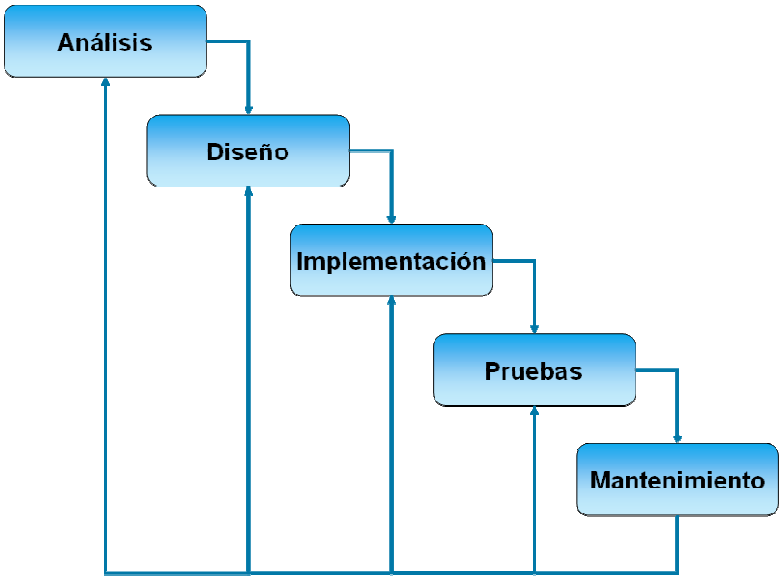
- El simulador debe permitir identificar los nodos de la red que se encuentran activos y de igual forma obtener los datos de su identificador para posteriormente realizar la selección del nodo líder.
- El proyecto a desarrollar contempla el diseño e implementación del algoritmo detector de fallos Omega a través de un modelo de simulación.
- La ejecución del simulador prototipo Omega será realizada en una PC.

- Para la realización del simulador se utilizará el método del ciclo de vida clásico de software (metodología en Cascada). Este ciclo comprende: análisis, diseño, implementación y pruebas del sistema.

A continuación se detallan algunas consideraciones importantes sobre esta metodología:

A esta metodología se la conoce como modelo clásico, lineal o secuencial. Se la utiliza para tener un control sobre el ciclo de vida de los sistemas, se compone de una serie de actividades entre las cuales destacan las siguientes: análisis de requerimientos, diseño, implementación, integración y pruebas. [23]

- El ***análisis de requerimientos*** permite comprender las necesidades del usuario final.
- El ***diseño*** incluye diagramas que proporcionan un detalle de cómo va a estar estructurado el producto.
- La ***implementación*** hace referencia a la programación. El desarrollo del aplicativo es realizado en cualquier lenguaje de programación, esta etapa incluye el código fuente.
- La ***integración*** este proceso realiza una integración de cada una de las partes para completar el producto final.



Este método ordena cada fase del ciclo de vida de software, para comenzar una etapa es necesario que la etapa anterior finalice. Si existe un atraso en una de las etapas permanece en la etapa actual hasta que esta haya concluido. Debido a que el proceso está planificado es más fácil la determinación de costos y plazos. Este modelo es comparado con una cascada de agua por cuanto incluye varios saltos y cada uno constituye cada etapa del ciclo de vida. [23]

La metodología en cascada se compone principalmente de:

1. El inicio y el alcance del proyecto
2. La estructura del proyecto (costo, calendario, recursos)
3. Descripción de las necesidades del negocio y un análisis que detalle la solución

4. La creación de la solución
5. Elaboración de pruebas que garanticen que la solución funciona, implementación de de la solución
6. Cierre del proyecto.

Ventajas

- El horario es establecido con los plazos más adecuados para cada fase de desarrollo.
- Este proceso permite realizar una entrega del proyecto a tiempo.
- El proceso es sencillo y facilita la gestión de proyectos.
- Permite obtener un control eficiente del proyecto.

En la parte inferior se detallan los principales parámetros que incluirá el simulador.

PARAMETROS DEL SIMULADOR

- El algoritmo se adaptará a un ambiente de nodos en vez de procesos.
- El simulador mostrará el intercambio de mensajes entre nodos.
- El simulador mostrará como el algoritmo se ejecuta en cada nodo.
- El simulador mostrará una ejecución paso a paso para comprender el funcionamiento del mismo.

- El simulador permitirá trabajar con direccionamiento IPV4 e IPV6.
- El simulador podrá contar con la opción de poder agregar un nuevo sensor a la red existente.
- El simulador permitirá la selección de un nodo líder aplicado a las redes de sensores inalámbricas.
- El simulador permitirá la activación o desactivación de nodos de la red mediante un archivo de parámetros de configuración.
- El simulador permitirá configurar el tipo de ejecución de los resultados: manual (paso a paso) o automática.
- El simulador visualizará en pantalla un panel de resultados con los resultados finales que incluyen los valores de los vectores del algoritmo.
- La implementación del Algoritmo Omega es un modelo crash & recovery (fallo - recuperación), esto significa que si un nodo falla será reemplazado por otro permitiendo de esta forma la recuperación del sistema, esto podrá ser comprobado dentro del simulador.

3.3 Diseño del Algoritmo

En esta etapa se ha tomado en cuenta ciertas restricciones que hacen referencia a las redes de sensores inalámbricas para de esta forma realizar un análisis y un diseño que se adapte de mejor manera a las necesidades.

Las restricciones del algoritmo se detallan a continuación:

- El simulador permitirá realizar una simulación en pantalla del algoritmo detector de fallos Omega con la finalidad de comprobar una de las características principales que posee, como es la tolerancia a fallos y observar que gracias al mismo la red continúa funcionando.

El Algoritmo Omega consiste en un sistema compuesto por un conjunto finito de nodos. Los identificadores de los nodos están totalmente ordenados. Cada nodo esta denotado por ***pi, pl, pr,*** de igual forma cada uno conoce acerca de su propio identificador y el número total de procesos en el sistema.

El vector ***Correct*** contiene los nodos que están activos.

El vector ***Membership*** contiene los nodos que conforman el sistema.

La variable ***Leader*** guarda el identificador del nodo que es considerado como líder. Este valor es el más pequeño en el vector ***Correct.***

El algoritmo incluye 2 tareas.

La tarea 1 obtiene la media de todos los nodos del vector **Membership,** luego realiza un broadcast con cada elemento, si la posición del nodo es mayor que la media obtenida entonces invoca a la función **next_to_i_in_correct** la cual obtiene como resultado el identificador correspondiente al siguiente nodo del conjunto de elementos del vector **Correct** en orden ascendente. El vector **Correct** está formado por el conjunto de los nodos activos resultantes del broadcast incluyendo el nodo siguiente retornado por la función especificada anteriormente.

Posición	1	2	3	4	5	6

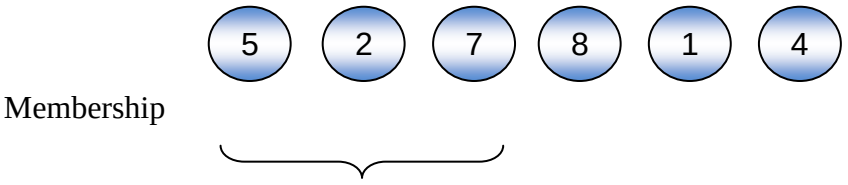


Figura 3.2 Funcionamiento del algoritmo Tarea 1

La tarea 2 verifica la creación de nodos nuevos y compara que el nodo creado no exista en los miembros del vector ***Membership***.

Crea la variable ***timer*** la cual es un tiempo local para cada elemento nuevo del vector ***Membership,*** este tiempo sirve para determinar si el nodo aspirante a ser líder falló.

De igual forma crea la variable ***timeout*** la cual es un tiempo de acuerdo para la elección del nodo líder.

Agrega al vector ***Correct*** el nuevo nodo.

Selecciona el identificador del nodo con el valor más pequeño del vector ***Correct.***

K = {18}

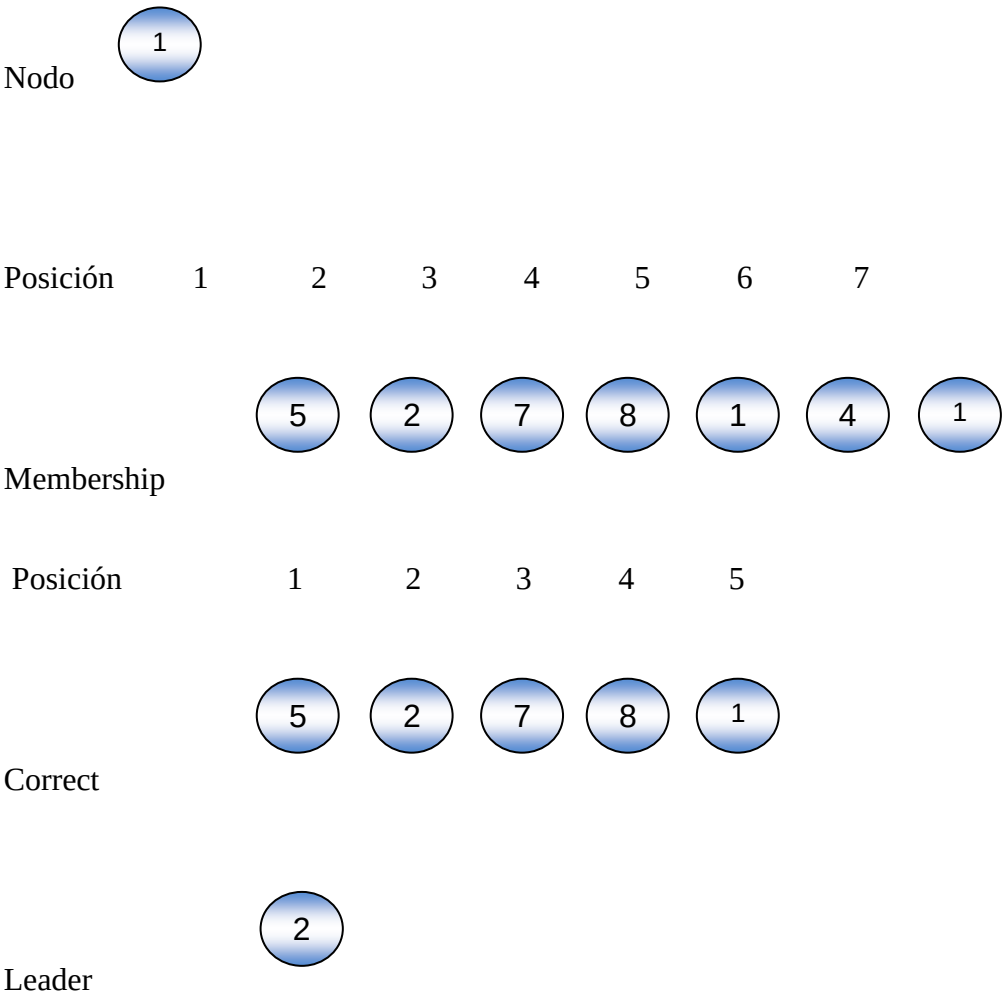


Figura 3.3 Funcionamiento del algoritmo Tarea 2

La variable ***Leader*** contiene el valor más pequeño en el vector **Correct.**

Este proceso se va a realizar mientras no expire el timer. Una vez expirado el tiempo visualiza los resultados.

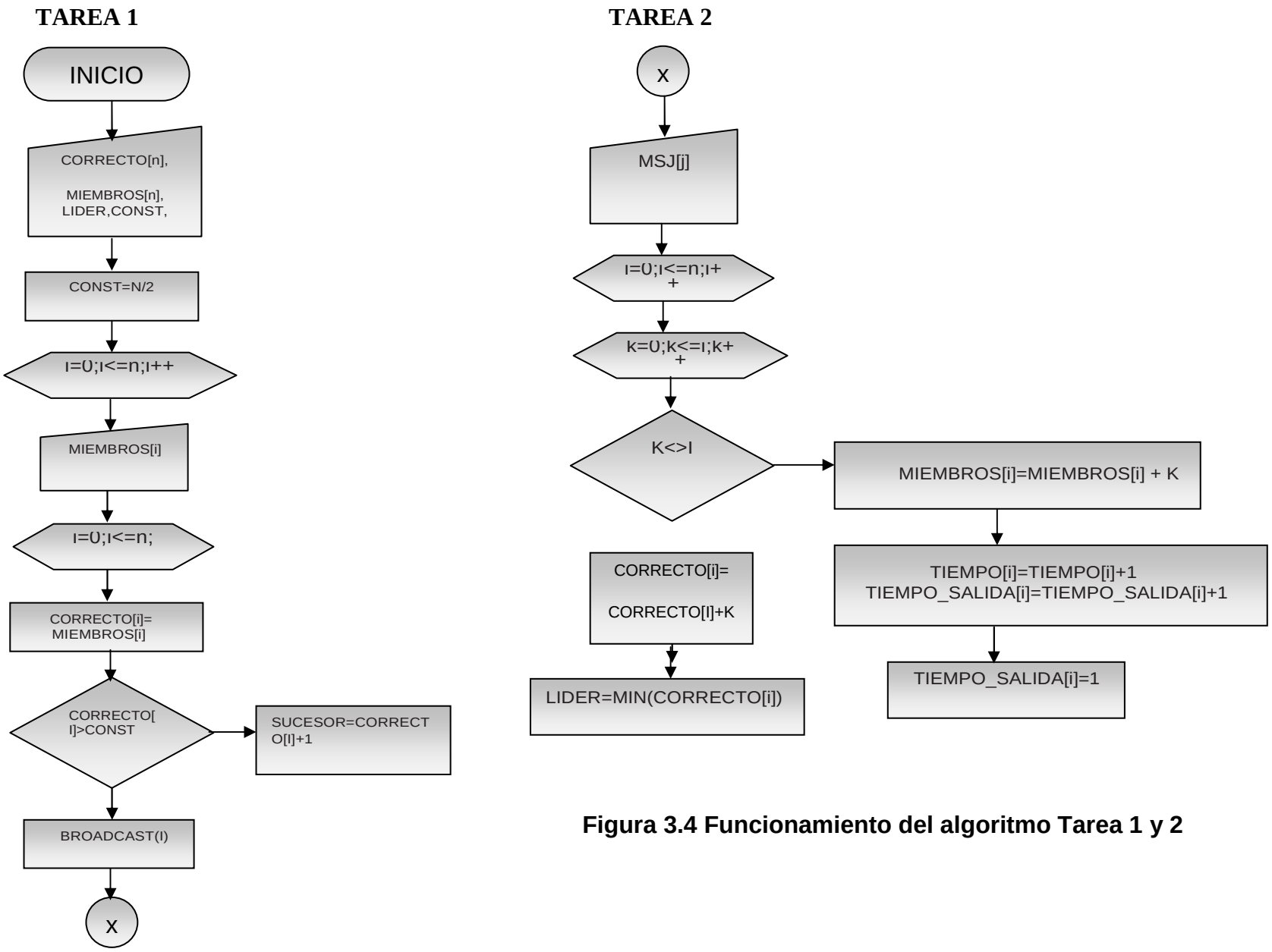


Figura 3.4 Funcionamiento del algoritmo Tarea 1 y 2

3.3.1 Implementación del Algoritmo

El simulador incluye dos métodos Tarea 1 y Tarea 2.

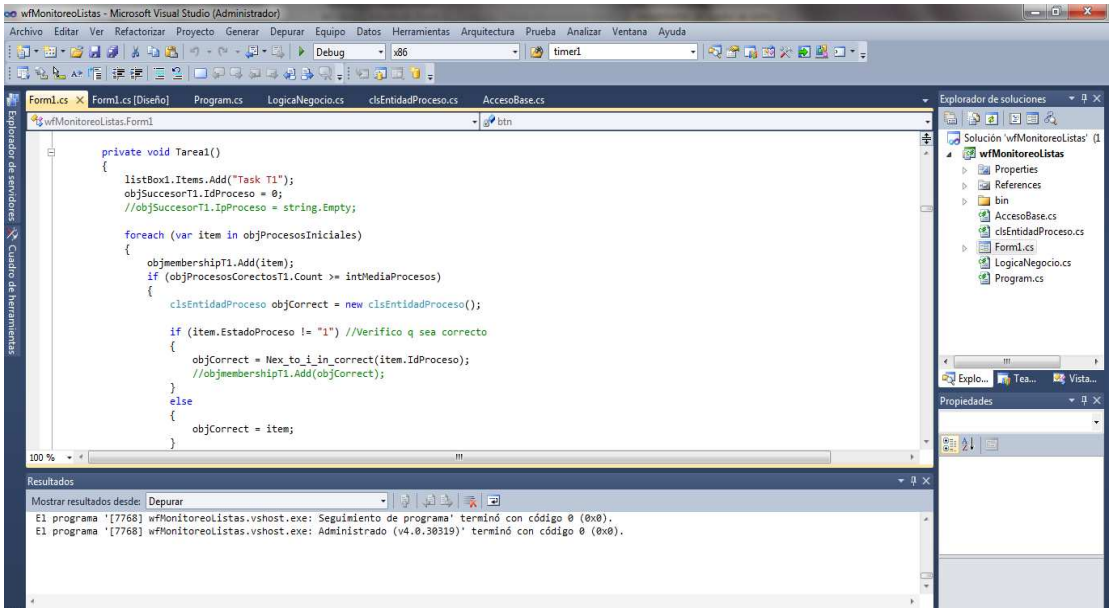


Figura 3.5 Código Fuente Tarea 1

La tarea 1 realiza un broadcast con cada uno de los nodos del sistema y verifica cuales nodos se encuentran activos y los coloca dentro del vector ***Correctos.*** De igual forma identifica el nodo sucesor en la secuencia formada por todos los nodos del sistema.

Código Fuente Tarea 1

//Método principal que llama a los métodos de ejecución automáticamente cada cierto tiempo (Timer) parametrizado a nivel de la pantalla.

```
//repeat forever each n time

private void Tarea1 ()

{

   listBox1.Items.Add ("Task T1"); //Muestra en pantalla el Inicio de la Tarea 1

    objSuccesorT1.IdProceso = 0; //Inicializa el Nodo sucesor con su id en cero

   objSuccesorT1.NombreProceso = "0"; //Inicializa el Nodo sucesor con
   nombre en cero

   foreach (var item in objProcesosIniciales) //Recorre todos los nodos de la red

   {

   objmembershipT1.Add (ítem); //Agrega los nodos a la lista de Miembros

    if (objProcesosCorectosT1.Count >= intMediaProcesos) //Valida si el
   numero de correctos es mayor a la media para continuar la ejecución y buscar
   el sucesor

      {

            clsEntidadProceso objCorrect = new clsEntidadProceso(); //Se crea un
            nuevo nodo para guardar el siguiente correcto que es el sucesor
```

```
            if (item.EstadoProceso!= "1") //Si el nodo no es correcto, se envía a
            buscar el siguiente correcto de toda la lista de nodos
        {
            objCorrect = Nex_to_i_in_correct (item.IdProceso); // Si ingreso por
            verdaderos agrego el siguiente nodo correcto
        }
        else
        {
            objCorrect = item; //Si la validación dio negativo este es el sucesor.
        }
objSuccesorT1 = Nex_to_i_in_correct(objCorrect.IdProceso);//Guardo el
nodo correcto como el sucesor.
 if (!entExist(objmembershipT1, objSuccesorT1)) //Valida si no existe el
sucesor no está en la lista de los miembros
            objmembershipT1.Add (objSuccesorT1); //Si no existe se agrega a la
            lista de los miembros
```

```
 if (objCorrect.IdProceso > (objProcesosIniciales.Count / 2)) //Se
valida nuevamente que los correctos sea mayor a la media pero esto
para ir mostrando en la pantalla

{

objProcesosCorectosT1.Add (objCorrect); //Agrego el nodo a la lista
de los correctos

 string strCorrectos = "";

 foreach (var itemCorrectos in objProcesosCorectosT1) //Recorre toda
la lista de correctos

 {

 strCorrectos = strCorrectos + itemCorrectos.NombreProceso + "  /  ";
//Concateno la cadena a mostrar con las IP's de los correctos separados
por slash (/)

 }

listBox1.Items.Add ("Correct {" + strCorrectos + "}"); // Muestro en
la pantalla los nodos correctos (Como texto)

listBox1.Items.Add ("Succesor {" + objSuccesorT1.NombreProceso +
"}"); //Muestro en la pantalla el sucesor (Como texto)
```

```
            break;
        }
    }
else //Si no cumple con los nodos correctos mayores a la media realiza el
Broadcast de cada nodo
    {
listBox1.Items.Add (strMensaje + " -> " + item.NombreProceso); // Se
muestra la Ip de la cual se realiza el Broadcast
        if (item.EstadoProceso == "1") //Valido si el Broadcast dio 1 que
        significa verdadero, si es verdadero agrego a los correctos caso contrario
        lo dejo por fuera
        {
            listBox1.Items.Add ("        " + strMensaje1); //Muestro en texto de
            pantalla si fue el Broadcast fue positivo
            objProcesosCorectosT1.Add (item); //Agrego a la lista de correctos
        }
        else
```

```
            {

                listBox1.Items.Add ("        " + strMensaje2); // Muestro en texto que el
                Broadcast fallo

            }

        }

    }

     if (string.IsNullOrEmpty (objSuccesorT1.IpProceso) &&
    objProcesosIniciales.Count > 0 &&
    objProcesosCorectosT1.Count>0)//Valida se tiene un nodo sucesor.

        objSuccesorT1 = Nex_to_i_in_correct (objProcesosIniciales.Count);
        //Guarda el sucesor del siguiente nodo correcto.

     listBox1.Items.Add ("------------------------------------------------------------");
    //Marca el fin de la ejecución de la Tarea 1, eso es cuando los correctos son
    mayores de la media o si se termino el análisis de los nodos de la red.

}
```

En la respectiva figura se detalla el código fuente de la tarea 2.

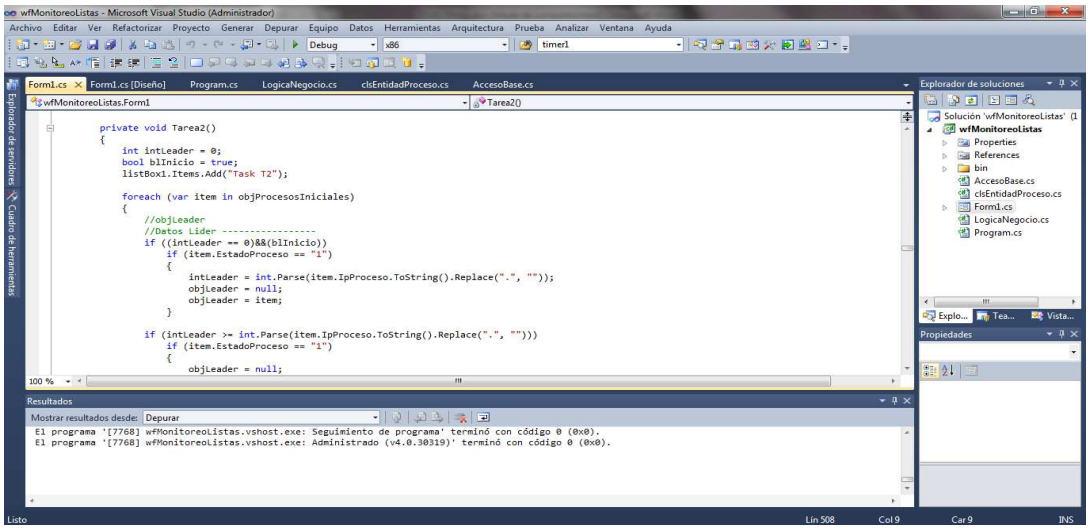


Figura 3.6 Código Fuente Tarea 2

La tarea 2 verifica si existen nuevos nodos y compara con los nodos existentes del vector ***Membership*** si algún nodo no se encuentra dentro de este vector agrega ese elemento y de igual forma selecciona un nodo líder tomando como consideración principal el nodo con el valor más pequeño del vector **Correct.**

Código Fuente Tarea 2

```
//Tarea 2, analiza todos los nodos de la red y obtiene el nodo líder (Nodo menor)

    private void Tarea2 ()

    {

       double intLeader = 0; // Inicializa líder en cero

       bool blInicio = true; //Controla si el menor es el inicio
```

```
listBox1.Items.Add ("Task T2"); //Muestra en pantalla el inicio de la Tarea 2

foreach (var item in objProcesosIniciales) //Recorre nodos mayores a la media y nuevos nodos para su análisis

{

if ((intLeader == 0) && (blInicio)) //Verifica si se encontró el nodo líder o si el recorrido de la lista regreso al inicio del nodo de la lista de nodos

if (item.EstadoProceso == "1")//Valido si el nodo es activo para tomarlo como líder

{

intLeader = double.Parse (item.IpProceso.ToString ().Replace (".", ""));  //Guardo la Ip como numero para poder validar si es el menor.

objLeader = null; //Inicializo en null el líder.

objLeader = item; //Guardo el nodo como líder.

}
```

```
                    if (intLeader >= double.Parse (item.IpProceso.ToString
                    ().Replace (".", "")))//Valido si el nodo actual es el menor con
                    el almacenado, si es verdadero debo tomar al menor

                     if (item.EstadoProceso == "1")//Si el actual es el menor valido
                    si es activo para tomarlo como el líder (Menor).

                    {

                     objLeader = null; //Inicializo en null el líder.

                     objLeader = item; //Guardo el nodo como líder.

                    intLeader = double.Parse (item.IpProceso.ToString ().Replace
                    ("."， ""))； //Guardo la Ip como numero para poder validar si es
                    el menor.

                     blInicio = false; // Cambio la variable para indicar que el
                    inicio no es el menor (Líder)

                    }

    //----------------------------

    //listBox1.Items.Add (item.IpProceso + “<<E>>  membership ");

 if (entExist(objmembershipT1, item))//Valido si es un nodo que ya fue
analizado (Nodo Con la misma información se rechaza)
```

```
{

}

else //Si es un nuevo nodo, muestro en pantalla como texto y luego guardo en las variables listas.

{

        //listBox1.Items.Add ("       Falso");

         listBox1.Items.Add (item.NombreProceso + "  <<No E>>
        membership "); // Muestro el texto que el nodo no está en los
        miembros (Es nuevo)

        listBox1.Items.Add ("membership <-- " +
        item.NombreProceso);//Muestro el nuevo nodo como miembro en la
        pantalla

        objmembershipT1.Add (item); //Agrego el nuevo nodo a los miembros

      if (item.EstadoProceso == "1") // Si el estado es 1 (Nodo activo) agrego
      a los correctos

       {
```

```
                listBox1.Items.Add (" correct  <-- correct U  " +
               item.NombreProceso); //Muestro en pantalla de texto que el nodo esta
               activo y fue agregado a la lista de correctos

                 objProcesosCorectosT1.Add(item);//Agrego el nuevo nodo a los
                 correctos.

                listBox1.Items.Add (" set timer (k) to timeout [k]"); //Muestra texto
               del tiempo de conexión y el time out que tiene el nodo.

            }

         }

      }

     listBox1.Items.Add ("Leader {" + objLeader.NombreProceso + "}"); //Se
    muestra el líder en pantalla

    listBox1.Items.Add ("<< expiración timer >>"); //Se muestra el tiempo de
    expiración del nodo

    listBox1.Items.Add ("------------------------------------------------------------");
    //Marca fin de la Tarea 2

}
```

3.4 Desarrollo del Algoritmo

Esta es la fase crucial del proyecto. En ella se desarrollará el algoritmo de detección de fallos Omega siguiendo el modelo de programación seleccionado.

El desarrollo del algoritmo incluye su comprensión, codificación, depuración y su optimización. Además, se deberán registrar los resultados obtenidos durante la ejecución.

Para la simulación del algoritmo se utilizará el lenguaje C#, el cual es un lenguaje orientado a objetos, flexible y potente diseñado para generar programas sobre dicha plataforma.

El lenguaje seleccionado para el desarrollo del este proyecto, es un lenguaje de programación moderno y orientado a objetos que combina la alta productividad con la flexibilidad, sin restringir la plataforma a utilizar y cuyo objetivo principal es reducir los costos y los tiempos de desarrollo de los servicios .NET, facilitando la detección de errores y siendo capaz de crear una gran variedad de aplicaciones ya sea de tipo consola, Windows o Web.

SQL SERVER 2005 como motor de base de datos

En el actual proyecto se ha escogido **SQL Server 2005** como lenguaje de base de datos, ya que constituye un lenguaje normalizado, que brinda la posibilidad de ubicar código apropiadamente en relación a su funcionalidad, ofreciendo una ventaja significativa al momento de integrarse con .NET Framework y lenguajes modernos

de programación como C# al ofrecer la posibilidad de desarrollar diversas aplicaciones presentando una interfaz que permita la ejecución de declaraciones SQL y la invocación de funciones y procedimientos mejorando de esta manera la seguridad, confiabilidad de integridad del producto que se va a construir.

Beneficios

La integración entre SQL Server y C# brinda varios beneficios importantes dentro de los cuales se menciona los siguientes:

- **Mejora en el modelo de programación:** SQL constituye un lenguaje compatible con .NET Framework y ofrece mejoras en la construcción y capacidad para el desarrollo.
- **Ambiente de desarrollo común:** El desarrollo de base de datos está integrado con el ambiente de desarrollo de Visual Studio.

3.5 Implementación de la Interfaz

El simulador consta de la siguiente interfaz.

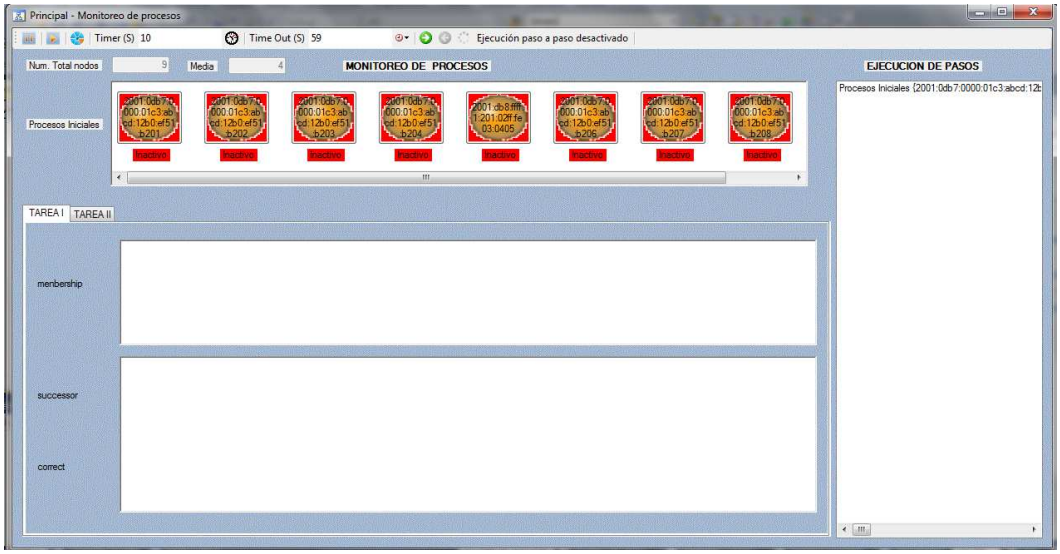


Figura 3.7 Interfaz del Simulador

En la figura 3.7 se puede observar la interfaz del simulador del Algoritmo Omega la cual permitirá visualizar en pantalla el funcionamiento del mismo y la elección de un nodo líder aplicado a redes WSN.

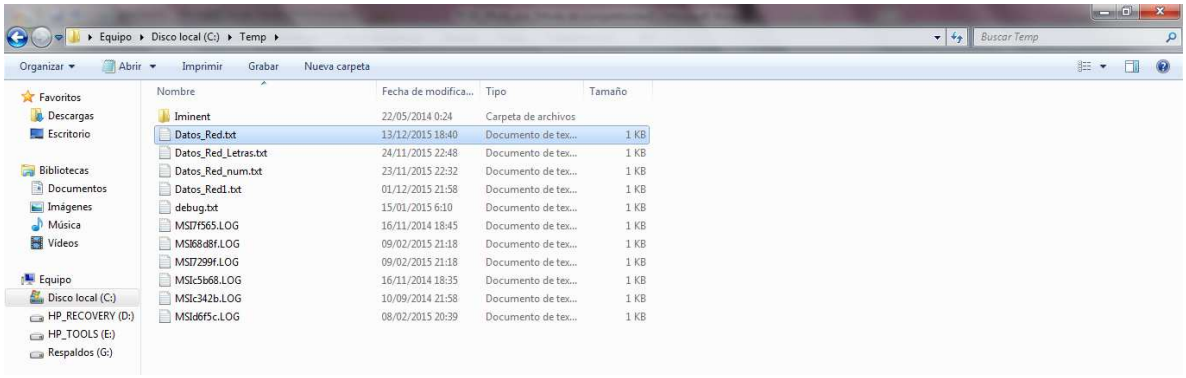


Figura 3.8 Archivo de Configuración del Simulador

La activación o desactivación de los nodos funciona mediante el archivo **Datos_red.txt** el cual se encuentra ubicado en la ruta C:\Temp el cual se puede

visualizar en la figura anterior. El archivo anteriormente mencionado contiene la siguiente estructura:

- Dirección IP (Puede ser IPv4 o IPv6)
- Nombre de la máquina
- Fecha y hora actual

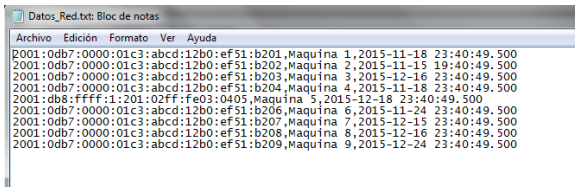


Figura 3.9 Parámetros de Configuración del Simulador

Para activar un nodo es necesario que la fecha colocada en el archivo **Datos_Red.txt** para ese nodo corresponda a la fecha actual. Si la fecha es menor a la fecha actual el nodo permanecerá en estado inactivo. Para este caso se puede observar que todos los nodos se encuentran en estado inactivo en la figura 3.9.

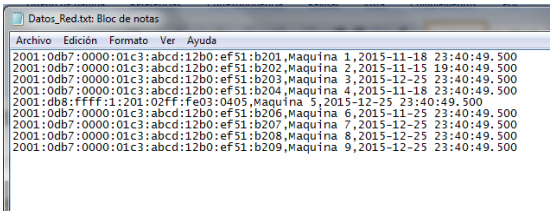


Figura 3.10 Activación de Nodos

En la figura anterior se puede observar que se procedió con la activación de los nodos correspondientes a las siguientes máquinas:

- Máquina 3
- Máquina 5
- Máquina 7
- Máquina 8
- Máquina 9

Para agregar un nodo nuevo el usuario deberá agregar los datos: Dirección IP, nombre de la máquina y fecha dentro del archivo de configuración **Datos_Red.txt** del simulador, guardarlo y proceder a ejecutar desde la respectiva interfaz.

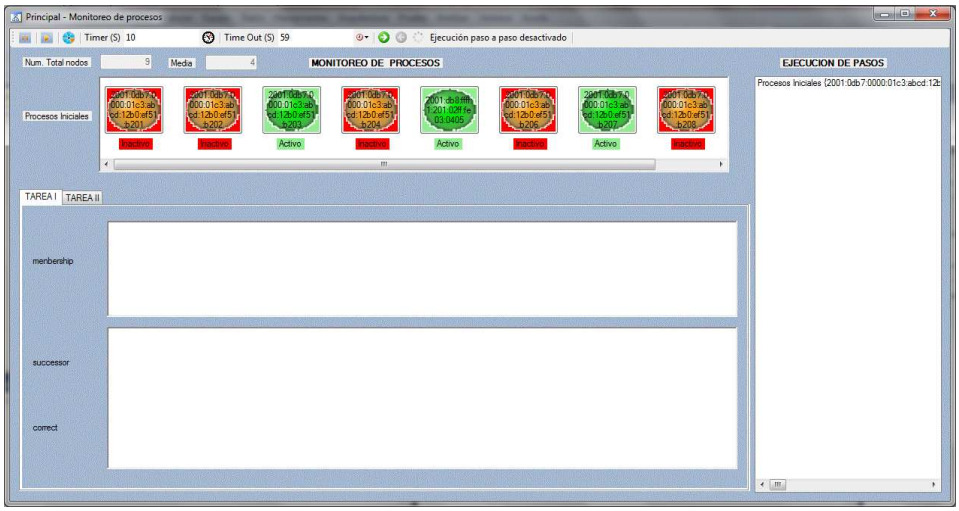


Figura 3.11 Visualización de Nodos Activos

En la figura anterior se puede visualizar que después de realizar la respectiva activación de los nodos en el archivo **Datos_Red.txt,** los nodos de color verde se encuentran activos. Las direcciones IP de los nodos activos se detallan a continuación:

- 2001:0db7:0000:01c3:abcd:12b0:ef51:b203
- 2001:db8:ffff:1:201:02ff:fe03:0405
- 2001:0db7:0000:01c3:abcd:12b0:ef51:b207
- 2001:0db7:0000:01c3:abcd:12b0:ef51:b208
- 2001:0db7:0000:01c3:abcd:12b0:ef51:b209

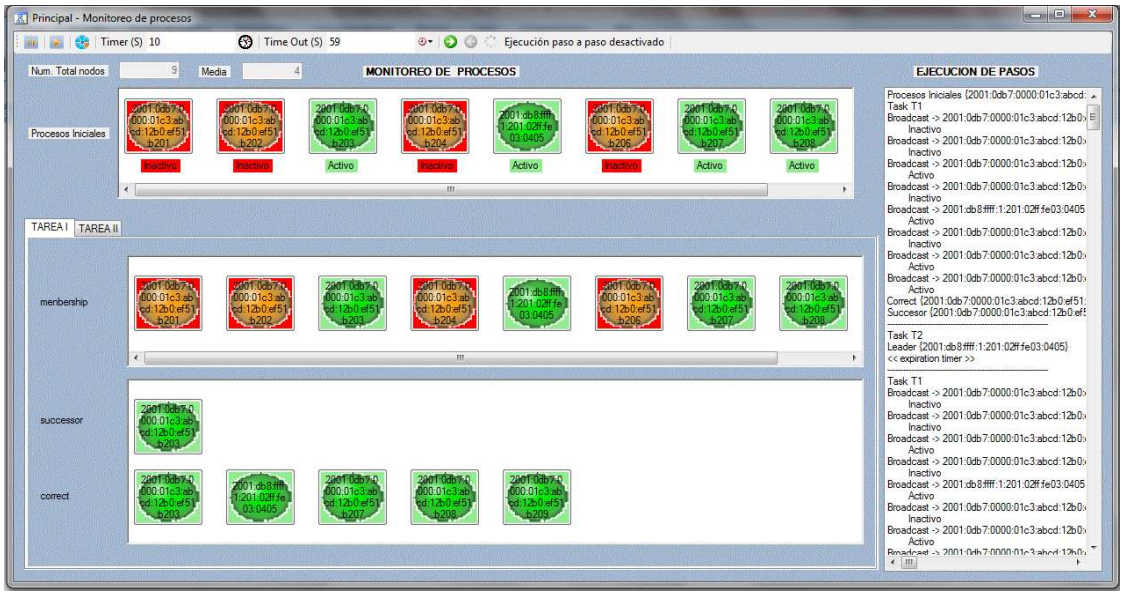


Figura 3.12 Visualización Resultados Tarea 1

En la figura 3.12 se puede visualizar los resultados finales correspondientes a la Tarea 1.

En la parte izquierda se pueden observar los valores de cada unos de los vectores del Algoritmo: **Membership, Sucessor y Correct.**

El vector **Membership** contiene los nodos activos e inactivos.

El vector **Correct** contiene únicamente los nodos activos.

La variable **sucessor** contiene el valor del nodo siguiente en la secuencia de nodos del vector **Membership.**

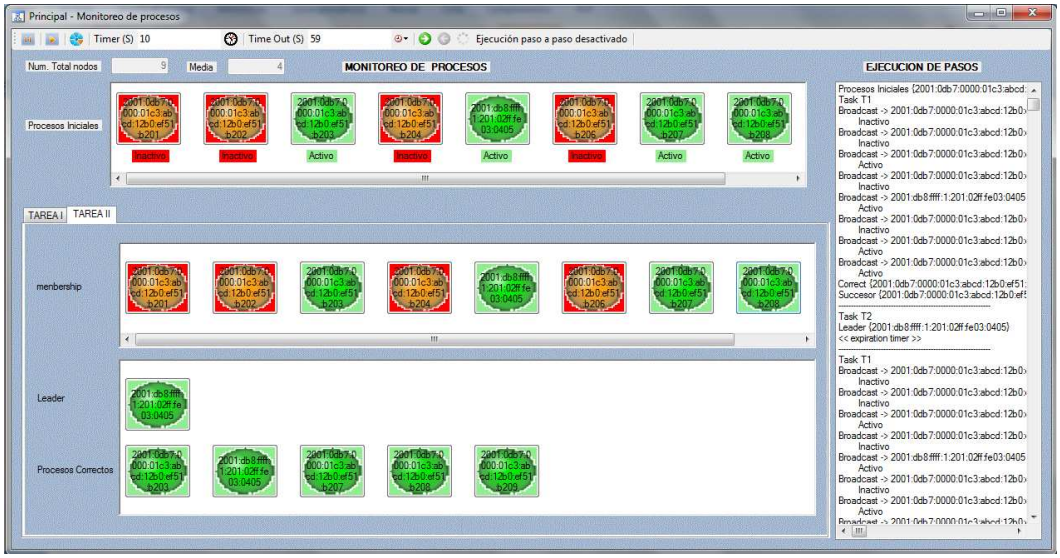


Figura 3.13 Visualización Resultados Tarea 2

La interfaz del simulador consta de 2 fichas: **TAREA I y TAREA II.** La figura 3.13 muestra los resultados finales correspondientes a la Tarea 2. Al pulsar sobre la ficha **TAREA II** se visualizan en pantalla el valor del nodo líder del Algoritmo Omega, el cual es el valor más pequeño del vector **Correct.**

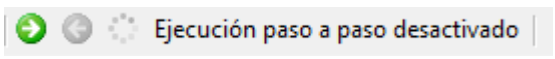


Figura 3.14 Barra de Ejecución del Simulador

El simulador consta de 2 tipos de ejecuciones: **Ejecución Automática** y **Ejecución paso a paso.** La figura anterior muestra la barra ubicada en la parte superior la cual

permite ejecutar el programa de acuerdo a la necesidad. Al pulsar sobre la fecha de color verde cambia el tipo de ejecución del programa.

Cuando se activa la **ejecución paso a paso** la **ejecución automática** se desactiva y viceversa. En este caso la **Ejecución paso a paso** se encuentra desactivada en la figura 3.14.

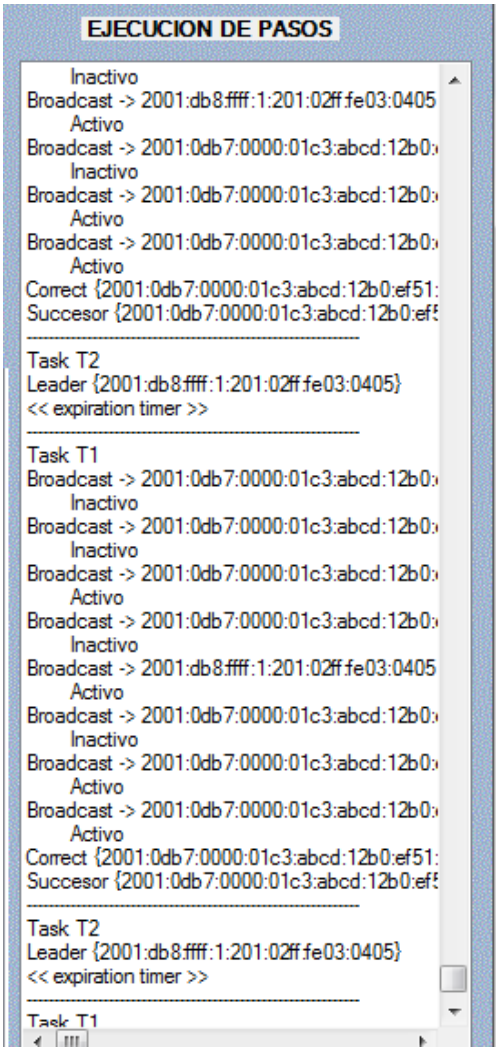


Figura 3.15 Ejecución de Pasos

La figura 3.15 muestra el panel de resultados de la **Ejecución paso a paso** del simulador el cual se encuentra ubicado en el lado derecho de la pantalla. Este panel muestra los valores de cada uno de los vectores y de la variable **leader.** La cual contiene la dirección IP con el valor más pequeño del vector **Correct.**

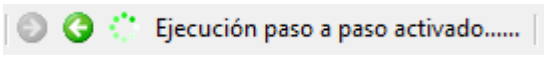


Figura 3.16 Activación de la ejecución paso a paso

La figura 3.16 muestra la activación de la **ejecución paso a paso** del simulador.

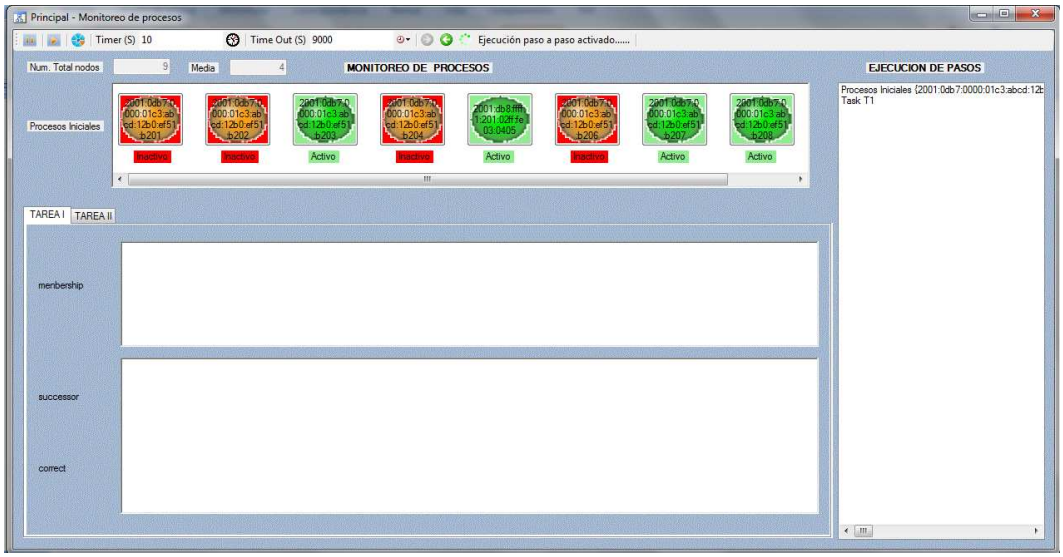


Figura 3.17 Ejecución paso a paso

En la figura anterior se muestra la activación de la **ejecución paso a paso.** Para visualizar los resultados de la **ejecución paso a paso** es necesario pulsar sobre el botón correspondiente al reloj.

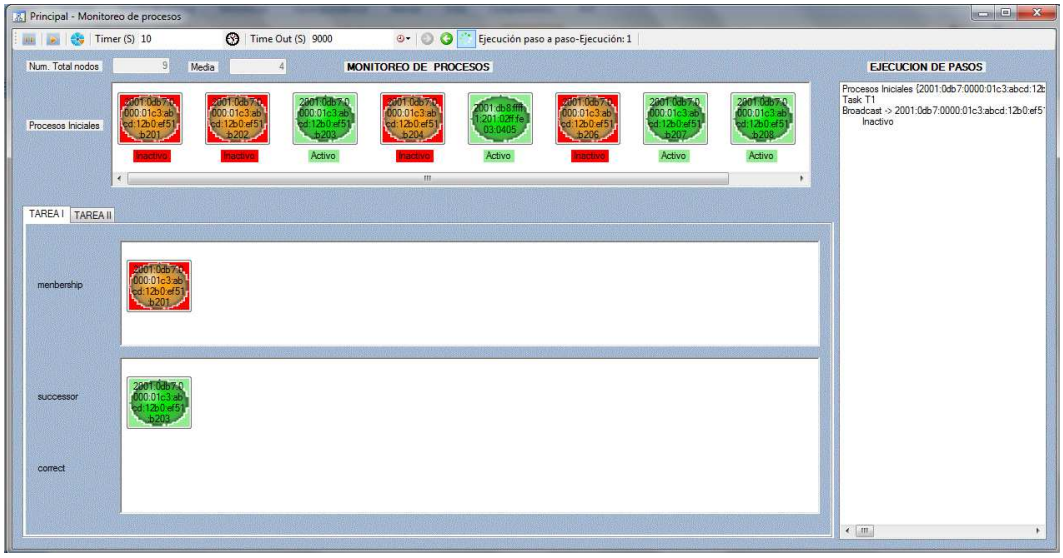


Figura 3.18 Ejecución paso a paso N°1

La figura anterior muestra los resultados de la primera **ejecución paso a paso** del simulador. En este caso se puede observar que el vector **Membership** contiene el valor del primer nodo cuya dirección IP es:

2001:0db7:0000:01c3:abcd:12b0:ef51:b201. La variable sucesor contiene el nodo siguiente de la secuencia de nodos con dirección IP:

2001:0db7:0000:01c3:abcd:12b0:ef51:b203.

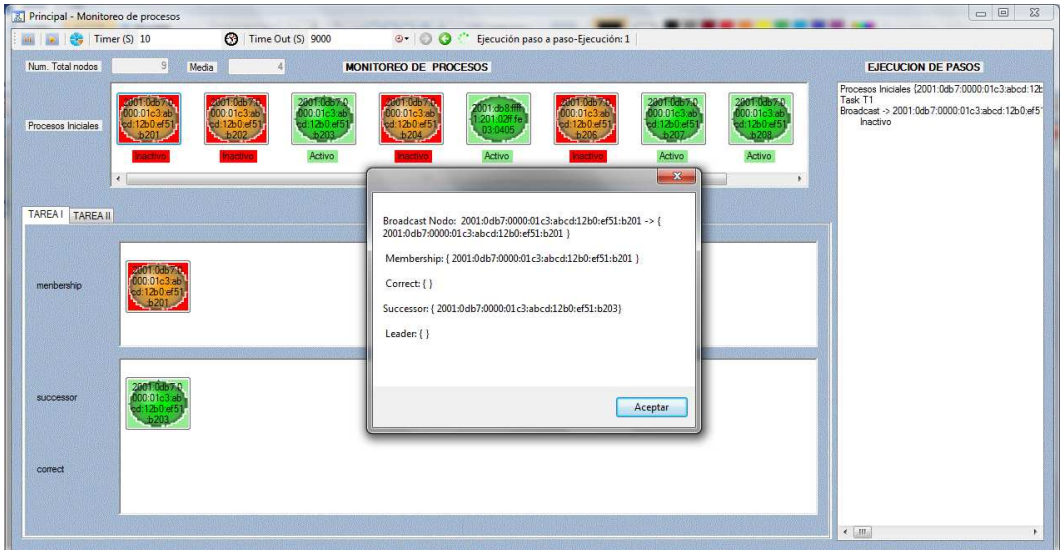


Figura 3.19 Visualización Resultados Vectores Nodo 1

En la figura 3.19 se puede observar los resultados de cada uno de los vectores correspondientes al nodo 1. Es importante mencionar que al dar clic sobre cada uno de los nodos se puede visualizar en pantalla una ventana emergente con los valores de los vectores: **Membership, Correct** y variables **Sucessor y Leader.**

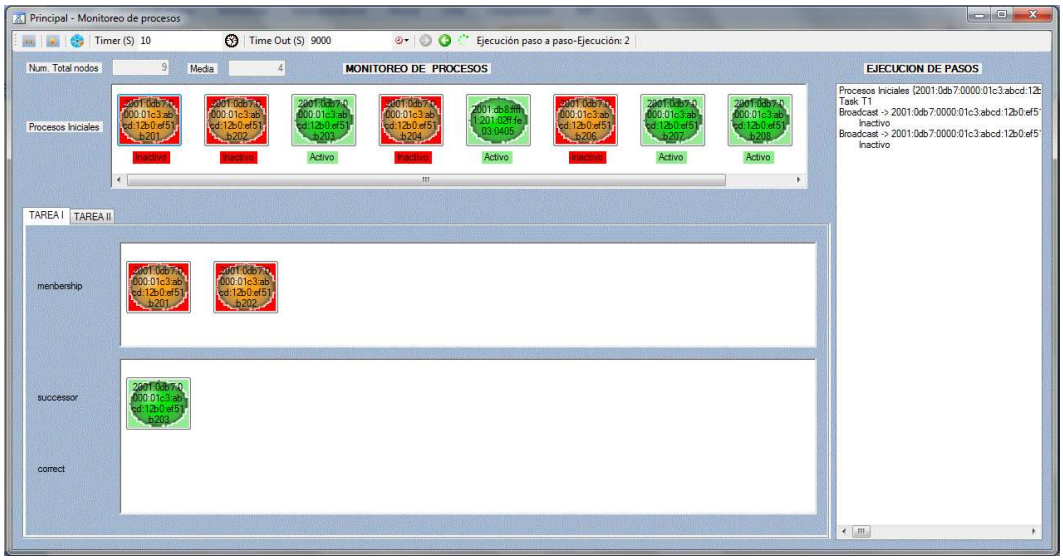


Figura 3.20 Visualización Resultados Ejecución N°2

En la figura 3.20 se puede observar los nodos con sus respectivas direcciones IP del vector **Membership** correspondientes al nodo 2, al igual que el valor de la variable **Sucessor.**

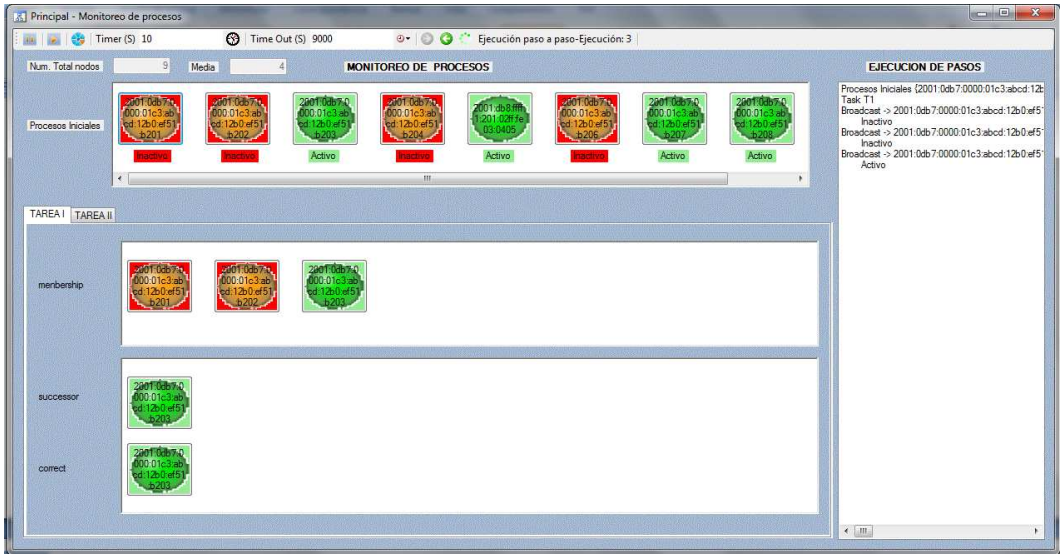


Figura 3.21 Visualización Resultados Ejecución N°3

En la figura 3.21 se puede observar los nodos con sus respectivas direcciones IP del vector **Membership** correspondientes al nodo 3, al igual que el valor de la variable **Sucessor** y el vector **Correct.**

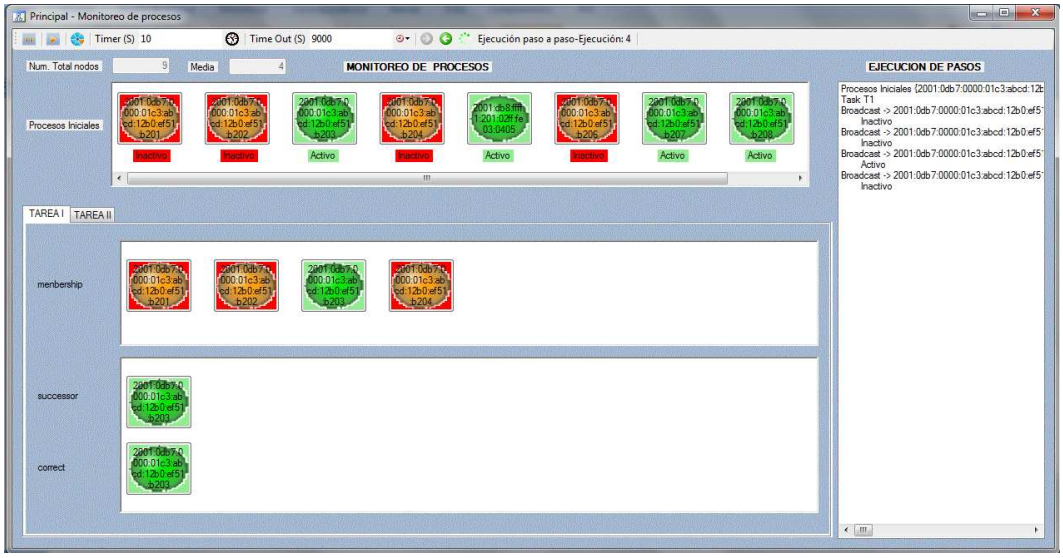


Figura 3.22 Visualización Resultados Ejecución N°4

En la figura 3.22 se puede observar los nodos con sus respectivas direcciones IP del vector **Membership** correspondientes al nodo 4, al igual que el valor de la variable **Sucessor** y el vector **Correct.**

Figura 3.23 Visualización Resultados Ejecución N°5

En la figura 3.23 se puede observar los nodos con sus respectivas direcciones IP del vector **Membership** correspondientes al nodo 5, al igual que el valor de la variable **Sucessor** y el vector **Correct.**

Figura 3.24 Visualización Resultados Ejecución N°6

En la figura 3.24 se puede observar los nodos con sus respectivas direcciones IP del vector **Membership** correspondientes al nodo 6, al igual que el valor de la variable **Sucessor** y el vector **Correct.**

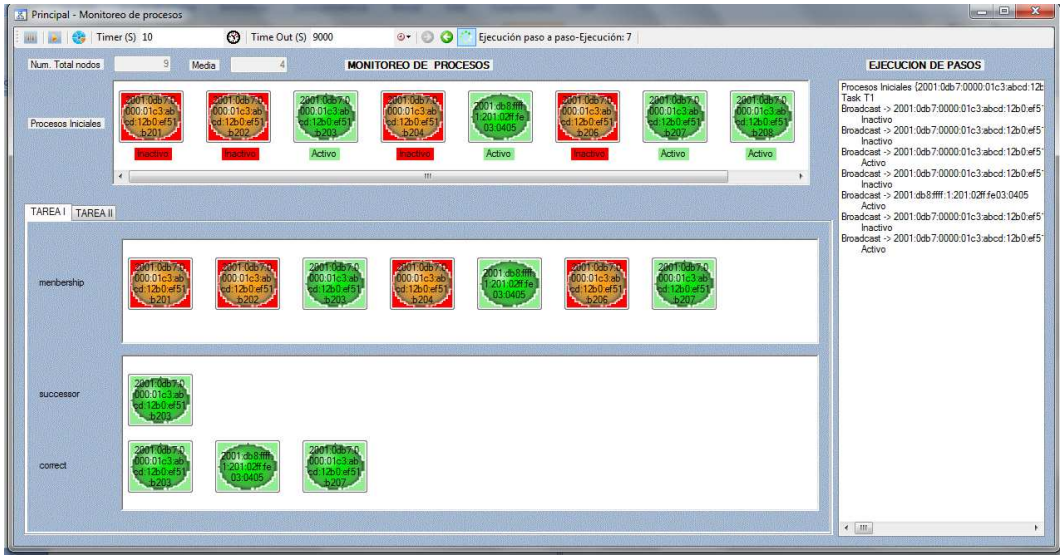


Figura 3.25 Visualización Resultados Ejecución N°7

En la figura 3.25 se puede observar los nodos con sus respectivas direcciones IP del vector **Membership** correspondientes al nodo 7, al igual que el valor de la variable **Sucessor** y el vector **Correct.**

Figura 3.26 Visualización Resultados Ejecución N°8

En la figura 3.26 se puede observar los nodos con sus respectivas direcciones IP del vector **Membership** correspondientes al nodo 8, al igual que el valor de la variable **Sucessor** y el vector **Correct.**

Figura 3.27 Visualización Resultados Ejecución N°9

En la figura 3.27 se puede observar los nodos con sus respectivas direcciones IP del vector **Membership** correspondientes al nodo 9, al igual que el valor de la variable **Sucessor** y el vector **Correct.**

Figura 3.28 Visualización Resultados Ejecución N°12 Tarea 1

En la figura 3.28 se puede observar los resultados finales correspondientes a la Tarea 1 de los vectores **Membership y Correct**, al igual que el valor de la variable **Sucessor.**

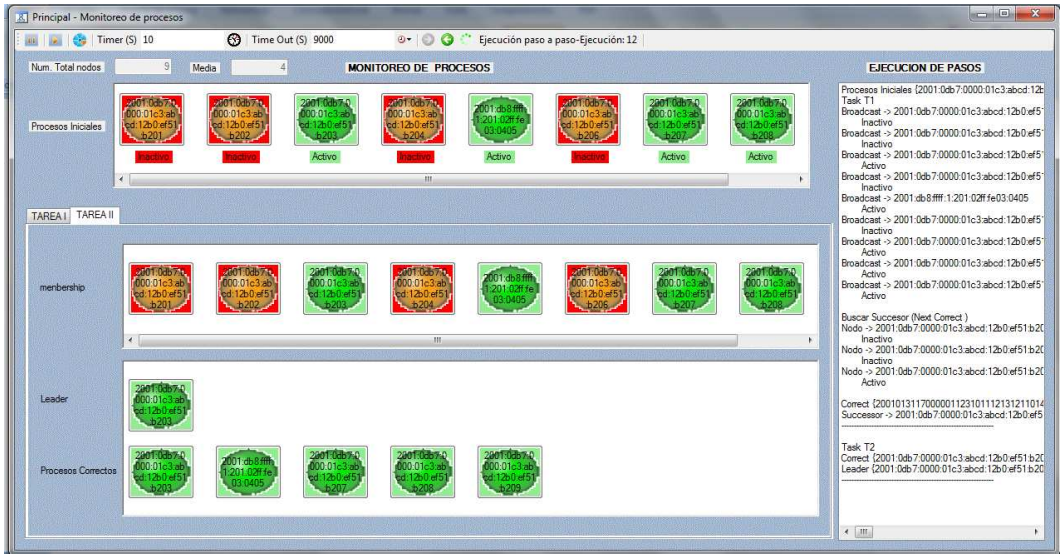


Figura 3.29 Visualización Resultados Ejecución N°12 Tarea 2

En la figura 3.29 se puede observar los resultados finales correspondientes a la Tarea 2 de los vectores **Membership, Correct**, al igual que el valor de la variable **Leader.**

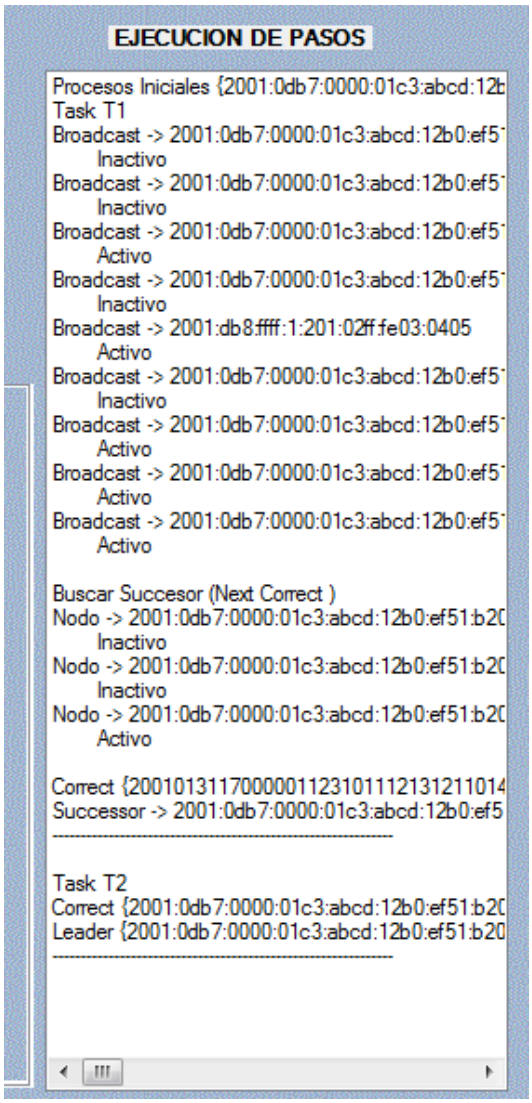


Figura 3.30 Visualización de Resultados Finales

En la figura 3.30 se puede observar el Panel ubicado en el lado derecho del simulador con los resultados finales del vector **Correct,** variable **Sucessor** para la Tarea 1 y 2. En este Panel se visualiza el **broadcast** realizado por cada uno de los nodos y si el **broadcast** fue exitoso o fallido.

CAPITULO IV: PRUEBAS, RESULTADOS Y CONCLUSIONES

4.1 Pruebas y Resultados

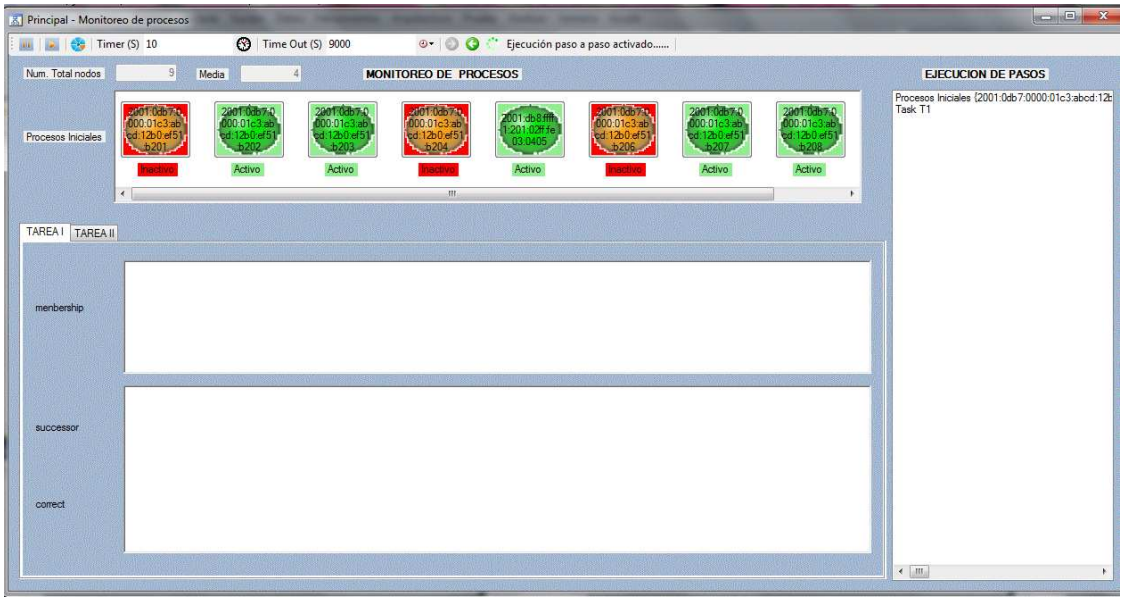


Figura 4.1.1 Activación de Nodos

Para la ejecución de las pruebas se empleó el simulador del algoritmo Omega el cual consta en su interfaz de dos tipos de ejecuciones: automática y paso a paso. La ejecución automática permite visualizar los resultados finales y la ejecución paso a paso permitirá visualizar en pantalla un seguimiento de la secuencia de pasos realizada hasta obtener los resultados. El simulador consta de 2 tipos de estado para los nodos: activo (verde) y fallido (anaranjado). Para la primera prueba se han activado los siguientes nodos:

2001:0db7:0000:01c3:abcd:12b0:ef51:b201 (Inactivo)

2001:0db7:0000:01c3:abcd:12b0:ef51:b202 (Activo)

2001:0db7:0000:01c3:abcd:12b0:ef51:b203 (Activo)

2001:0db7:0000:01c3:abcd:12b0:ef51:b204 (Inactivo)

2001:db8:ffff:1:201:02ff:fe03:0405 (Activo)

2001:0db7:0000:01c3:abcd:12b0:ef51:b206 (Inactivo)

2001:0db7:0000:01c3:abcd:12b0:ef51:b207 (Activo)

2001:0db7:0000:01c3:abcd:12b0:ef51:b208 (Activo)

2001:0db7:0000:01c3:abcd:12b0:ef51:b209 (Inactivo)

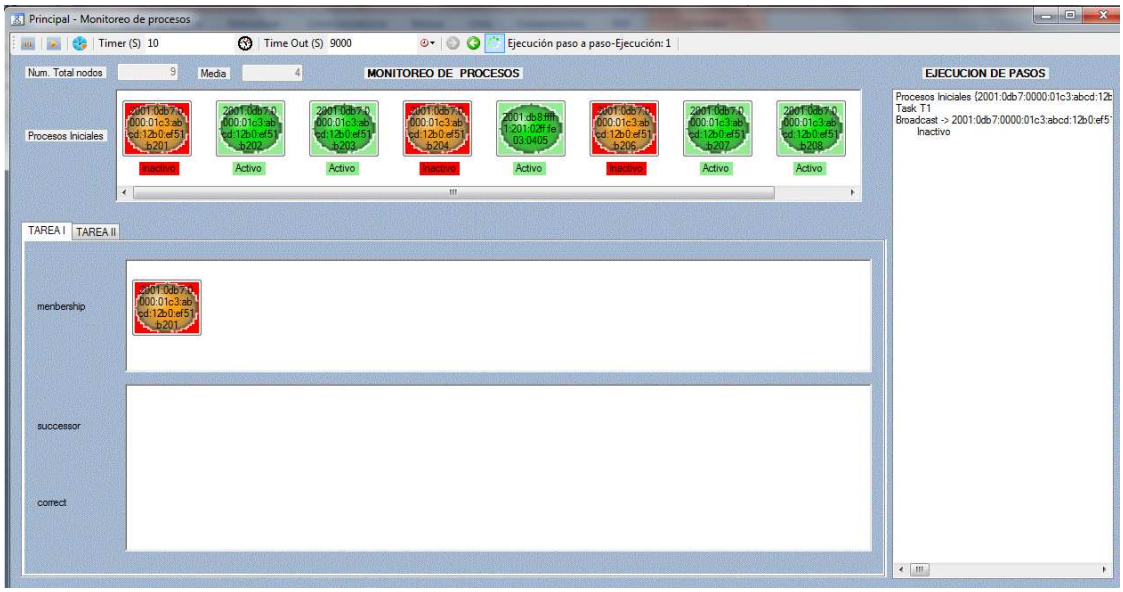


Figura 4.1.2 Ejecución N°1

En la primera ejecución se puede visualizar que el vector ***Membership*** contiene el nodo cuya dirección IP es la:

2001:0db7:0000:01c3:abcd:12b0:ef51:b201. El vector ***Miembros*** está formado por todos los nodos sean activos o inactivos.

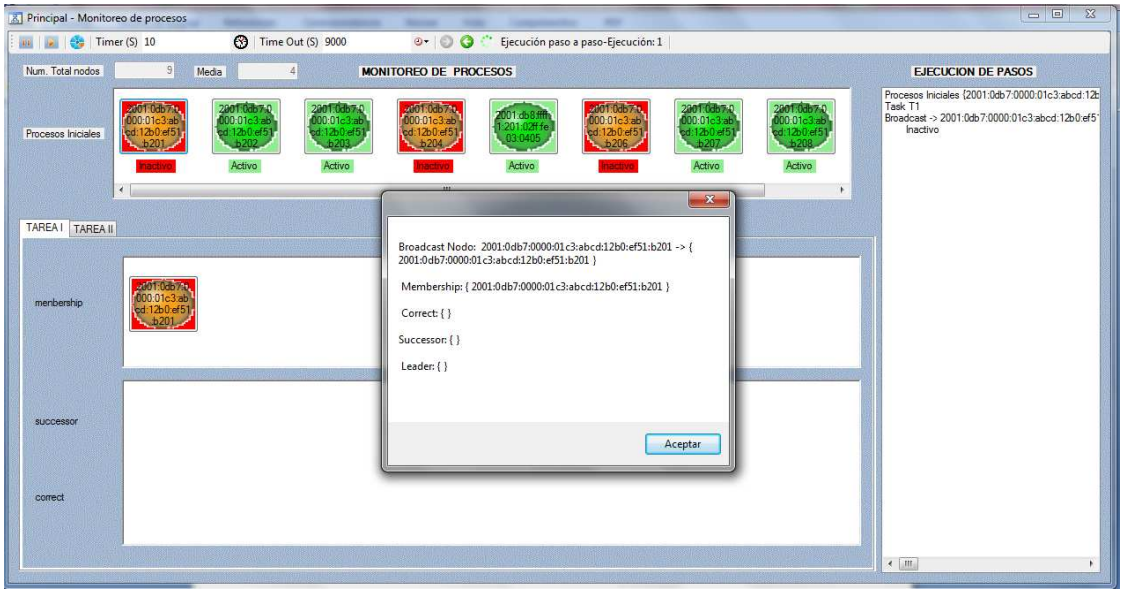


Figura 4.1.3 Ejecución N°1 Visualización de Resultados

A continuación se detallan los valores de los vectores ***Membership, Correct, Sucessor y Leader.***

Membership {2001:0db7:0000:01c3:abcd:12b0:ef51:b201}

Correct { }

Sucessor { }

Leader { }

Figura 4.1.4 Ejecución N°2

En la figura 4.1.4 el vector ***Membership*** está conformado por los siguientes nodos:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b201,

2001:0db7:0000:01c3:abcd:12b0:ef51:b202}

El vector ***Correct*** está formado por los nodos activos, en este caso:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b202}

Los vectores ***Sucessor y Leader*** no contienen ningún nodo.

Sucessor { }

Leader { }

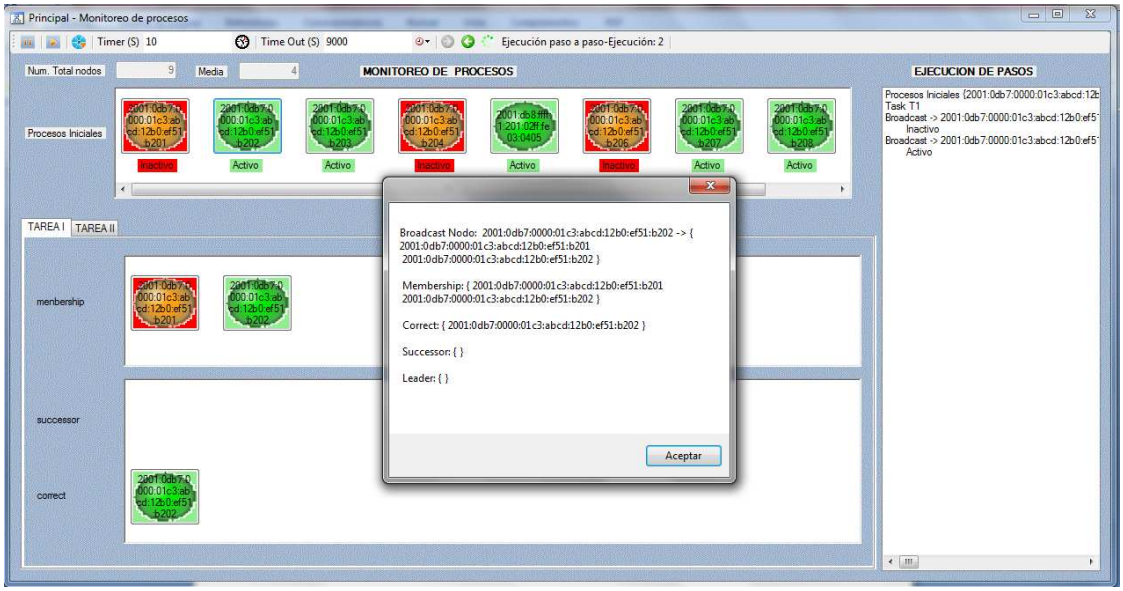


Figura 4.1.5 Ejecución N°2 Visualización de Resultados

En la figura 4.1.5 se pueden visualizar los vectores ***Membership***, ***Correct***, ***Sucessor*** *y* ***Leader*** con sus respectivos valores.

Figura 4.1.6 Ejecución N°3

En la figura 4.1.6 el vector ***Membership*** está conformado por los siguientes nodos:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b201,

2001:0db7:0000:01c3:abcd:12b0:ef51:b202,

2001:0db7:0000:01c3:abcd:12b0:ef51:b203}

El vector ***Correct*** está formado por los nodos:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b202,

2001:0db7:0000:01c3:abcd:12b0:ef51:b203}

Los vectores ***Sucessor*** *y* ***Leader*** no contienen ningún nodo.

Sucessor { }

Leader { }

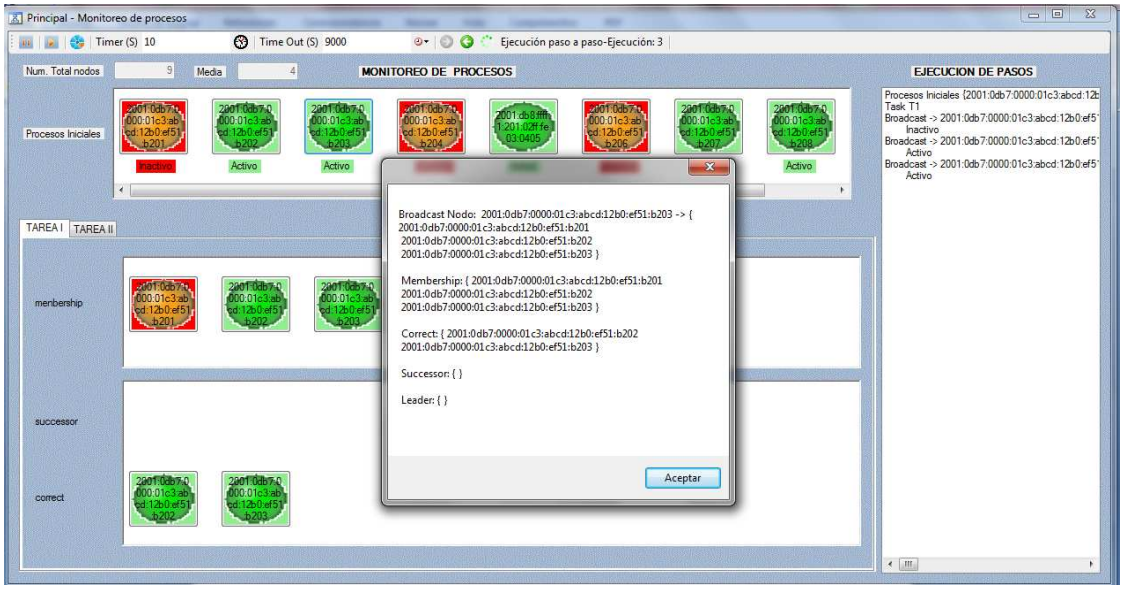


Figura 4.1.7 Ejecución N°3 Visualización de Resultados

En la figura 4.1.7 se pueden visualizar los vectores ***Membership***, ***Correct***, ***Sucessor*** *y* ***Leader*** con sus respectivos valores.

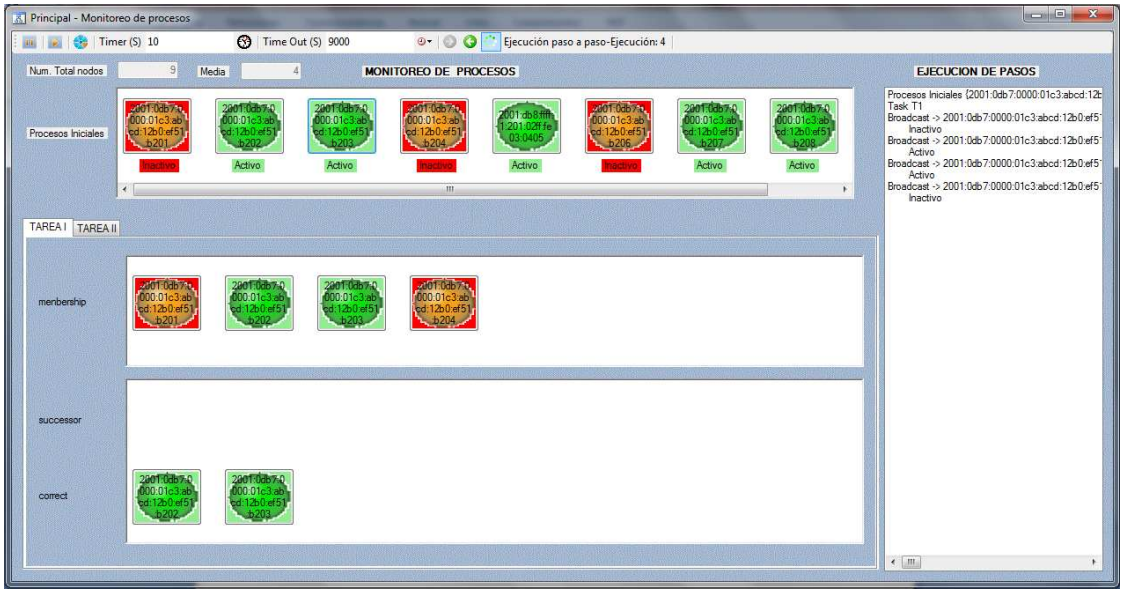


Figura 4.1.8 Ejecución N°4

En la figura 4.1.8 El vector ***Membership*** está conformado por los siguientes nodos:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b201,

2001:0db7:0000:01c3:abcd:12b0:ef51:b202,

2001:0db7:0000:01c3:abcd:12b0:ef51:b203,

2001:0db7:0000:01c3:abcd:12b0:ef51:b204}

El vector ***Correct*** está formado por los nodos activos, en este caso:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b202,

2001:0db7:0000:01c3:abcd:12b0:ef51:b203}

Los vectores ***Sucessor*** *y* ***Leader*** no contienen ningún nodo.

Sucessor { }

Leader { }

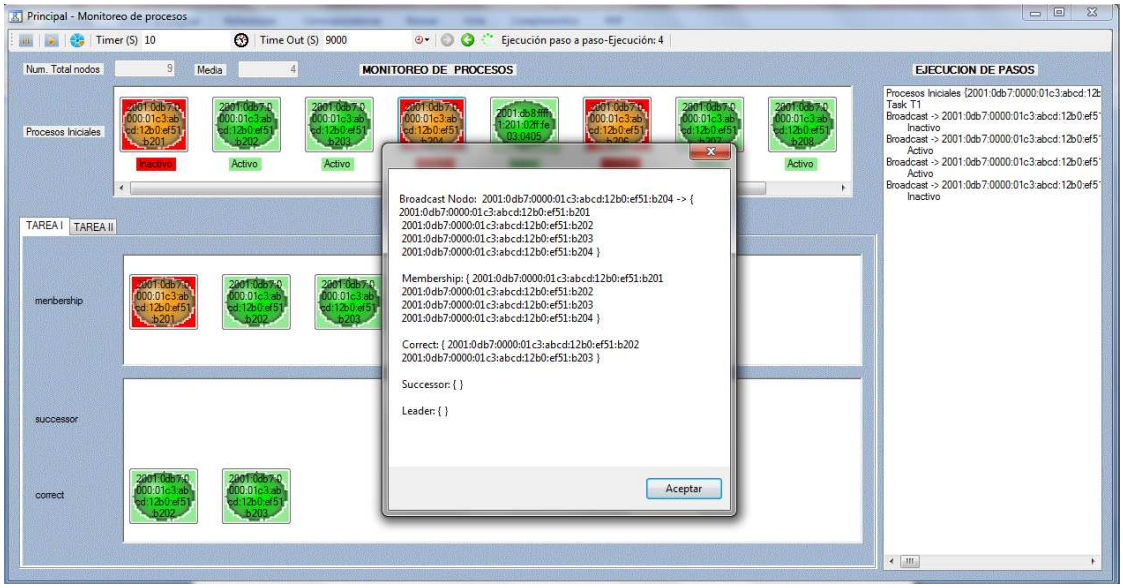


Figura 4.1.9 Ejecución N°4 Visualización de Resultados

En la figura 4.1.9 se pueden visualizar los vectores ***Membership***, ***Correct***, ***Sucessor*** *y* ***Leader*** con sus respectivos valores.

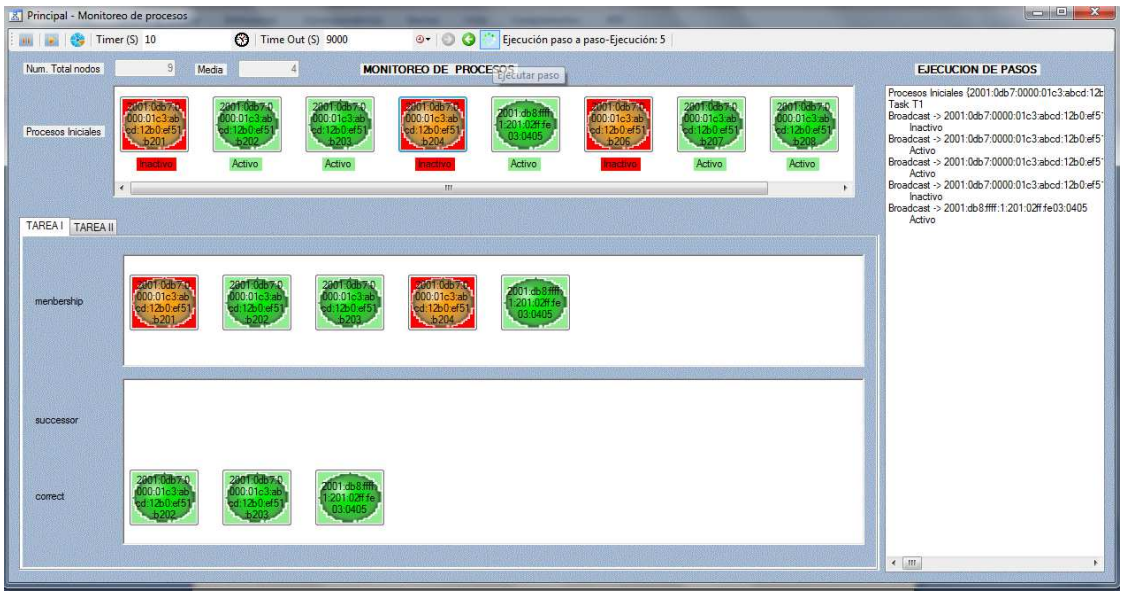


Figura 4.1.10 Ejecución N°5

En la figura 4.1.10 el vector ***Membership*** está conformado por los siguientes nodos:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b201,

2001:0db7:0000:01c3:abcd:12b0:ef51:b202,

2001:0db7:0000:01c3:abcd:12b0:ef51:b203,

2001:0db7:0000:01c3:abcd:12b0:ef51:b204,

2001:db8:ffff:1:201:02ff:fe03:0405}

El vector ***Correct*** está formado por los nodos activos, en este caso:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b202,

2001:0db7:0000:01c3:abcd:12b0:ef51:b203,

2001:db8:ffff:1:201:02ff:fe03:0405}

Los vectores ***Sucessor*** *y* ***Leader*** no contienen ningún nodo.

Sucessor { }

Leader { }

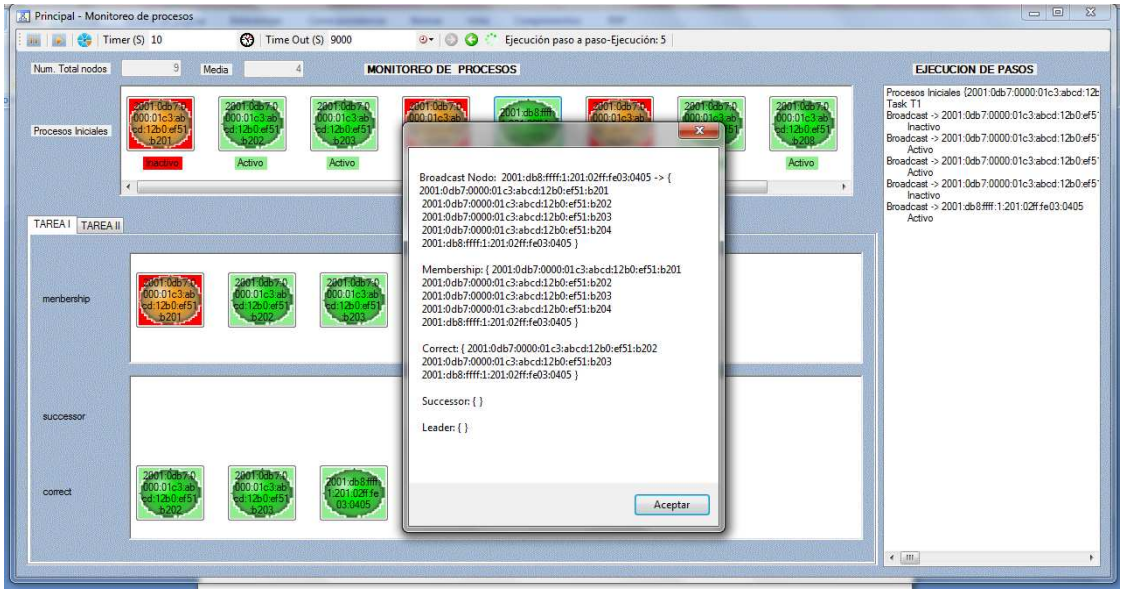


Figura 4.1.11 Ejecución N°5 Visualización de Resultados

En la figura 4.1.11 se pueden visualizar los vectores ***Membership***, ***Correct***, ***Sucessor*** *y* ***Leader*** con sus respectivos valores.

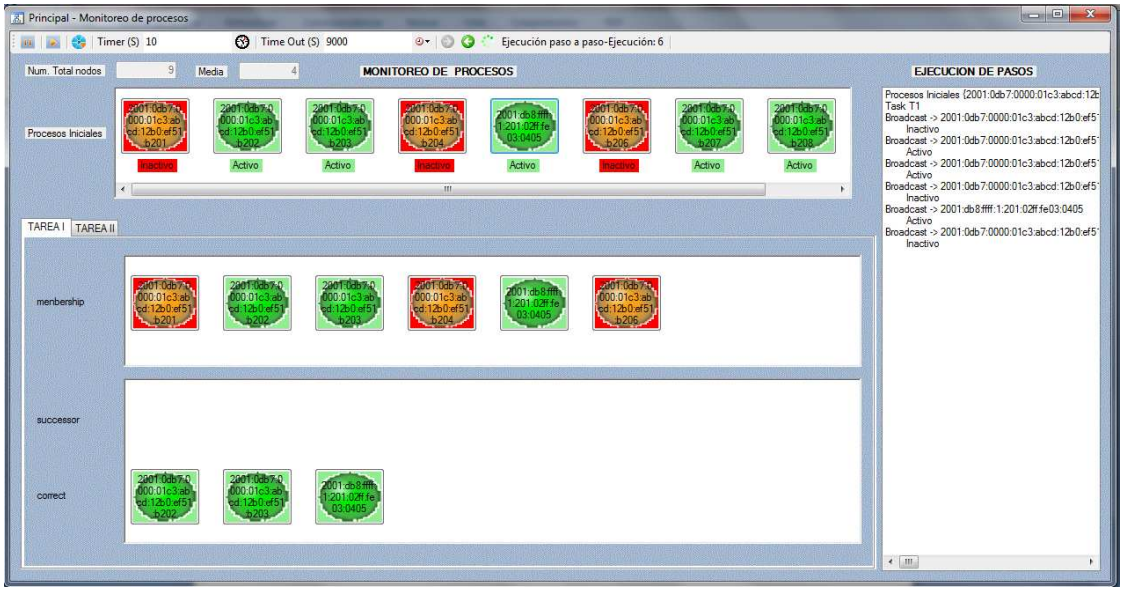


Figura 4.1.12 Ejecución N°6

En la figura anterior el vector ***Membership*** está conformado por los siguientes nodos:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b201,

2001:0db7:0000:01c3:abcd:12b0:ef51:b202,

2001:0db7:0000:01c3:abcd:12b0:ef51:b203,

2001:0db7:0000:01c3:abcd:12b0:ef51:b204,

2001:db8:ffff:1:201:02ff:fe03:0405,

2001:0db7:0000:01c3:abcd:12b0:ef51:b206}

El vector ***Correct*** está formado por los nodos activos, en este caso:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b202,

2001:0db7:0000:01c3:abcd:12b0:ef51:b203,

2001:db8:ffff:1:201:02ff:fe03:0405}

Los vectores ***Sucessor*** *y* ***Leader*** no contienen ningún nodo.

Sucessor { }

Leader { }

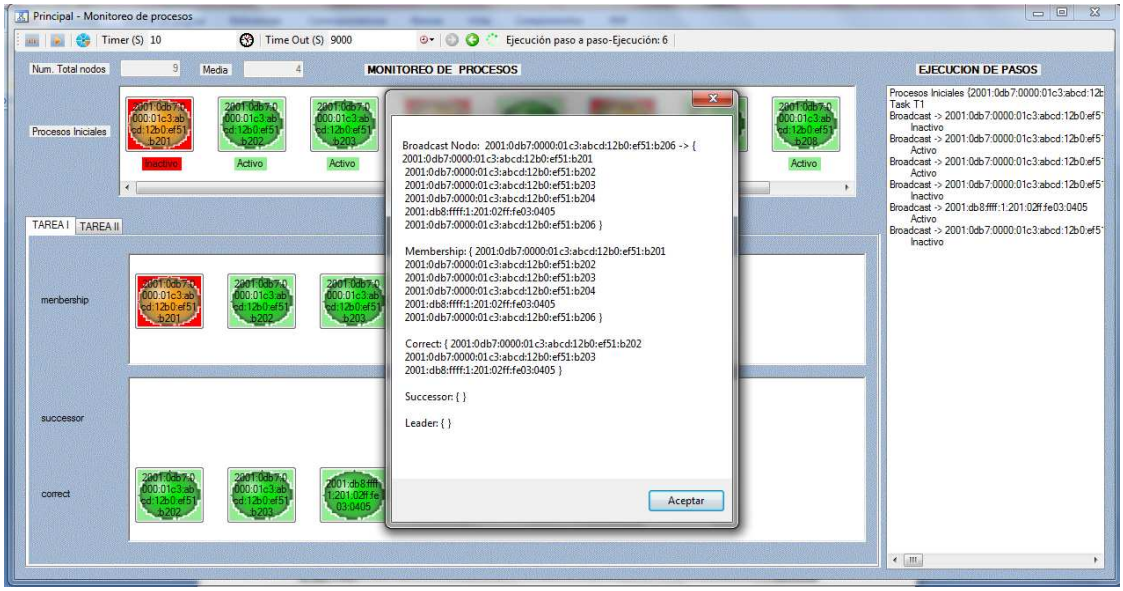


Figura 4.1.13 Ejecución N°6 Visualización de Resultados

En la figura 4.1.13 se pueden visualizar los vectores ***Membership***, ***Correct***, ***Sucessor*** *y* ***Leader*** con sus respectivos valores.

Figura 4.1.14 Ejecución N°7

En la figura anterior El vector ***Membership*** está conformado por los siguientes nodos:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b201,

2001:0db7:0000:01c3:abcd:12b0:ef51:b202,

2001:0db7:0000:01c3:abcd:12b0:ef51:b203,

2001:0db7:0000:01c3:abcd:12b0:ef51:b204,

2001:db8:ffff:1:201:02ff:fe03:0405,

2001:0db7:0000:01c3:abcd:12b0:ef51:b206,

2001:0db7:0000:01c3:abcd:12b0:ef51:b207}

El vector ***Correct*** está formado por los nodos activos, en este caso:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b202,

2001:0db7:0000:01c3:abcd:12b0:ef51:b203,

2001:db8:ffff:1:201:02ff:fe03:0405,

2001:0db7:0000:01c3:abcd:12b0:ef51:b207}

Los vectores ***Sucessor*** *y* ***Leader*** no contienen ningún nodo.

Sucessor { }

Leader { }

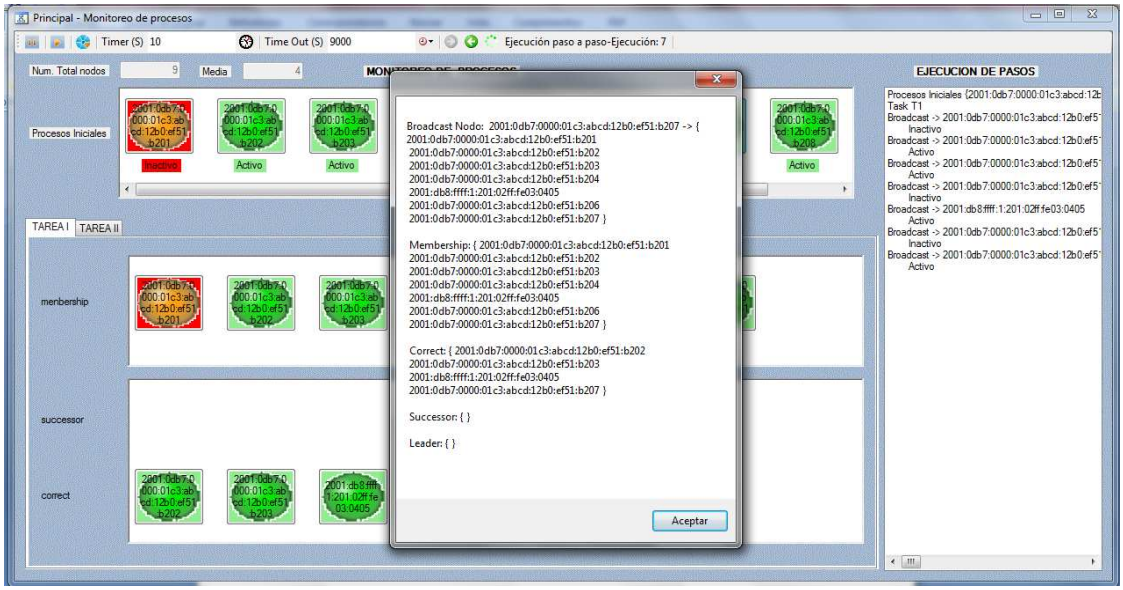


Figura 4.1.15 Ejecución N°7 Visualización de Resultados

En la figura 4.1.15 se pueden visualizar los vectores ***Membership***, ***Correct***, ***Sucessor*** *y* ***Leader*** con sus respectivos valores.

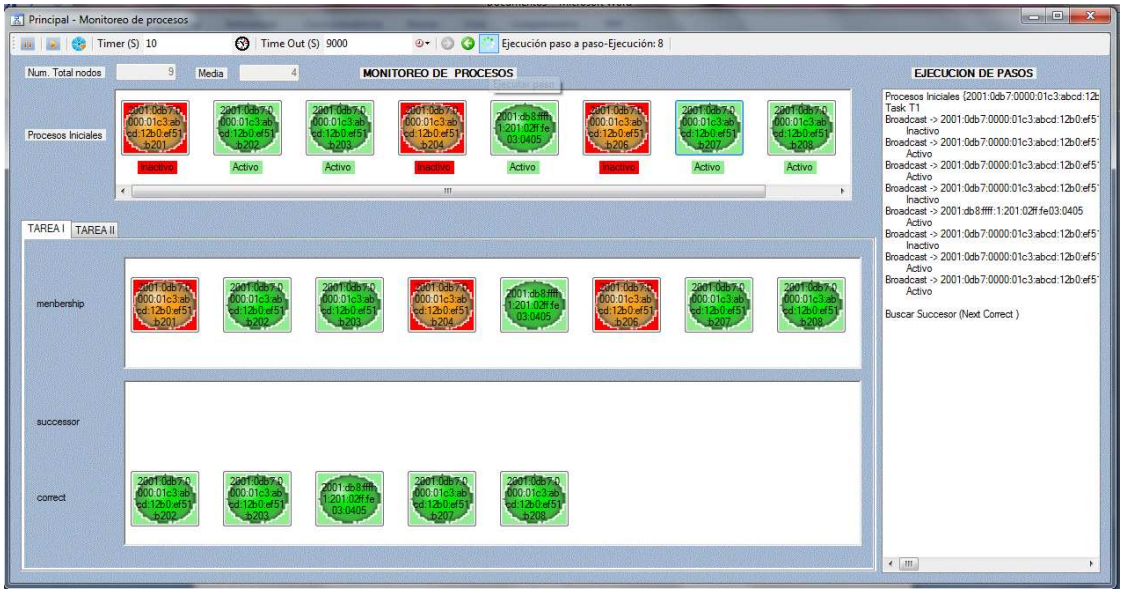


Figura 4.1.16 Ejecución N°8

En la figura anterior el vector ***Membership*** está conformado por los siguientes nodos:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b201,

2001:0db7:0000:01c3:abcd:12b0:ef51:b202,

2001:0db7:0000:01c3:abcd:12b0:ef51:b203,

2001:0db7:0000:01c3:abcd:12b0:ef51:b204,

2001:db8:ffff:1:201:02ff:fe03:0405,

2001:0db7:0000:01c3:abcd:12b0:ef51:b206,

2001:0db7:0000:01c3:abcd:12b0:ef51:b207,

2001:0db7:0000:01c3:abcd:12b0:ef51:b208}

El vector ***Correct*** está formado por los nodos activos, en este caso:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b202,

2001:0db7:0000:01c3:abcd:12b0:ef51:b203,

2001:db8:ffff:1:201:02ff:fe03:0405,

2001:0db7:0000:01c3:abcd:12b0:ef51:b207,

2001:0db7:0000:01c3:abcd:12b0:ef51:b208}

Los vectores ***Sucessor*** *y* ***Leader*** no contienen ningún nodo.

Sucessor { }

Leader { }

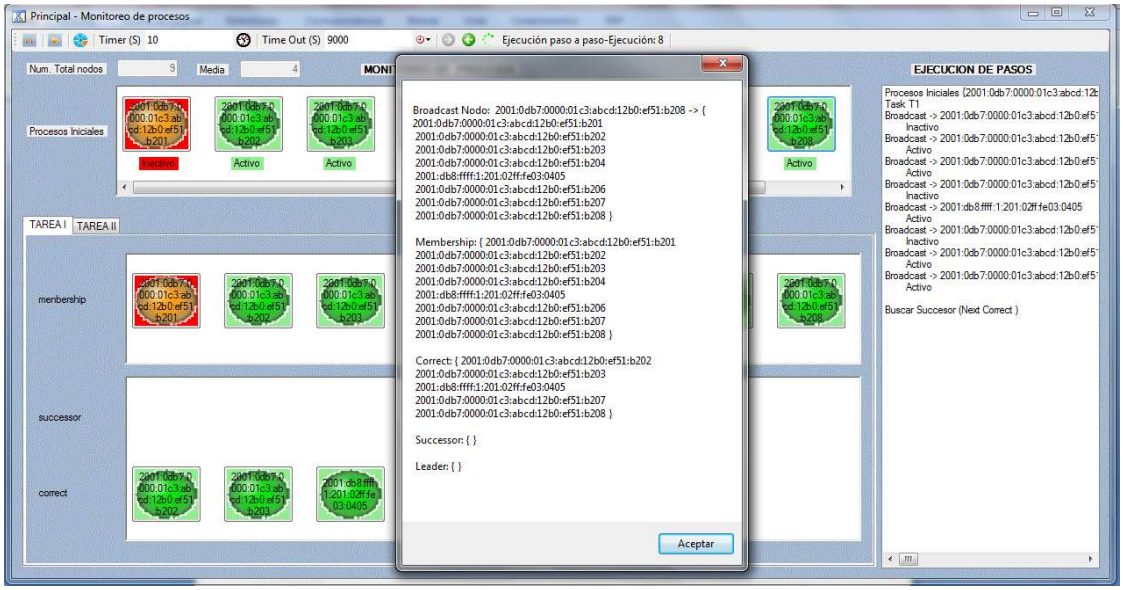


Figura 4.1.17 Ejecución N°8 Visualización de Resultados

En la figura 4.1.17 se pueden visualizar los vectores ***Membership***, ***Correct***, ***Sucessor*** *y* ***Leader*** con sus respectivos valores.

Figura 4.1.18 Ejecución N°9

En la figura anterior el vector ***Membership*** está conformado por los siguientes nodos:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b201,

2001:0db7:0000:01c3:abcd:12b0:ef51:b202,

2001:0db7:0000:01c3:abcd:12b0:ef51:b203,

2001:0db7:0000:01c3:abcd:12b0:ef51:b204,

2001:db8:ffff:1:201:02ff:fe03:0405,

2001:0db7:0000:01c3:abcd:12b0:ef51:b206,

2001:0db7:0000:01c3:abcd:12b0:ef51:b207,

2001:0db7:0000:01c3:abcd:12b0:ef51:b208,

2001:0db7:0000:01c3:abcd:12b0:ef51:b209}

El vector ***Correct*** está formado por los nodos activos, en este caso:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b202,

2001:0db7:0000:01c3:abcd:12b0:ef51:b203,

2001:db8:ffff:1:201:02ff:fe03:0405,

2001:0db7:0000:01c3:abcd:12b0:ef51:b207,

2001:0db7:0000:01c3:abcd:12b0:ef51:b208}

Los vectores ***Sucessor*** *y* ***Leader*** no contienen ningún nodo.

Sucessor { }

Leader { }

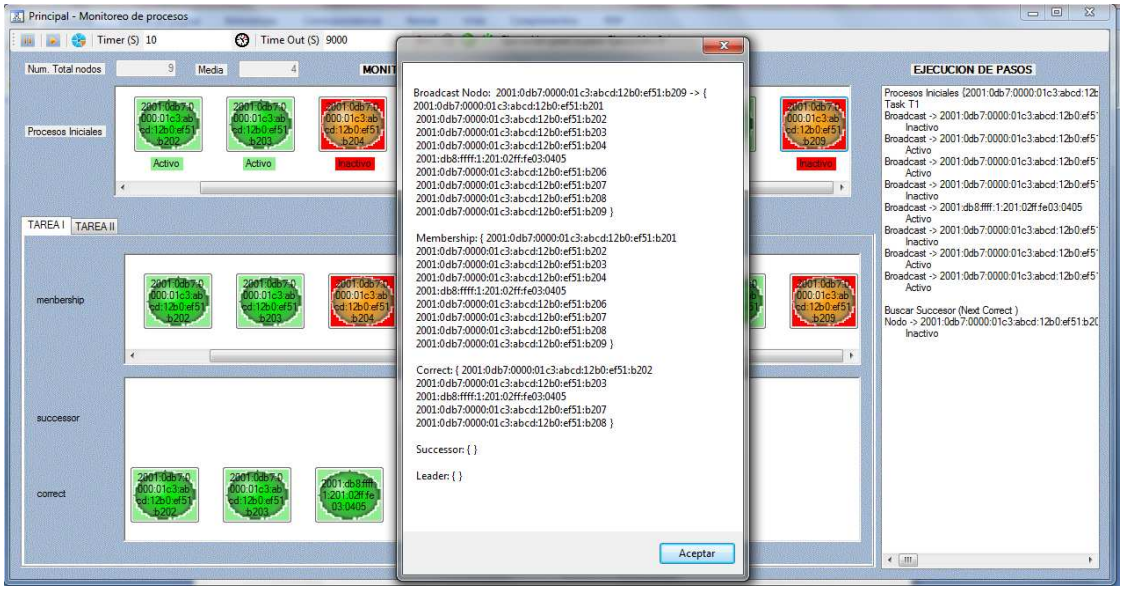


Figura 4.1.19 Ejecución N°9 Visualización de Resultados

En la figura 4.1.19 se pueden visualizar los vectores ***Membership***, ***Correct***, ***Sucessor*** *y* ***Leader*** con sus respectivos valores.

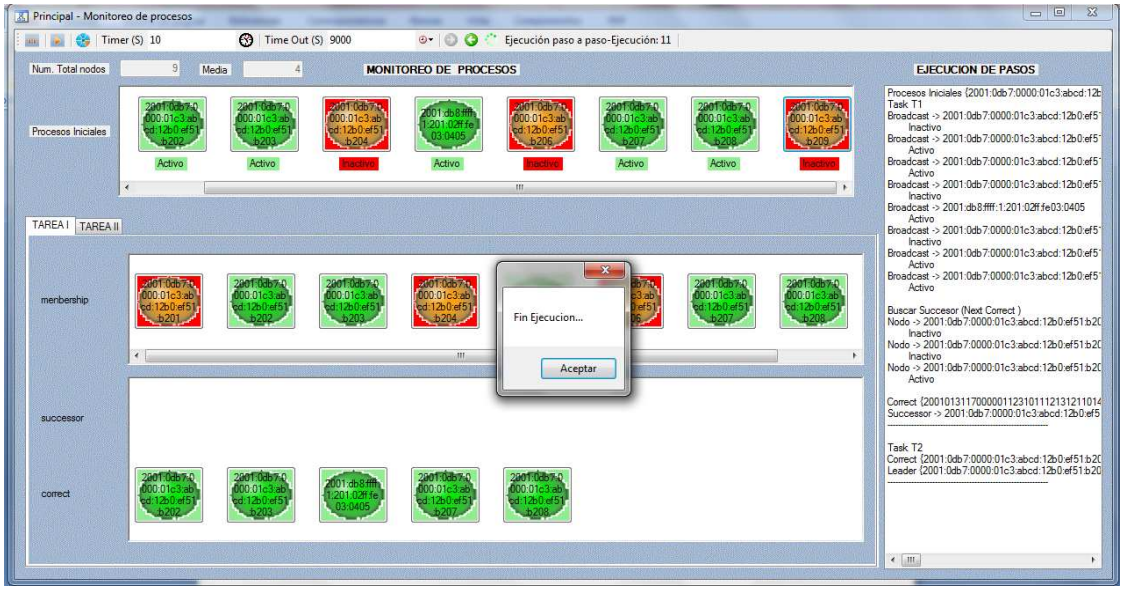


Figura 4.1.20 Ejecución N°11

Una vez finalizada la ejecución se visualizan los resultados en pantalla.

Figura 4.1.21 Ejecución N°11 Tarea 1

En la figura anterior el vector ***Membership*** está conformado por los siguientes nodos:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b201,

2001:0db7:0000:01c3:abcd:12b0:ef51:b202,

2001:0db7:0000:01c3:abcd:12b0:ef51:b203,

2001:0db7:0000:01c3:abcd:12b0:ef51:b204,

2001:db8:ffff:1:201:02ff:fe03:0405,

2001:0db7:0000:01c3:abcd:12b0:ef51:b206,

2001:0db7:0000:01c3:abcd:12b0:ef51:b207,

2001:0db7:0000:01c3:abcd:12b0:ef51:b208,

2001:0db7:0000:01c3:abcd:12b0:ef51:b209}

El vector ***Correct*** está formado por los nodos activos, en este caso:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b202,

2001:0db7:0000:01c3:abcd:12b0:ef51:b203,

2001:db8:ffff:1:201:02ff:fe03:0405,

2001:0db7:0000:01c3:abcd:12b0:ef51:b207,

2001:0db7:0000:01c3:abcd:12b0:ef51:b208}

Sucessor {2001:0db7:0000:01c3:abcd:12b0:ef51:b202}

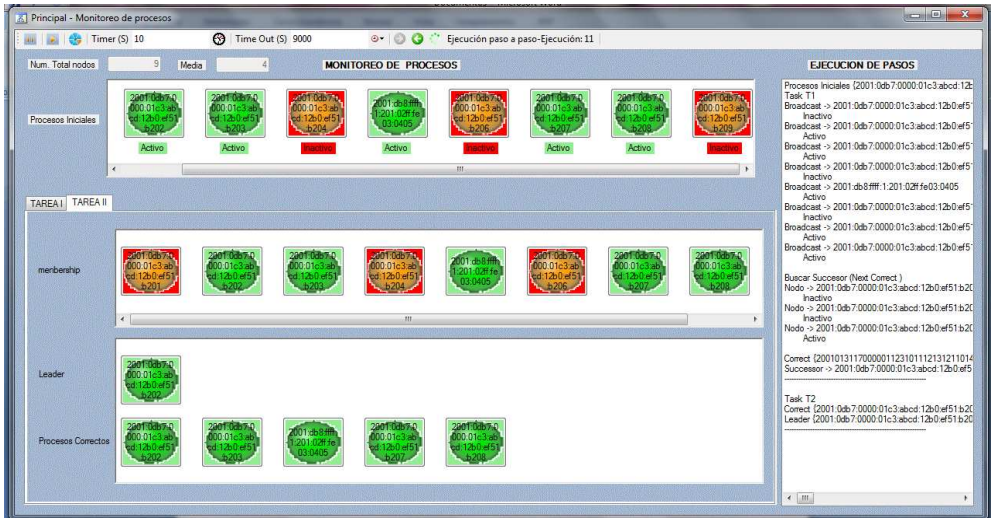


Figura 4.1.22 Ejecución N°11 Tarea 2

En la figura anterior el vector ***Membership*** está conformado por los siguientes nodos:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b201,

2001:0db7:0000:01c3:abcd:12b0:ef51:b202,

2001:0db7:0000:01c3:abcd:12b0:ef51:b203,

2001:0db7:0000:01c3:abcd:12b0:ef51:b204,

2001:db8:ffff:1:201:02ff:fe03:0405,

2001:0db7:0000:01c3:abcd:12b0:ef51:b206,

2001:0db7:0000:01c3:abcd:12b0:ef51:b207,

2001:0db7:0000:01c3:abcd:12b0:ef51:b208,

2001:0db7:0000:01c3:abcd:12b0:ef51:b209}

El vector ***Correct*** está formado por los nodos activos, en este caso:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b202,

2001:0db7:0000:01c3:abcd:12b0:ef51:b203,

2001:db8:ffff:1:201:02ff:fe03:0405,

2001:0db7:0000:01c3:abcd:12b0:ef51:b207,

2001:0db7:0000:01c3:abcd:12b0:ef51:b208}

Leader {2001:0db7:0000:01c3:abcd:12b0:ef51:b202}

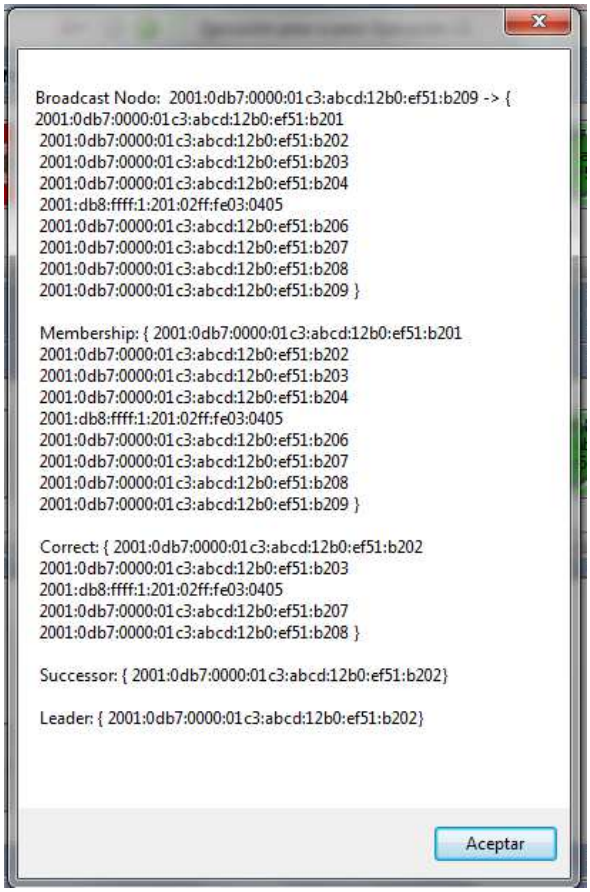


Figura 4.1.23 Resultados Finales

En la figura 4.1.23 se ve representados los resultados finales de cada uno de los vectores. Adicionalmente con esto se puede verificar que a nivel de la simulación está funcionando correctamente.

Figura 4.1.24 Prueba N°2 Desactivación de Nodo Líder

La implementación del Algoritmo Omega es un modelo crash & recovery (fallo - recuperación), esto significa que si un nodo falla será reemplazado por otro permitiendo de esta forma la recuperación del sistema. Para verificar su correcto funcionamiento en lo relacionado con esta consideración se procedió a desactivar el nodo líder en la interfaz como se puede observar en la figura 4.1.24.

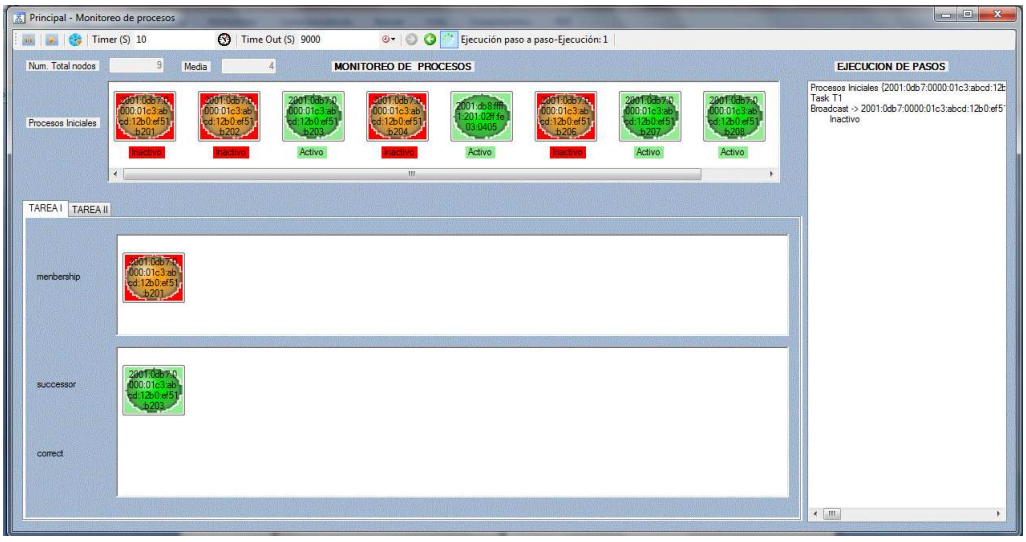


Figura 4.1.25 Ejecución N°1

En la figura anterior el vector ***Membership*** está conformado por el siguiente nodo:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b201}

El Vector ***Sucessor*** contiene el nodo siguiente de la secuencia cuya dirección IP es la

{2001:0db7:0000:01c3:abcd:12b0:ef51:b203}

Los vectores ***Correct*** *y* ***Leader*** no contienen ningún nodo.

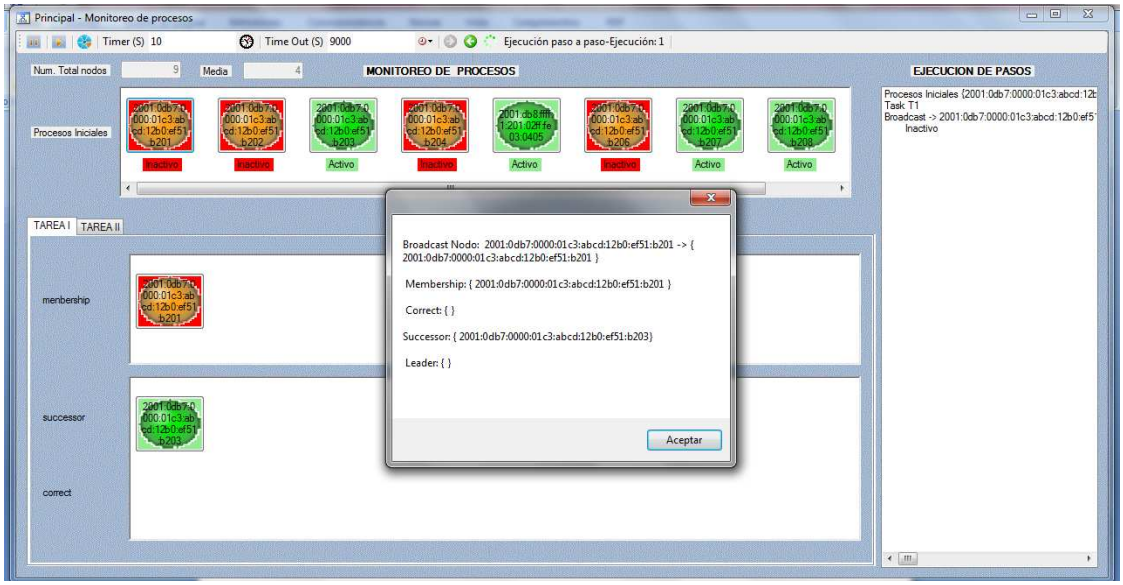


Figura 4.1.26 Ejecución N°1 Visualización de Resultados

En la figura 4.1.26 se pueden visualizar los vectores ***Membership***, ***Correct***, ***Sucessor*** *y* ***Leader*** con sus respectivos valores.

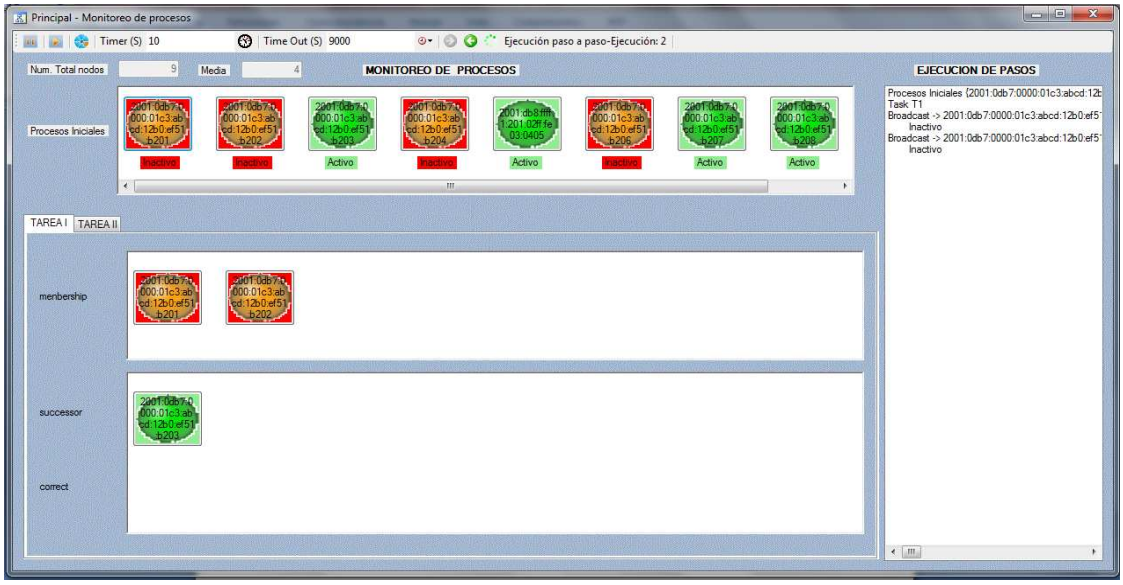


Figura 4.1.27 Ejecución N°2

En la figura anterior el vector ***Membership*** está conformado por los siguientes nodos:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b201,

2001:0db7:0000:01c3:abcd:12b0:ef51:b202}

El vector ***Sucessor*** contiene el nodo siguiente de la secuencia cuya dirección IP es la

{2001:0db7:0000:01c3:abcd:12b0:ef51:b203}

Los vectores ***Correct*** *y* ***Leader*** no contienen ningún nodo.

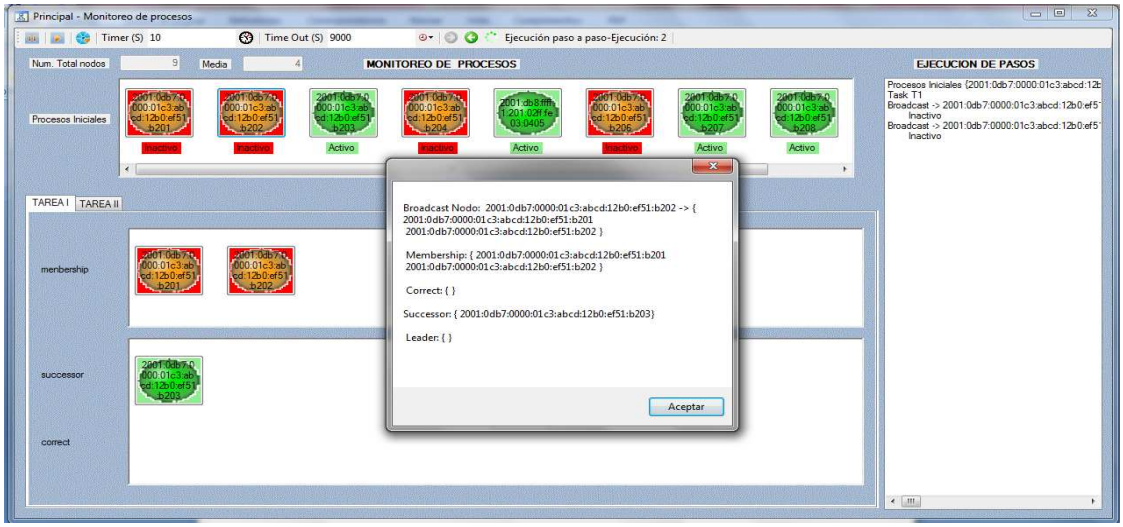


Figura 4.1.28 Ejecución N°2 Visualización de Resultados

En la figura 4.1.28 se pueden visualizar los vectores ***Membership***, ***Correct***, ***Sucessor*** *y* ***Leader*** con sus respectivos valores.

Figura 4.1.29 Ejecución N°3

En la figura anterior el vector ***Membership*** está conformado por los siguientes nodos:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b201,

2001:0db7:0000:01c3:abcd:12b0:ef51:b202,

2001:0db7:0000:01c3:abcd:12b0:ef51:b203}

El vector ***Sucessor*** contiene el nodo siguiente de la secuencia cuya dirección IP es la **{2001:0db7:0000:01c3:abcd:12b0:ef51:b203}**

El vector ***Correct*** contiene el nodo cuya dirección IP es:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b203}

El Vector ***Leader*** no contiene ningún nodo.

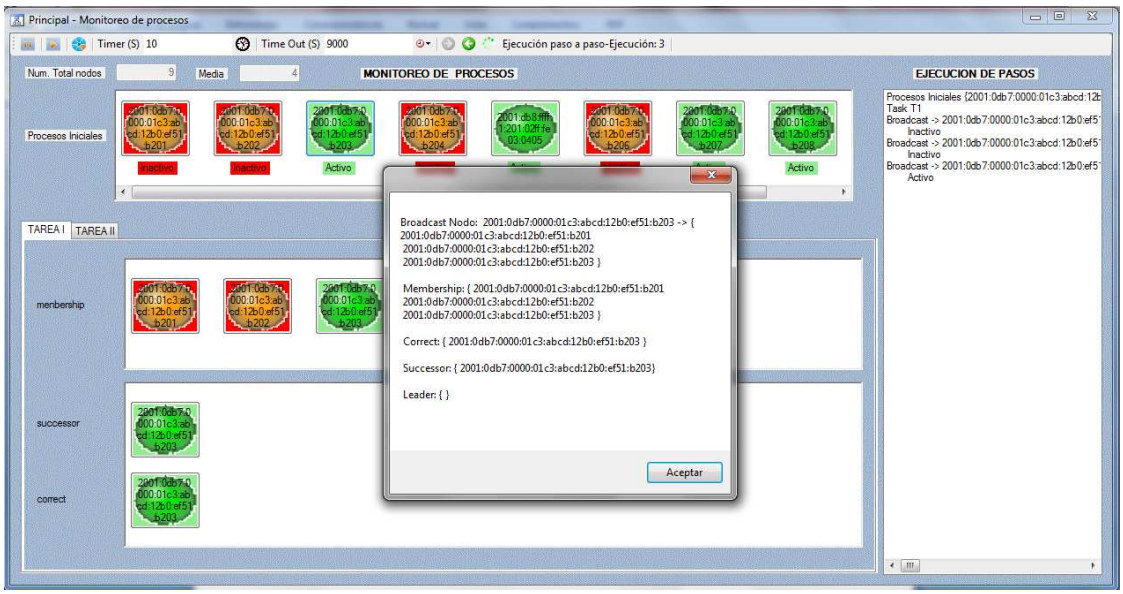


Figura 4.1.30 Ejecución N°3 Visualización de Resultados

En la figura 4.1.30 se pueden visualizar los vectores ***Membership***, ***Correct***, ***Sucessor*** *y* ***Leader*** con sus respectivos valores.

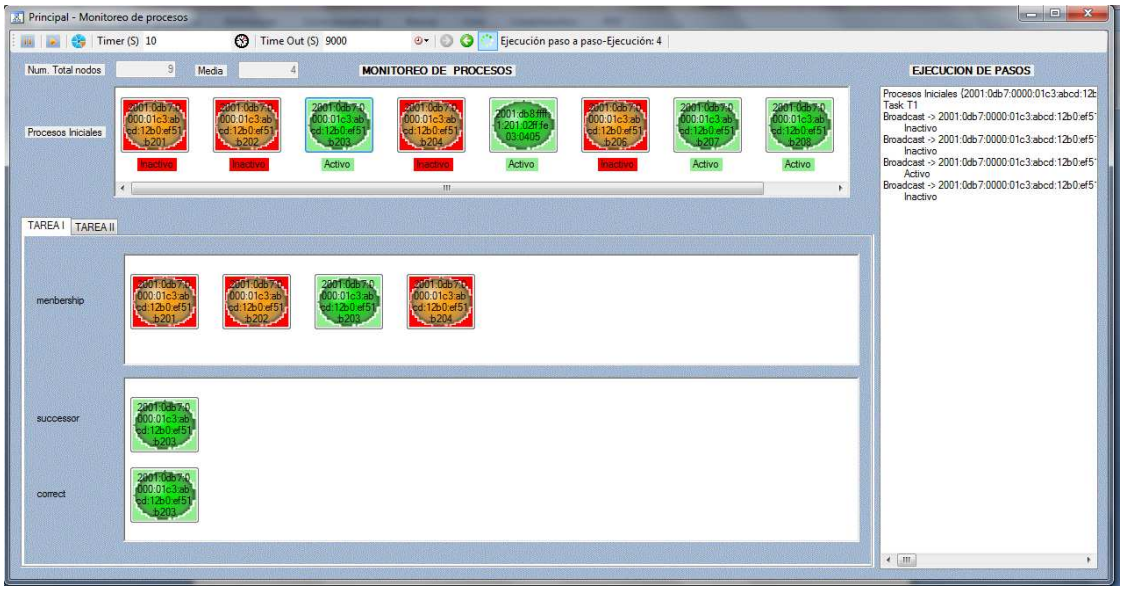


Figura 4.1.31 Ejecución N°4

En la figura anterior el vector ***Membership*** está conformado por los siguientes nodos:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b201,

2001:0db7:0000:01c3:abcd:12b0:ef51:b202,

2001:0db7:0000:01c3:abcd:12b0:ef51:b203,

2001:0db7:0000:01c3:abcd:12b0:ef51:b204}

El vector ***Sucessor*** contiene el nodo siguiente de la secuencia cuya dirección IP es la **{2001:0db7:0000:01c3:abcd:12b0:ef51:b203}**

El vector ***Correct*** contiene el nodo cuya dirección IP es la siguiente **{2001:0db7:0000:01c3:abcd:12b0:ef51:b203}**

El vector ***Leader*** no contiene ningún nodo.

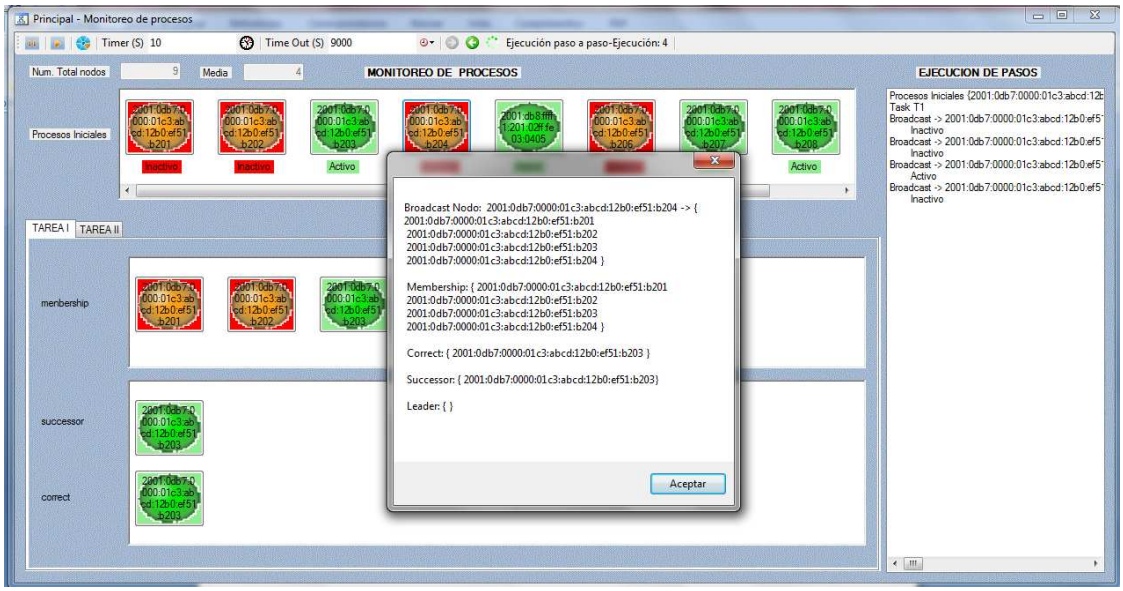


Figura 4.1.32 Ejecución N°4 Visualización de Resultados

En la figura 4.1.32 se pueden visualizar los vectores ***Membership***, ***Correct***, ***Sucessor*** *y* ***Leader*** con sus respectivos valores.

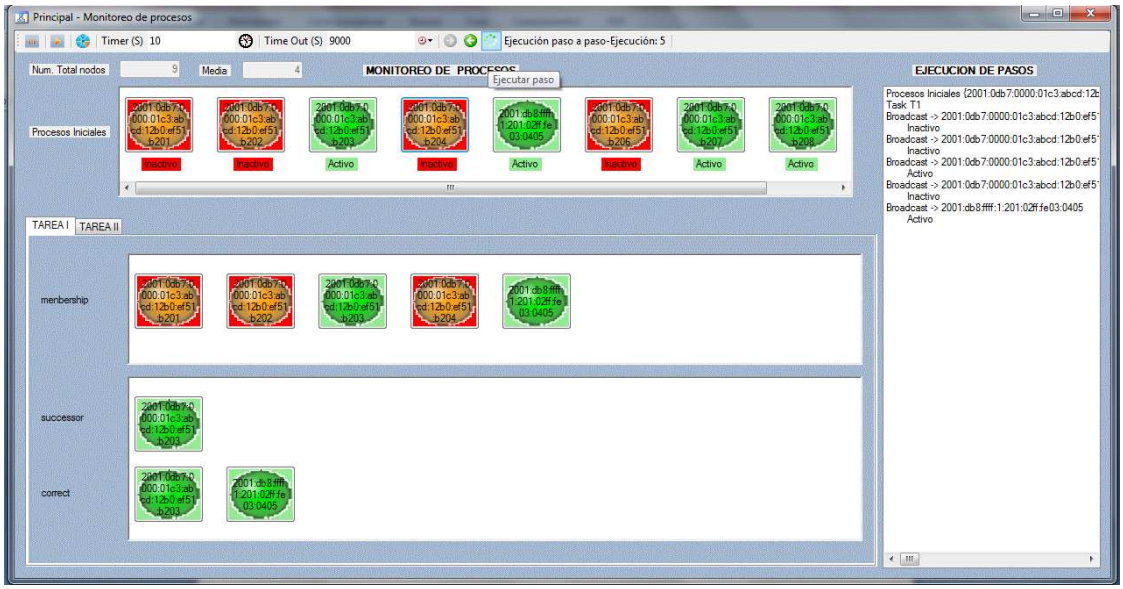


Figura 4.1.33 Ejecución N°5

En la figura anterior el vector ***Membership*** está conformado por los siguientes nodos:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b201,

2001:0db7:0000:01c3:abcd:12b0:ef51:b202,

2001:0db7:0000:01c3:abcd:12b0:ef51:b203,

2001:0db7:0000:01c3:abcd:12b0:ef51:b204,

2001:db8:ffff:1:201:02ff:fe03:0405}

El vector ***Sucessor*** contiene el nodo siguiente de la secuencia cuya dirección IP es la **{2001:0db7:0000:01c3:abcd:12b0:ef51:b203}**

El vector ***Correct*** contiene los siguientes nodos los cuales poseen las direcciones IP detalladas a continuación:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b203,

2001:db8:ffff:1:201:02ff:fe03:0405}

El vector ***Leader*** no contiene ningún nodo.

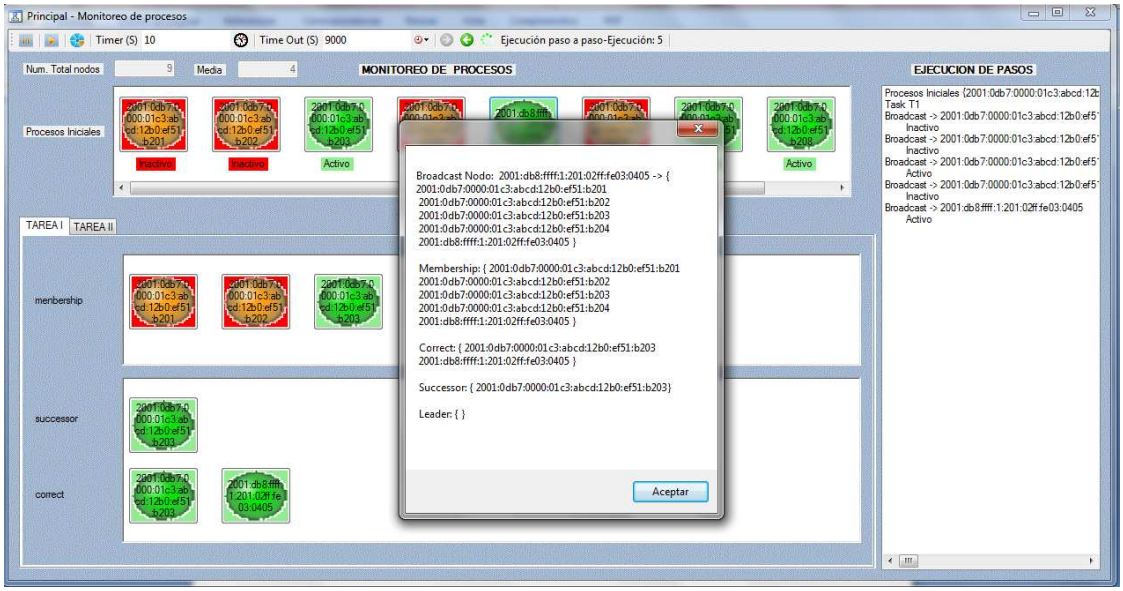


Figura 4.1.34 Ejecución N°5 Visualización de Resultados

En la figura 4.1.34 se pueden visualizar los vectores ***Membership***, ***Correct***, ***Sucessor*** *y* ***Leader*** con sus respectivos valores.

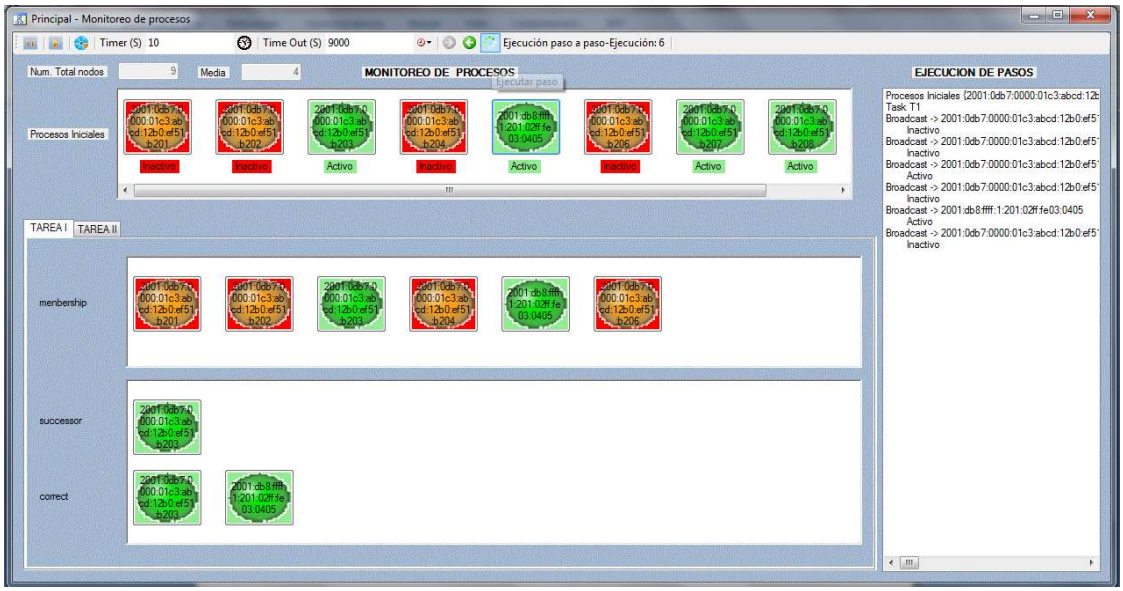


Figura 4.1.35 Ejecución N°6

En la figura anterior el vector ***Membership*** está conformado por los siguientes nodos:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b201,

2001:0db7:0000:01c3:abcd:12b0:ef51:b202,

2001:0db7:0000:01c3:abcd:12b0:ef51:b203,

2001:0db7:0000:01c3:abcd:12b0:ef51:b204,

2001:db8:ffff:1:201:02ff:fe03:0405,

2001:0db7:0000:01c3:abcd:12b0:ef51:b206}

El vector ***Sucessor*** contiene el nodo siguiente de la secuencia cuya dirección IP es la **{2001:0db7:0000:01c3:abcd:12b0:ef51:b203}**

El vector ***Correct*** contiene los nodos los cuales poseen las siguientes direcciones IP:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b203, 2001:db8:ffff:1:201:02ff:fe03:0405}

El vector ***Leader*** no contiene ningún nodo.

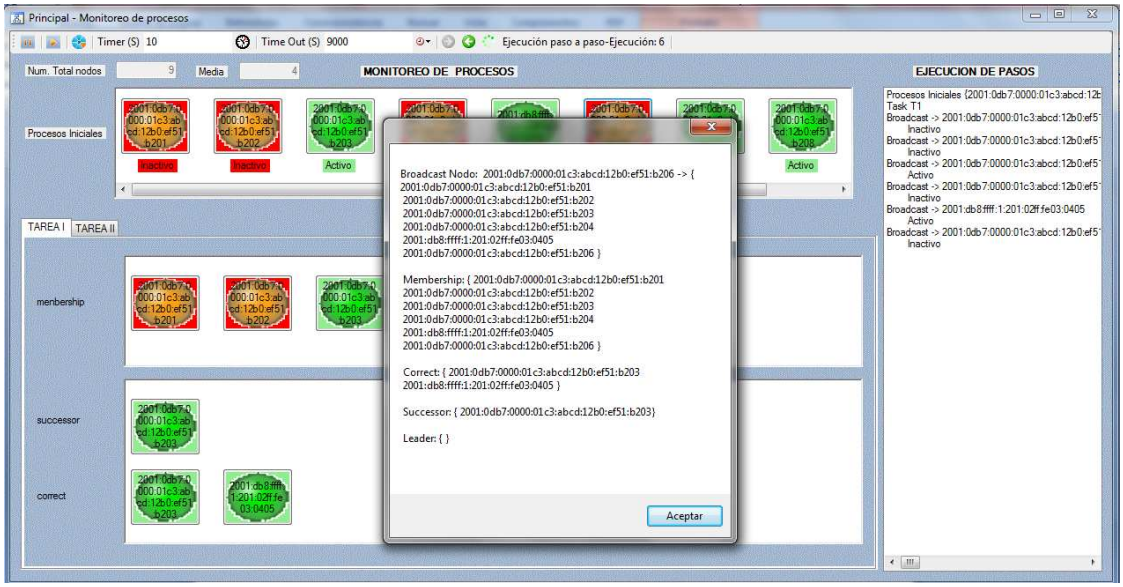


Figura 4.1.36 Ejecución N°6 Visualización de Resultados

En la figura 4.1.36 se pueden visualizar los vectores ***Membership***, ***Correct***, ***Sucessor*** *y* ***Leader*** con sus respectivos valores.

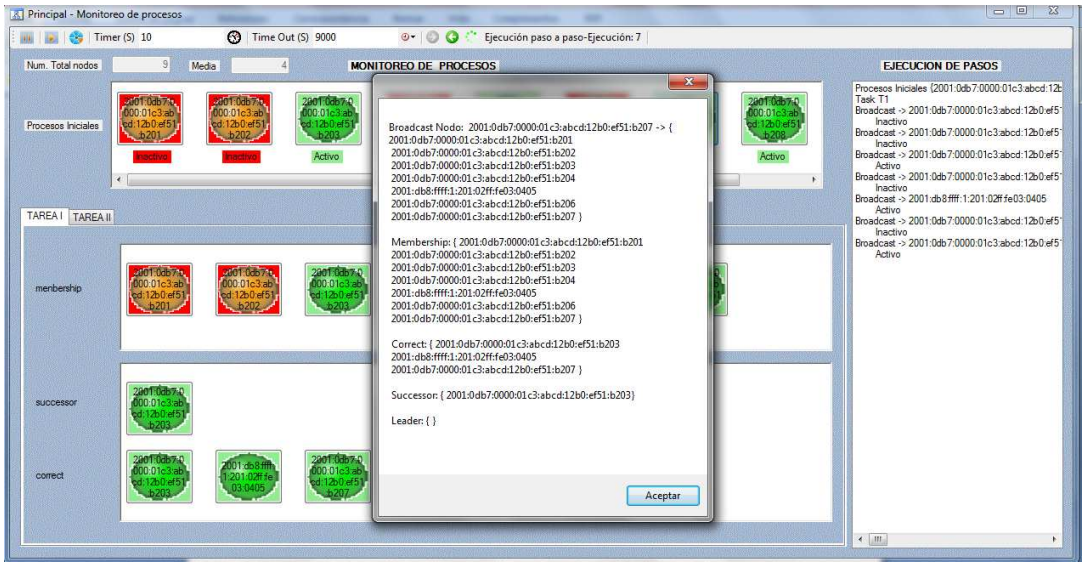


Figura 4.1.37 Ejecución N°7

En la figura anterior el vector ***Membership*** está conformado por los siguientes nodos:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b201,

2001:0db7:0000:01c3:abcd:12b0:ef51:b202,

2001:0db7:0000:01c3:abcd:12b0:ef51:b203,

2001:0db7:0000:01c3:abcd:12b0:ef51:b204,

2001:db8:ffff:1:201:02ff:fe03:0405,

2001:0db7:0000:01c3:abcd:12b0:ef51:b206,

2001:0db7:0000:01c3:abcd:12b0:ef51:b207}

El vector ***Sucessor*** contiene el nodo siguiente de la secuencia cuya dirección IP es la **{2001:0db7:0000:01c3:abcd:12b0:ef51:b203}**

El vector ***Correct*** contiene los nodos los cuales poseen las siguientes direcciones IP:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b203,

2001:db8:ffff:1:201:02ff:fe03:0405,

2001:0db7:0000:01c3:abcd:12b0:ef51:b207}

El vector ***Leader*** no contiene ningún nodo.

Figura 4.1.38 Ejecución N°8

En la figura anterior el vector ***Membership*** está conformado por los siguientes nodos:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b201,

2001:0db7:0000:01c3:abcd:12b0:ef51:b202,

2001:0db7:0000:01c3:abcd:12b0:ef51:b203,

2001:0db7:0000:01c3:abcd:12b0:ef51:b204,

2001:db8:ffff:1:201:02ff:fe03:0405,

2001:0db7:0000:01c3:abcd:12b0:ef51:b206,

2001:0db7:0000:01c3:abcd:12b0:ef51:b207,

2001:0db7:0000:01c3:abcd:12b0:ef51:b208}

El vector ***Sucessor*** contiene el nodo siguiente de la secuencia cuya dirección IP es la **{2001:0db7:0000:01c3:abcd:12b0:ef51:b203}**

El vector ***Correct*** contiene los nodos los cuales poseen las siguientes direcciones IP:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b203,

2001:db8:ffff:1:201:02ff:fe03:0405,

2001:0db7:0000:01c3:abcd:12b0:ef51:b207,

2001:0db7:0000:01c3:abcd:12b0:ef51:b208}

El vector ***Leader*** no contiene ningún nodo.

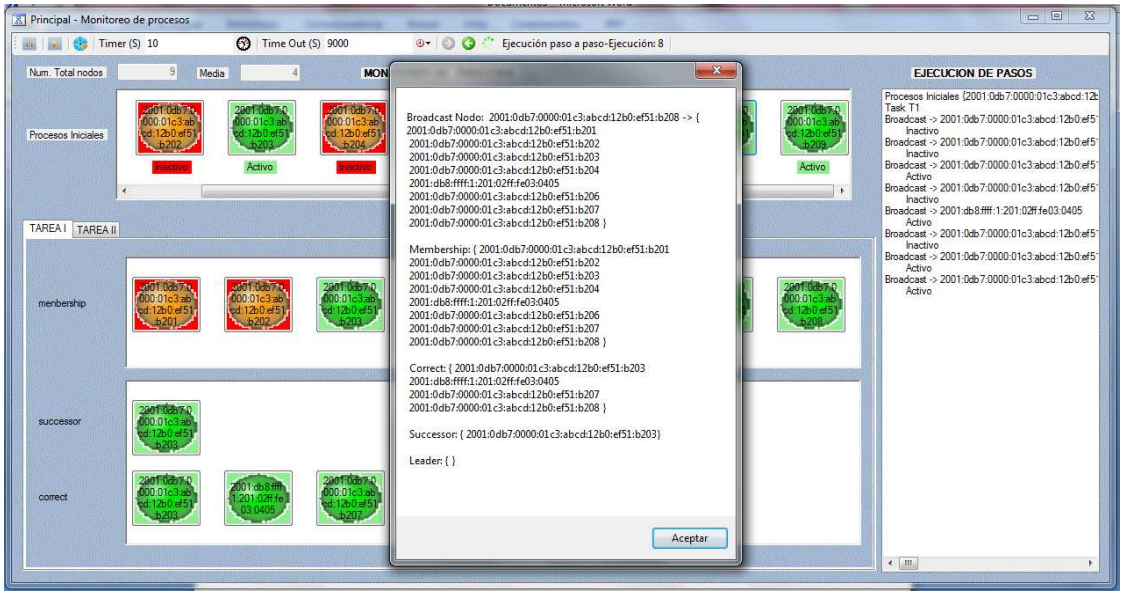


Figura 4.1.39 Ejecución N°8 Visualización de Resultados

En la figura 4.1.39 se pueden visualizar los vectores ***Membership***, ***Correct***, ***Sucessor*** *y* ***Leader*** con sus respectivos valores.

Figura 4.1.40 Ejecución N°9

El vector ***Membership*** está conformado por los siguientes nodos:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b201,

2001:0db7:0000:01c3:abcd:12b0:ef51:b202,

2001:0db7:0000:01c3:abcd:12b0:ef51:b203,

2001:0db7:0000:01c3:abcd:12b0:ef51:b204,

2001:db8:ffff:1:201:02ff:fe03:0405,

2001:0db7:0000:01c3:abcd:12b0:ef51:b206,

2001:0db7:0000:01c3:abcd:12b0:ef51:b207,

2001:0db7:0000:01c3:abcd:12b0:ef51:b208,

2001:0db7:0000:01c3:abcd:12b0:ef51:b209}

El vector ***Sucessor*** contiene el nodo siguiente de la secuencia cuya dirección IP es la {2001:0db7:0000:01c3:abcd:12b0:ef51:b203}

El vector ***Correct*** contiene los nodos los cuales poseen las siguientes direcciones IP:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b203,

2001:db8:ffff:1:201:02ff:fe03:0405,

2001:0db7:0000:01c3:abcd:12b0:ef51:b207,

2001:0db7:0000:01c3:abcd:12b0:ef51:b208,

2001:0db7:0000:01c3:abcd:12b0:ef51:b209}

El vector ***Leader*** no contiene ningún nodo.

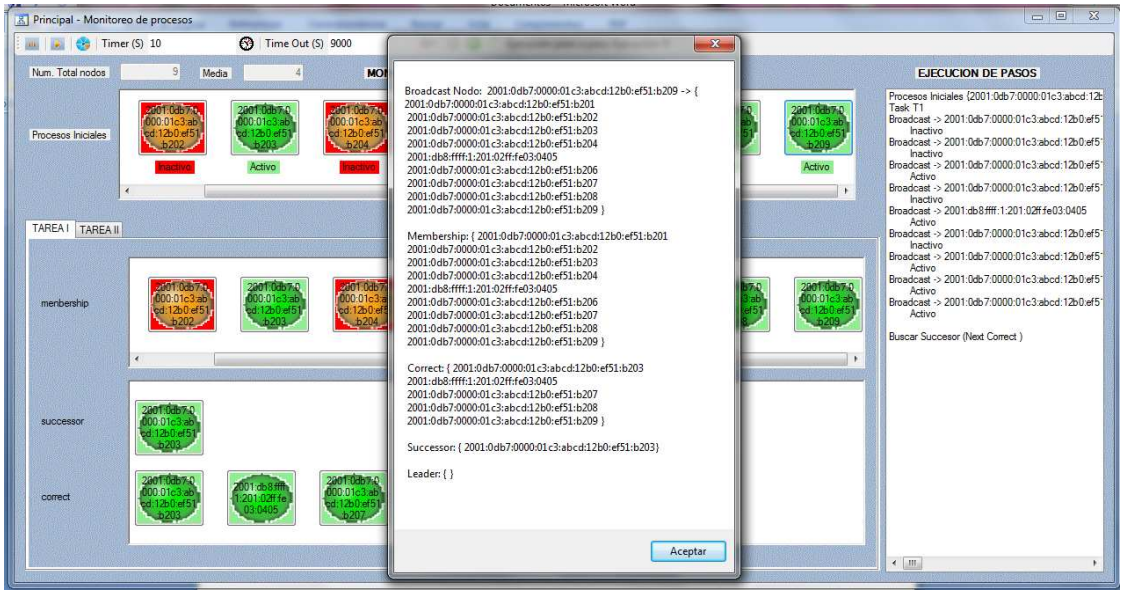


Figura 4.1.41 Ejecución N°9 Visualización de Resultados

En la figura 4.1.42 se pueden visualizar los vectores ***Membership***, ***Correct***, ***Sucessor*** *y* ***Leader*** con sus respectivos valores.

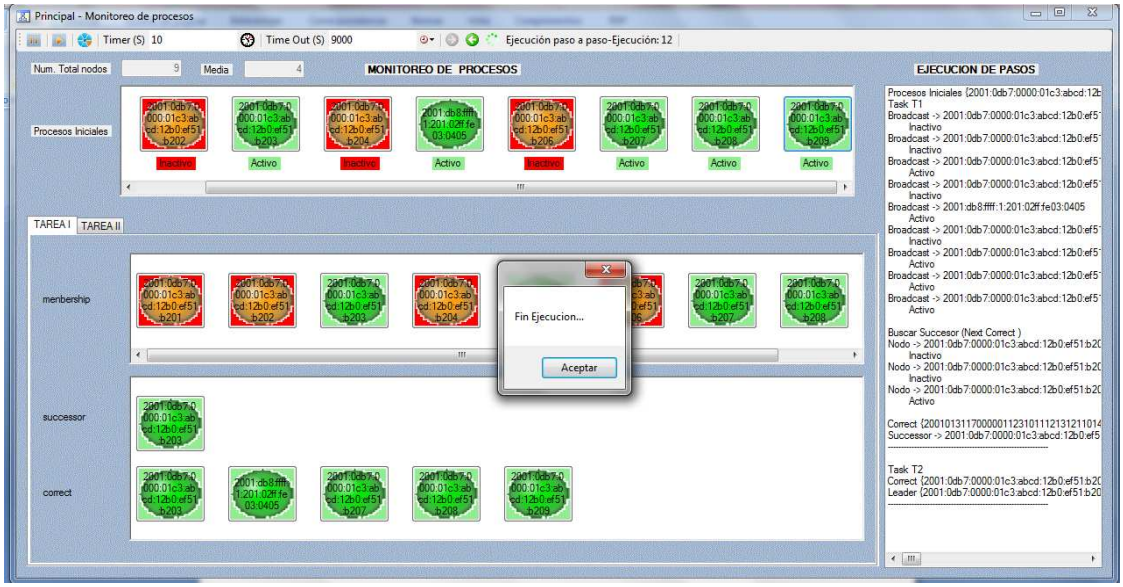


Figura 4.1.42 Fin de la Ejecución Tarea 1

En la figura anterior se visualizan los resultados de la Tarea 1 con los respectivos valores de sus vectores.

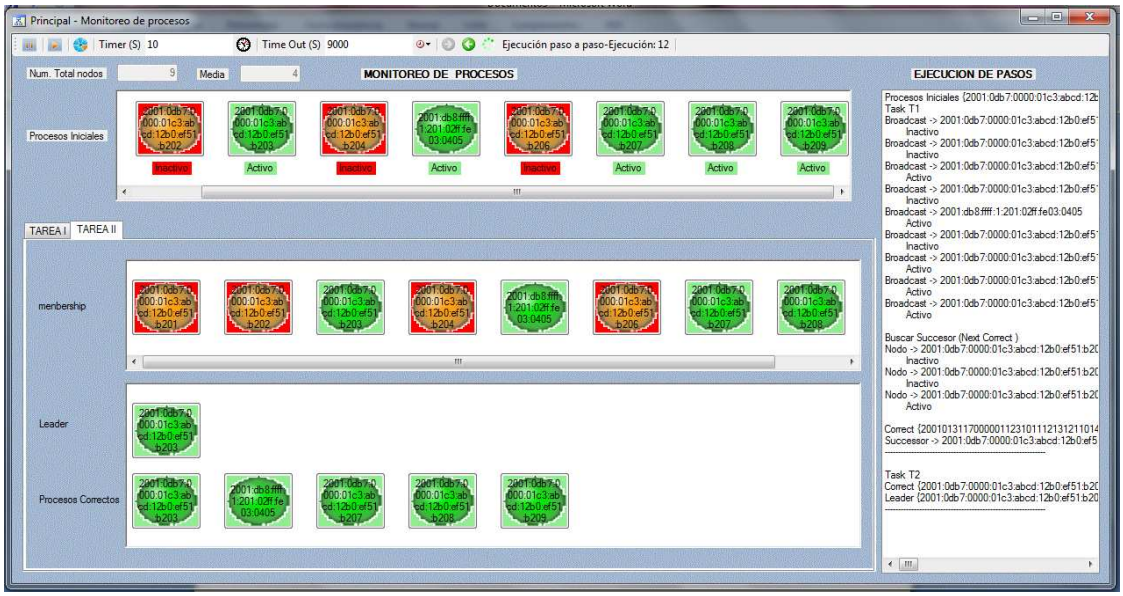


Figura 4.1.43 Fin de la Ejecución Tarea 2

En la figura anterior se visualizan los resultados de la Tarea 2 con los respectivos valores de sus vectores.

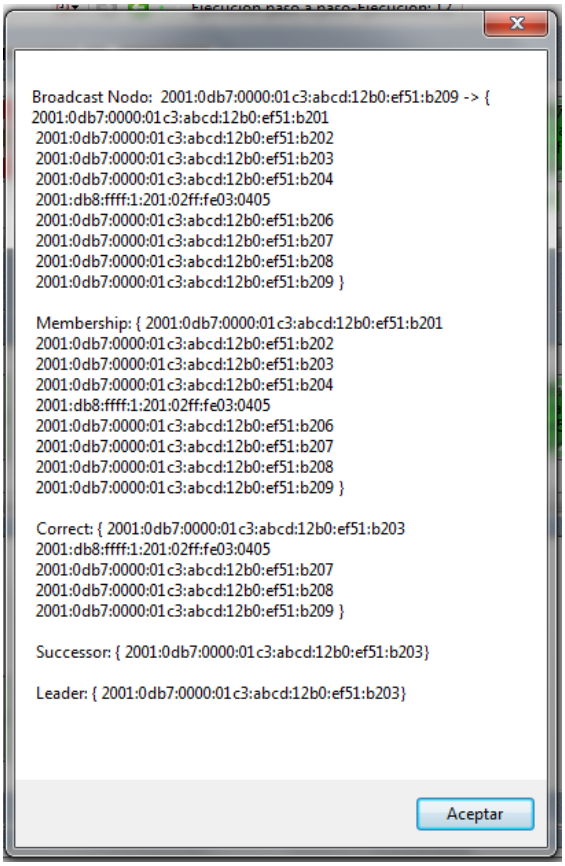


Figura 4.1.44 Resultados Finales

En la figura 4.1.44 se visualizan los resultados finales de la ejecución del simulador:

El vector ***Membership*** está conformado por los siguientes nodos:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b201,

2001:0db7:0000:01c3:abcd:12b0:ef51:b202,

2001:0db7:0000:01c3:abcd:12b0:ef51:b203,

2001:0db7:0000:01c3:abcd:12b0:ef51:b204,

2001:db8:ffff:1:201:02ff:fe03:0405,

2001:0db7:0000:01c3:abcd:12b0:ef51:b206,

2001:0db7:0000:01c3:abcd:12b0:ef51:b207,

2001:0db7:0000:01c3:abcd:12b0:ef51:b208,

2001:0db7:0000:01c3:abcd:12b0:ef51:b209}

El vector ***Sucessor*** contiene el nodo siguiente de la secuencia cuya dirección IP es la **{2001:0db7:0000:01c3:abcd:12b0:ef51:b203}**

El vector ***Correct*** contiene los nodos los cuales poseen las siguientes direcciones IP:

{2001:0db7:0000:01c3:abcd:12b0:ef51:b203,

2001:db8:ffff:1:201:02ff:fe03:0405,

2001:0db7:0000:01c3:abcd:12b0:ef51:b207,

2001:0db7:0000:01c3:abcd:12b0:ef51:b208,

2001:0db7:0000:01c3:abcd:12b0:ef51:b209}

El vector ***Leader*** contiene el nodo cuya dirección IP es la: **{2001:0db7:0000:01c3:abcd:12b0:ef51:b203}**

El vector ***Leader*** contiene el menor valor de los nodos del vector ***Correct***.

Una vez concluidas las pruebas se puede llegar a determinar que la implementación del algoritmo Omega se ha ejecutado en forma satisfactoria por cuanto se puede visualizar que existe una correcta recuperación del sistema, al momento de existir algún fallo en el nodo designado como líder nuevamente se vuelve a ejecutar la

lógica del algoritmo para la selección de un nuevo nodo líder logrando de esta forma una estabilidad y un óptimo rendimiento de la red.

4.2 Conclusiones

- Al finalizar el Proyecto de Titulación se realizó el análisis, diseño y simulación del Algoritmo Omega para redes de sensores inalámbricas basado en un modelo **crash & recovery (fallo - recuperación),** es decir que si un nodo falla será reemplazado por otro permitiendo de esta forma la recuperación del sistema, esto es muy importante por cuanto permite una alta disponibilidad del mismo.

- Una vez desarrollado el simulador se determinó el ambiente de pruebas con la finalidad de probar el correcto funcionamiento del algoritmo detector de fallos Omega, los resultados de las pruebas fueron satisfactorios y permitieron obtener como conclusión que la adaptación realizada a los procesos con los nodos fue compatible y su comportamiento fue el esperado.

- El alcance inicial del proyecto contemplaba la implementación del algoritmo Omega con las funciones adaptadas a IPv6, pero durante la fase de desarrollo del mismo se presentó el inconveniente que no se encuentran disponibles entornos de depuración de código para los microcontroladores utilizados en las motas, lo cual dificulta la fase de localización y corrección de errores, que se debe realizar a partir de los LEDs disponibles en las mismas. De igual manera el aprendizaje y la familiarización con el lenguaje para la programación de los nodos se

volvió compleja, por tal motivo se redujo el alcance y se realizó las respectivas adaptaciones al algoritmo Omega mediante el desarrollo de un simulador regido por ciertos parámetros configurables para recrear su funcionamiento, tomando en cuenta que una de sus principales ventajas radica en que es independiente de la topología de la red lo cual reduce tiempo y optimiza recursos en comparación con otros algoritmos de enrutamiento.

- Es conveniente aclarar que el objetivo de este Proyecto de Titulación es efectuar las acciones que permitan adaptar el algoritmo OMEGA en una red de sensores inalámbricos, permitiendo dar una confiabilidad a este tipo de redes. Como se observó, el poder simular este tipo de redes de acuerdo a las necesidades del mundo actual, permitirá brindar soluciones tecnológicas en un futuro no muy lejano.

4.3 Recomendaciones

- Es evidente que el futuro de las redes está relacionado cada vez más al software, el cual permite acelerar el ritmo de innovación de las mismas, por ello se considera importante fomentar el desarrollo de aplicaciones orientadas a las Redes WSN para transformar las redes estáticas actuales.

- Se alienta a los desarrolladores a incursionar en el amplio y apasionante mundo de las redes mediante la elaboración de nuevas aplicaciones que incluyan algoritmos tolerantes a fallos, este es un factor muy fundamental dentro de una red por cuanto cuando existe algún fallo dentro de alguno de los nodos que conforman la red WSN la misma no se verá impactada de forma significativa. La presente investigación deja un antecedente para aquellas personas que están estudiando las redes de sensores inalámbricas orientadas a su recuperación y disponibilidad.

- La estructuración del software es una parte esencial al momento de diseñar un simulador, por lo tanto se recomienda analizar los requerimientos y limitantes de hardware y software existentes para el desarrollo de la aplicación, además del lenguaje de programación que se vaya a utilizar.

- Debido al tiempo y a la complejidad del modelo, se obtuvo un simulador de complejidad intermedia pero muy interesante, motivo por lo cual para mejorarlo se recomienda incluir otras variables dentro su parametrización.

4.4 Bibliografía

[1] "MIT Technology Review" 10 Emerging Technologies That Will Change the World (10 Tecnologias Emergentes que pueden cambiar el mundo), Febrero 2003

http://www2.technologyreview.com/news/401775/10-emerging-technologies-that-will-change-the/2/

[2] Detector de Fallas Omega

http://edgar-guerra.tripod.com/data-base-2007/id26.html

[3] "Redes inalámbricas de sensores: Una nueva arquitectura eficiente y robusta basada en jerarquía dinámica de grupos", CAPELA JUAN, Abril 2010
https://riunet.upv.es/bitstream/handle/10251/8417/tesisUPV3326.pdf

[4] "Protocolos de Enrutamiento Para la Capa de Red en

Arquitecturas de Redes de Datos", MARQUEZ GOMEZ ANDRES

http://www.monografias.com/trabajos-pdf/enrutamiento-redes-datos/enrutamiento-redes-datos.pdf

[5] "Wireless Sensor Networks Estado del Arte e Investigación", M.SOLEDAD ESCOBAR DÍAZ

http://www.arcos.inf.uc3m.es/~sescolar/index_files/presentacion/wsn.pdf

[6] T.D. Chandra, S. Toueg, Unreliable failure detectors for reliable distributed systems, Journal of the ACM 43 (2) (March 1996) 225-267

[7] TOLERANCIA A FALLOS Y AUTOCONFIGURACION EN REDES DE SENSORES, PABLO ALBIZU BALERDI, Septiembre del 2010

http://academica-e.unavarra.es/bitstream/handle/2454/2105/577206.pdf?sequence=1

[8] ELECCIÓN FUTURA DE LÍDER EN SISTEMAS DISTRIBUIDOS NO FIABLES, ANTONIO FERNANDEZ

http://www.dim.uchile.cl/~redes/escuela07/eleccion-futura-lider-v3.pdf

[9] SENSORES, J. RAMIRO MARTÍNEZ DE DIOS

http://www.esi2.us.es/~jdedios/asignaturas/Sensores.pdf

[10] SIMULACIÓN DE UN ALGORITMO DE ENRUTAMIENTO PARA REDES DE SENSORES INALÁMBRICOS, MAYTHÉ GONZALEZ GUTIERREZ, FEBRERO 2012

http://digeset.ucol.mx/tesis_posgrado/Pdf/Maythe_Gonzalez_Gutierrez.pdf

[11] ALGORITMOS DE ENRUTAMIENTO POR INUNDACION PARA REDES DE SENSORES INALÁMBRICOS, ANTONIO GARCIA ABURTO

http://cdigital.uv.mx/bitstream/123456789/32048/1/garciaaburtoantonio.pdf

[12] DESARROLLO Y PRUEBAS DE DESEMPEÑO DE UN ALGORITMO DE ENRUTAMIENTO DINÁMICO BAJO EL PROTOCOLO IEEE 802.15.4, LOURDES K GAMARRA, SEPTIEMBRE 2008

http://biblioteca2.ucab.edu.ve/anexos/biblioteca/marc/texto/AAR4095.pdf

[13] INTRODUCCIÓN A LAS REDES DE SENSORES INALÁMBRICAS

http://www.mfbarcell.es/conferencias/wsn.pdf

[14] REDES DE SENSORES INALÁMBRICOS, FRANCISCO ORTIZ TAPIA

http://profesores.elo.utfsm.cl/~tarredondo/info/networks/Presentacion_sensores.pdf

[15] ADMINISTRACIÓN DE LA ENERGÍA

http://catarina.udlap.mx/u_dl_a/tales/documentos/lem/tapia_z_jl/capitulo9.pdf

[16] C. Martín Hernández, "The omega failure in the crash-recovery model," Universidad del País Vasco, 2014.

[17] S. Arévalo, E. Jiménez, M. Larrea, and L. Mengual, "Communication efficient and crash-quiescent Omega with unknown membership," *Inf. Process. Lett.*, vol. 111, no. 4, pp. 194–199, 2011.

[18] P. Albizu Balerdi and others, "Tolerancia a fallos y autoconfiguración en redes de sensores," *Syst. Eng. Procedia*, 2010.

[19] C. Martín and M. Larrea, "A simple and communication-efficient Omega algorithm in the crash-recovery model," *Inf. Process. Lett.*, vol. 110, no. 3, pp. 83–87, 2010.

[20] E. Jiménez, S. Arévalo, and A. Fernández, "Implementing unreliable failure detectors with unknown membership," *Inf. Process. Lett.*, vol. 100, no. 2, pp. 60– 63, 2006.

[21] M. Larrea, S. Arévalo, and A. Fernandez, "Efficient algorithms to implement unreliable failure detectors in partially synchronous systems," Springer, 1999, pp. 34–49.

[22] REDES DE SENSORES INALÁMBRICAS WSN, JOSÉ G. MONTES ARRÁIZ, JULIO 2012

https://redesdesensoreswsn.blogspot.com/2012/07/redes-de-sensores-inalambricas-wsn-upt.html

[23] INGENIERÍA DE SOFTWARE CUARTA EDICIÓN PAG: 24-28, BRAUDE

http://metodologiaencascada.blogspot.com/

[24] Diferencia entre IP pública e IP privada, Ángel Gutiérrez, Enero 2016-09-17

http://windowsespanol.about.com/od/RedesYDispositivos/f/IP-Publica-IP-Privada.htm

[25] Diferencias entre TCP/IPv4 y TCP/IPv6, Josito, Octubre 2007

http://www.configurarequipos.com/doc693.html

[26] ¿Qué ha pasado con IPv6?, Ramón Jesús Millán, 2004

http://www.ramonmillan.com/tutoriales/estadoipv6.php