Casa abierta al tiempo
UNIVERSIDAD AUTÓNOMA METROPOLITANA

Estudio de la dinámica estructural de las redes complejas usando simulación basada en agentes

Idónea Comunicación de Resultados para obtener el grado de

Maestra en Ciencias y Tecnologías de la Información

Presentado por:
Lic. Magali Alexander López Chavira

Asesor: Dr. Ricardo Marcelín Jiménez

Jurado Calificador:

Presidente: Dr. Víctor Manuel Landassuri Moreno

Secretario: Dr. Ricardo Marcelín Jiménez

Vocal: Dr. Michael Pascoe Chalke

México, D.F. a 17 de Octubre del 2014

Resumen

Los sistemas complejos en general, y las redes complejas en particular, son el resultado de acciones o decisiones individuales, y locales, tomadas por los agentes que componen a estos sistemas en donde, como resultado de la sinergia entre sus agentes, surge un comportamiento o una estructura emergente. Se sabe que la construcción de la Internet y la WWW (por las siglas en inglés de World Wide Web) obedecen a estas premisas y exhiben la estructura de las redes complejas. Se sabe también que esta estructura les dota de características especiales, como su diámetro, su tolerancia a fallas y también, su fragilidad ante ataques. Por otro lado, sabemos que la construcción de una red compleja puede ser el resultado de un proceso dinámico que le da forma con el paso del tiempo. Existen algunos experimentos que proponen modelar este proceso como un grafo en el que cada vértice es capaz de recablear sus enlaces a su conveniencia. En el presente trabajo se propone una plataforma de experimentación, basada en un simulador de eventos discretos, en la que se pueden estudiar las diferentes "fuerzas" que le dan forma a los cambios por los que puede atravesar una red. El objetivo es reconocer aquellos parámetros locales de los que pueden emerger propiedades globales interesantes, como las que pueden encontrarse en las redes complejas.

Agradecimientos

Agradezco a mi asesor el Dr. Ricardo Marcelín Jiménez, por aceptarme en este proyecto de investigación, por su orientación, apoyo constante y consejos. Extiendo el agradecimiento a la Universidad Autónoma Metropolitana por haberme dado la oportunidad de ingresar al posgrado. Así mismo agradezco al CONACYT por brindarme la beca que me permitió cumplir este gran paso en mi vida.

Agradezco a mi familia por estar a mi lado apoyándome en cada momento importante de mi vida y por su comprensión. En especial a mi esposo, por impulsarme en cada momento para terminar mi proyecto, por todo su apoyo y por aguantar mis largas noches de desvelo.

Contenido

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

Un sistema complejo está conformado por un conjunto de elementos individuales que actúan usando sólo reglas de comportamiento locales y la información que observan en su entorno. Los componentes del sistema pueden ser, por ejemplo, personas, neuronas, páginas web o computadoras. Analizando la interacción entre estos individuos, o componentes del sistema, se encuentran comportamientos y estructuras globales que son llamados *emergentes*. Estas propiedades emergentes, aunque son consecuencia de la interacción entre los individuos siguiendo sus reglas, no suelen ser consecuencia directa de la "suma"de las reglas locales. Las acciones emergentes nos brindan información adicional del sistema que las forma. Algunas de las estructuras formadas por los vínculos existentes entre los individuos del sistema muestran las propiedades que de manera genérica se describen como *redes complejas*.

El estudio de las redes complejas se ha incrementado en las últimas décadas gracias a que se puede ver que emergen en diversos sistemas de forma natural dentro de áreas de conocimiento como la física, biología y ciencias sociales, por mencionar algunas [1]. En el área de Tecnologías de la información se han utilizado para describir la estructura de la Internet, una red de computadoras interconectadas y de la propia WWW (del inglés World Wide Web), una red de ligas web.

Estudios recientes se centran, por un lado, en el comportamiento de distintos fenómenos sociales o biológicos sobre las redes complejas, por ejemplo, la velocidad con que se propaga una enfermedad en una población. Por otro lado, se analizan las propiedades estructurales, como el coeficiente de agrupamiento o el diámetro, lo cual permite conocer las ventajas y desventajas que tienen diversas estructuras.

Algunas de las características funcionales de los sistemas están basadas en sus propiedades estructurales, un ejemplo de ello es la distancia máxima entre cualesquiera dos nodos que suele ser pequeña. Esto puede beneficiar a ciertos sistemas de comunicación donde se requiera enviar un paquete con un mínimo de saltos, o realizar algún tipo de actualización en componentes conectados con una topología de red compleja. Un ejemplo en el área de sistemas paralelos y distribuidos se describe en [2], en donde se presenta una red par a par (P2P) dinámica. Una red de este tipo contiene Tablas Hash Distribuidas que contienen información actualizada del estado de la red. Cada vez que ocurre un cambio en la estructura de la red, ya sea por la adición o eliminación de un equipo se deben actualizar las tablas, dichas actualizaciones generan costos grandes y limitan la escalabilidad en la red. La solución propuesta es conectar los equipos de la red para que formen una estructura de red de mundos pequeños, que es un tipo de red compleja. Como resultado se tuvo una red que necesitaba pocos enlaces hacia otros nodos de la red, y un costo bajo en la actualización de las tablas ya que se aprovecha la propiedad de la red de tener distancias muy cortas entre cada nodo. En conclusión se logra que una red de este tipo presente un mayor grado de escalabilidad.

Recientemente se ha incrementado el uso de redes complejas en el estudio y creación de redes inalámbricas de sensores [3]. Para optimizar las características de estas redes de sensores se propuso incrementar la potencia de algunos sensores y calcular la mejor distribución de estos de manera que se tenga una topología de red de mundos pequeños. De este modo se reducen el número de saltos que separan a todos los componentes, lo cual puede incrementar la velocidad con que se entrega la información al destino final y aumentar la eficiencia de la red.

Sabemos que la construcción de una red compleja puede ser el resultado de un proceso dinámico que le da forma con el paso del tiempo. Existen algunos experimentos que proponen modelar este proceso como un grafo en el que cada vértice es capaz de recablear sus enlaces a su conveniencia. Al observar la estructura global podemos ver que acciones locales se deben tomar para obtener diversas mejoras en la red.

Las necesidades más importantes en el estudio de este tipo especial de estructuras son: manejo de grafos con cantidades masivas de nodos, herramientas de creación de redes, herramientas de visualización y una sencilla simulación de eventos en la red, por lo que recurrimos a la simulación basada en agentes.

El uso de simulaciones es un medio de investigación que ha sido usado con mayor frecuencia en la actualidad ya que sólo necesitamos definir qué reglas locales deben seguir los componentes del sistema. En el caso de una implementación real puede ser muy costosa la realización de un experimento. En el caso de las simulaciones existen diversas herramientas que logran abarcar todos los parámetros necesarios para representar un sistema real, permitiendo una buena observación del experimento.

Para desarrollar una simulación basada en agentes existen diversas herramientas, en este trabajo se eligieron aquellas de software libre. Una de las herramientas más usadas es NetLogo [4], que permite la programación de las reglas a seguir por los agentes dentro de un entorno sencillo y la visualización del sistema, además permite monitorizar las interacciones entre agentes. A pesar de ser una herramienta muy útil, se descartó su uso debido a que la herramienta nos muestra la animación del sistema y esta visualización no puede ser omitida. Analizando la gran cantidad de vértices que se quieren modelar y la aún mayor cantidad de mensajes, los primeros experimentos con esta herramienta fueron muy tardados y para un gran número de vértices no se pudieron visualizar todos los elementos y estructuras formadas.

Lo que se busca de la herramienta a utilizar es que pueda manejar cantidades muy grandes de vértices. Dentro de las herramientas para la simulación de redes complejas también encontramos a GNS3 [5], Pajek [6] y Mason [7]. Estas herramientas son simuladores multiagente basados en simuladores de eventos discretos. Estos simuladores separan la parte de la visualización del sistema simulado, por lo que se convirtieron en una mejor opción para realizar el modelo propuesto y para el manejo de grandes cantidades de vértices. Al desarrollar este proyecto se contaba con experiencia manejando un simulador de eventos discretos desarrollado en Python [8] por lo que se optó por tomar la idea de las herramientas mencionadas anteriormente y generar la interacción de nuestros agentes sobre este tipo de herramienta y mantener separada la representación gráfica.

En el presente trabajo se propone un modelo que permite la creación de redes a partir de interacciones locales, lo que se busca es identificar qué acciones individuales debe tomar un individuo para formar una estructura global con diversas ventajas. A diferencia de los modelos estudiados, el modelo propuesto se genera tomando en cuenta la simulación basada en agentes para permitir que la red se genere de manera similar a las redes del mundo real, es decir, que se cree como consecuencia de las acciones de agentes independientes. De este modo se pueden identificar algunas estructuras globales que puedan ser útiles en las nuevas tecnologías de la información y las

comunicaciones.

1.1. Objetivos

En el presente trabajo se propone una plataforma de experimentación, basada en un simulador de eventos discretos, en la que se pueden estudiar las diferentes "fuerzas" que le dan forma a los cambios por los que puede atravesar una red. El objetivo es reconocer aquellos parámetros locales de los que pueden emerger propiedades globales interesantes, como las que pueden encontrarse en las redes complejas. A continuación se describen de manera puntual el objetivo general y los objetivos específicos.

Objetivo general: Estudiar las propiedades estructurales de las redes complejas, bajo condiciones dinámicas, utilizando la simulación basada en agentes.

Objetivos específicos:

1. Reconocer un conjunto de medidas que caractericen el estado de una red.
2. Proponer una serie de mecanismos que puedan describir la formación de una red.
3. Evaluar el estado de una red mientras es sometida a los distintos mecanismos de formación.

1.2. Estructura del documento

El resto de este trabajo incluye las siguientes partes: En el Capítulo 2 se revisan los antecedentes de las redes complejas, se da un vistazo a la simulación basada en agentes así como aplicaciones de redes complejas y los trabajos relacionados a este trabajo. En el Capítulo 3 se presenta el modelo propuesto para el estudio de redes complejas, su construcción y la descripción de las herramientas utilizadas. En el Capítulo 4 se presenta la evaluación del modelo, los escenarios propuestos y los parámetros utilizados así como los resultados obtenidos de las simulaciones. Finalmente en el Capítulo 5 se presentan la conclusiones de este trabajo.

Capítulo 2

Antecedentes

Las redes complejas pueden contener miles o millones de *nodos*, por ejemplo, la red que modela a la sociedad global, en donde las personas son vistas como *nodos* y donde, una pareja de nodos comparte una *arista* si las personas que representan se conocen directamente. Decimos que dos nodos o *vértices* están a una distancia 1, si tienen una arista que los conecta directamente. Decimos que están a una distancia 2, si el camino más corto que los une tiene dos aristas. Podemos generalizar esta medida y decir que están a una distancia d, si el camino más corto que los une tiene d aristas [9].

Uno de los descubrimientos más importantes de las redes complejas, con miles de millones de nodos, es que podemos llegar a cualquier persona en el mundo a través de nuestros contactos dando muy pocos saltos, es decir, la distancia que nos separa de otras personas en el mundo no es tan grande como podríamos imaginar. En 1967 Stanley Milgram, psicólogo estadounidense, condujo un experimento para conocer la distancia entre cualesquiera dos personas en los Estados Unidos [10]. El experimento consistió en enviar cartas desde Omaha, Nebraska a una persona en Boston, Massachusetts sin utilizar ningún servicio de paquetería, sólo pasando la carta de mano en mano, es decir, una persona sólo podía entregársela a uno de sus conocidos directos con el fin de direccionar la carta hacia Boston. Las interrogantes eran ¿Podría la carta llegar a su destino? y si fuera así, ¿Cuántas personas se necesitarían para que la carta llegara a su destino? El resultado fue sorprendente pues el número de saltos que dio la carta fue pequeño, en promedio se necesitaron 5.5 saltos para entregar la carta. El primer salto de la carta era el más largo dentro de su recorrido, siento éste el que la acercaba en mayor medida al destino, los restantes sirvieron para localizar el punto de entrega. La propiedad de tener estas distancias en redes complejas es llamada, fenómeno de los mundos pequeños, debido a que la distancia encontrada es

muy pequeña tomando en cuenta la gran cantidad de nodos que conforman la red.

En el área de tecnologías de la información el fenómeno de los mundos pequeños resulta útil, por ejemplo, en la construcción de redes de comunicaciones, en este caso permite tener caminos cortos que ayuden en la pronta entrega de paquetes dentro de redes con un orden tan grande como el de la Internet.

2.1. Teoría de grafos

El estudio de este conjunto de redes hace uso de conceptos de la teoría de grafos para poder analizar las distintas características de cada una y así poder determinar las ventajas o desventajas que pueden tener al ser empleadas en sistemas específicos. Representamos una red compleja como un grafo $G = (V, E)$ que está formado por un conjunto V de vértices, también llamados nodos, y un conjunto E de aristas, también llamadas enlaces. Existen distintas características y conceptos fundamentales que permiten analizar una red [11], algunas de las propiedades más importantes y en las que se centran muchos estudios son:

- **Orden:** Se refiere al número de vértices que forman parte del grafo.
- **Tamaño:** Se refiere al número de aristas que contiene el grafo.
- **Grado de un vértice:** Dado un grafo $G = (V, E)$, el grado de un vértice $v \epsilon V$, se denota como $\delta(v)$, es el número de aristas que inciden en él, si G es dirigido tenemos el grado de aristas entrantes y el grado de aristas salientes. Los vértices con mayor grado usualmente son llamados *hubs* o *concentradores*.
- **Distribución de grados:** Si el grado de un vértice se percibe como una variable aleatoria, entonces puede describirse mediante una función de densidad de probabilidad.
- **Coeficiente de agrupamiento:** Es la medida en que los nodos adyacentes a un vértice v están conectados entre sí. Si el coeficiente de agrupamiento de v es igual a 1 podemos decir que todos los vecinos de v se conocen entre sí (están conectados).
- **Camino ($v_0 - v_k$):** Es una secuencia de vértices y aristas de la forma $v_0, e_1, v_1, e_2 ..., e_k, v_k$ tal que los extremos de la arista e_i son los vértices

$\langle v_{i-1}, v_i \rangle$. Un *camino simple* es un camino en el cual todas las aristas son distintas.

- **Trayectoria:** Es un camino simple en el cual todos los vértices son distintos.
- **Distancia:** Sea $G = (V, E)$ con $u, v \epsilon V(G)$. La *distancia* entre u y v, $d(u, v)$, es la longitud de la trayectoria más corta (u,v).
- **Longitud de trayectoria promedio:** Es el promedio de todas las distancias posibles en la red.
- **Diámetro de una red:** El diámetro es la mayor distancia posible entre dos vértices de G.
- **Componente:** Un grafo es conectado o *conexo* si para todos los pares de vértices distintos $u, v \epsilon V(G)$ existe un camino que los une. Un subgrafo H de G es llamado *componente* si H es conectado y no está contenido en un subgrafo conectado de G que tenga más vértices y aristas.

Además de las características mencionadas se puede realizar un análisis de la red para obtener otras medidas de la estructura de una red, una de ellas es la *modularidad*. Un módulo es un subconjunto de vértices altamente conectados a otros vértices en el mismo módulo y tienen pocos enlaces hacia vértices de otros módulos, también son vistos como agrupamientos o comunidades [12]. Los sistemas modulares son redes de mundos pequeños pues se tienen coeficientes de agrupamiento altos y longitudes de trayectoria cortas. Las características de este tipo de redes se explican más adelante.

La modularidad mide la fuerza de la división de una red en módulos, si se tiene una alta modularidad, entonces cada grupo está altamente definido, es decir, los nodos dentro de un módulo están densamente conectados entre sí y tienen pocas conexiones con otros módulos. En una red con modularidad baja se vuelve difícil separar cada grupo, el número de enlaces entre los nodos pertenecientes al módulo no es tan alto mientras que se tiene un número grande de enlaces entre módulos, lo que dificulta la partición de la red.

Una gran cantidad sistemas sociales o biológicos pueden ser representados con redes para obtener mayor información de ellos. El cerebro es un sistema que puede ser representado por una red, en el trabajo de David Meunier et. al [12] se presenta una revisión del estudio de modularidad en redes cerebrales formadas por datos de neurociencia y neuroimágenes en donde las neuronas pueden ser vistas como nodos. En este trabajo se observa que una

red cerebral tiene la característica de ser modular por lo tanto se tiene una estructura de mundos pequeños lo que puede presentar diversas ventajas en el cerebro, como son distancias cortas entre nodos pertenecientes al mismo módulo.

Existen diversos algoritmos que intentan encontrar una partición de la red que permita tener la más alta modularidad. Como vemos en [13] Roger Guimerá y Luís A. Nunes Amaral proponen una metodología para la extracción de información en redes complejas, en particular particiones que generan una alta modularidad. La idea principal es predecir la ubicación que tendrá un nodo basándose en su conectividad en la red, lo que produce una representación cartográfica de la red. Para encontrar una división de la red que permita identificar los módulos, utilizan el algoritmo de recocido simulado para optimizar la modularidad de la red. Después de extraer los módulos se define el rango o rol que tiene cada nodo tomando en cuenta su conectividad para determinar si nodos con rangos similares se encuentran en los mismos módulos.

Dividir una red en módulos o agrupamientos puede generar estructuras que ofrezcan diversas ventajas, por ejemplo en sistemas de comunicaciones en donde no sea necesaria una completa conexión entre todos los componentes. Encontrar una estructura en la que se tengan módulos permitirá tener conexiones entre los pequeños grupos de equipos y sólo algunas conexiones de largo alcance para interconectar a cada grupo. En [14] M. E. J. Newman y M. Girvan presentan un conjunto de algoritmos para encontrar divisiones en la red que permitan ver a formación de comunidades en la estructura, estas comunidades son subgrupos densamente conectados.

2.2. Redes aleatorias

En 1959 Paul Erdös y Alfréd Rényi introdujeron uno de los primeros modelos para generación de redes aleatorias, consideraron que la estructura que subyace en las redes de la vida real se forma de manera aleatoria, es decir, que en un comienzo en un ambiente en el que ningún individuo se conoce; las relaciones se forman a partir de elecciones aleatorias. Este modelo de grafos es llamado modelo Erdös-Rényi. Al conjunto de redes formadas bajo este modelo se les conoce como *Grafos aleatorios ER* y se identifican como $ER(n, p)$ que describe el conjunto de grafos de n vértices en los que cada posible pareja de vértices está conectada con una probabilidad p. Algunos de los modelos de creación de redes complejas se basan en el conjunto de

grafos $ER(n, p)$.

Para crear un grafo aleatorio ER de n vértices, se toma cada posible pareja de vértices y se les conecta dependiendo de una probabilidad p. Una manera alternativa de crear grafos aleatorios de orden n es estableciendo un número M de aristas y un conjunto de n vértices sin ninguna conexión entre si, entonces se conectan dos vértices seleccionados al azar hasta lograr generar M aristas. Este proceso da lugar a otro conjunto de grafos aleatorios distinto de $ER(n, p)$.

En una red aleatoria podemos tener casos con una probabilidad muy pequeña de conexión o un número M de aristas mucho menor que el número de nodos, en éstas redes existe una alta probabilidad de tener una gran cantidad de nodos o componentes aislados, tal y como se muestra en la Figura 2.1.

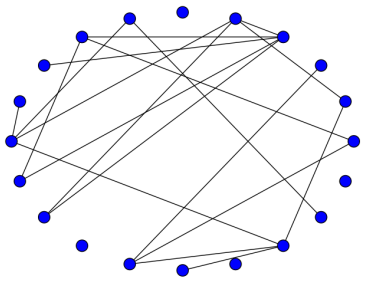


Figura 2.1: Grafo aleatorio ER con probabilidad de conexión $p = 0.1$.

Ahora describiremos las propiedades de este tipo de redes, comencemos con la distribución de grados del grafo. Un grafo $ER(n, p)$ sigue una distribución binomial sobre la variable aleatoria δ, que es el grado de un vértice [11]. La Ecuación 2.2 describe la distribución de probabilidad de los grados de un vértice. El valor esperado del grado de un vértice $v \epsilon G$ es $p(n-1)$.

$$P[\delta(u) = k] = \binom{n-1}{k} p^k (1-p)^{n-1-k} \tag{2.1}$$

Un grafo ER tiene un coeficiente de agrupamiento igual a p, y una longitud de trayectoria promedio igual a $\frac{log(n)}{log(np)}$ [11].

2.3. Redes complejas

Como puede observarse en el mundo real, las conexiones entre personas no sólo se dan de forma aleatoria, en muchos casos los individuos forman parte de una comunidad, lo que podemos ver como agrupamientos de personas y a su vez tienen contactos alejados de esta comunidad. Observaciones de este tipo dan lugar a formar nuevos modelos de redes en los que, además de movimientos aleatorios, se toman en cuenta diferentes estrategias de construcción que dan lugar a propiedades como la de mundos pequeños, gran coeficiente de agrupamiento o distancias cortas entre vértices. En estas redes, cada nodo actúa de manera independiente a los demás y es posible observar a los nodos en conjunto para buscar comportamientos emergentes. A este tipo de redes se les llama comúnmente redes complejas pues podemos verlas como un sistema complejo en el que cada nodo es un agente del sistema. Existen dos modelos importantes de redes complejas: las redes de mundos pequeños y las redes libres de escala. A continuación describiremos con más detalle en que consisten cada una de ellas.

2.3.1. Redes de mundos pequeños

El experimento de Stanley Milgram permitió conocer la longitud de trayectoria promedio en una red formada por la sociedad, sin embargo, la red social del mundo real no es totalmente representada por el modelo de Erdös y Rényi. Sabemos que en la red conformada por la sociedad, ocasionalmente, entablamos relaciones con personas que no se encuentran en nuestro círculo inmediato. Usualmente, tenemos una gran cantidad de conocidos a nuestro alrededor, ya sean vecinos, familiares o compañeros de trabajo, estos son nuestros contactos locales. Algunos de nuestros conocidos tendrán un conocido a una distancia mucho más lejana, y cada uno de nuestros contactos tendrá a su vez nuevos contactos distintos a los nuestros. Este tipo de contactos en la vida real explica los resultados del experimento de Milgram.

En 1998 Duncan J. Watts y Steven H. Strogatz [15] desarrollaron un modelo para la creación de redes cuyas características principales son que se cumpla el fenómeno de los mundos pequeños, es decir, que la longitud promedio de las trayectorias sea pequeña y el coeficiente de agrupamiento alto. Los grafos creados con este modelo son llamados *Grafos de mundos pequeños Watts-Strogatz (WS).*

Para formar un grafo de mundos pequeños WS se consideran n vértices

y un número par k donde $n \gg k \gg ln(n) \gg 1$, los vértices se acomodan en un anillo y se establecen por cada vértice $k/2$ conexiones a la izquierda y $k/2$ conexiones a la derecha. Después, para cada vértice, tomamos cada una de sus aristas y con una probabilidad p de reconexión, reconectamos cada una de sus aristas hacia otro nodo que también se elige al azar. En la Figura 2.2a se muestra un ejemplo de un grafo de mundos pequeños con $p = 0$, se observan los k vértices conectados como se establece al inicio, el grafo resultante es también llamado *Grafo regular*. En la Figura 2.2b se observa que se reconectaron algunas aristas. Lo que es importante a destacar es que si establecemos la probabilidad $p = 1$ como se muestra en la Figura 2.2c, el grafo resultante es de hecho un grafo aleatorio de tipo $ER(n, p)$.

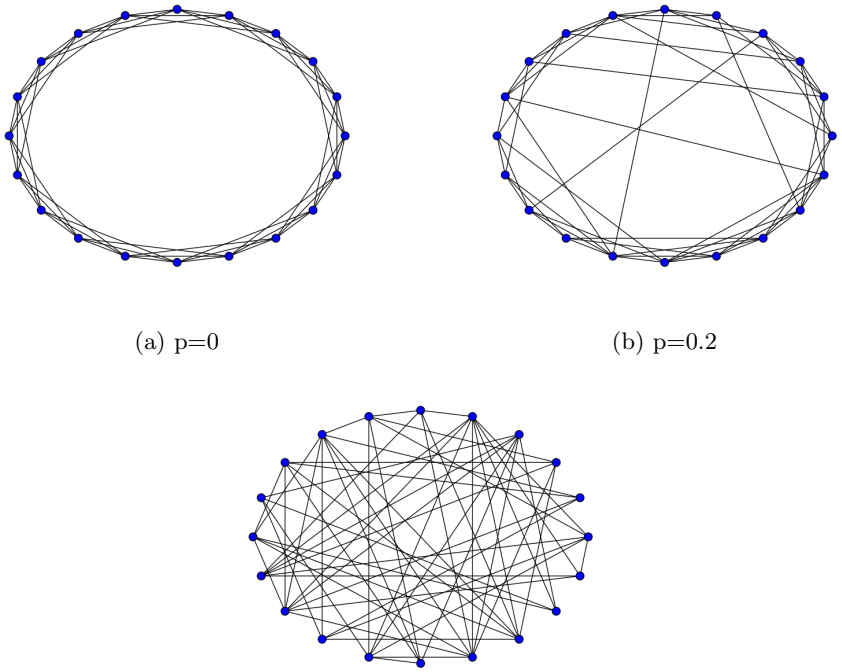
El modelo Watts-Strogatz garantiza que conforme se avanza en la reconexión de vértices, la longitud promedio de la trayectoria y el coeficiente de agrupamiento disminuyen. El grado promedio de G está dado por $p(n-1)$. Recordemos que la principal característica de este tipo de redes es que tienen una longitud de trayectoria promedio pequeña que es de orden $O(logn)$. El coeficiente de agrupamiento es $\frac{3(k-2)}{4(k-1)}$ [11].

2.3.2. Redes libres de escala

Existen redes en el mundo real que tienen características que no pueden ser totalmente representadas por las redes de mundos pequeños. En 1999 Albert-Lázló Barabási y su estudiante Réka Albert [16] realizaron un estudio en el que se observó que la distribución de grados de muchas redes sigue un distribución de ley de potencias. Las redes que tienen este tipo de distribución son llamadas *Redes libres de escala*. Este nombre se debe a que la ley de potencias tiene la característica de ser libre de escala, lo que significa que observaremos el mismo comportamiento de la función en distintas escalas.

El *modelo Barabási-Albert (BA)* refleja una característica importante que es la formación de un subconjunto muy pequeño de nodos con un grado muy alto, a los que se denomina nodos concentradores. En una red común para nosotros como Internet, por ejemplo, los concentradores más grandes e importantes son los nodos que permiten la conexión entre los continentes. Barabási y Albert nos proporcionan un modelo denominado *con preferencias de conexión*, en el que crece el número de nodos hasta alcanzar un cierto orden.

El procedimiento para crear un grafo del modelo BA, también llamados



(a) p=0 (b) p=0.2

(c) p=1

Figura 2.2: Grafos de mundos pequeños con 20 vértices, k=6 y distintas probabilidades de reconexión.

Grafos BA, comienza con una red pequeña de tipo Erdös-Renyi, en cada iteración insertaremos un vértice v a la red con un número m de aristas conectadas a él. Posteriormente se conectan sus aristas con vértices que ya pertenecen a la red, esta conexión se da de acuerdo a una probabilidad proporcional al grado del vértice elegido. Este procedimiento se sigue hasta conseguir que la red tenga n nodos.

La distribución de grados de este tipo de redes esta dada por $P[k] \propto k^{-\alpha}, \alpha > 1$, α es un exponente de escala. Esto nos indica que existen pocos vértices que tienen un grado muy alto y muchos vértices con un grado bajo. La longitud promedio de trayectoria es $\frac{log(n)}{log(log(n))}$ [11].

Las grafos BA describen sistemas tolerantes a fallos, un fallo sucede cuando un elemento aleatorio en la red detiene su funcionamiento. Reuven

Cohen et. al [17] analizaron cómo actúa una red ante constantes fallos, y encontraron el porcentaje de vértices que pueden fallar antes de que la red se divida o desconecte en varios componentes. Mostraron analítica y numéricamente que, para redes con distribución de ley de potencias y $\alpha \leq 3$, esta desconexión de vértices logra suceder cuando el número de vértices en fallo es muy alto. Internet tiene una distribución de grados con $\alpha \approx 2.5$. En el caso de Internet se mostró que es resistente a fallos, el resultado de Reuven Cohen et. al nos dice que estando cerca del cien por ciento de vértices en fallo, la red aun permanece conectada.

En un fallo es muy probable que la pérdida de un nodo no implique una desconexión de toda la estructura de la red, pero si el nodo que falla es elegido premeditadamente con un conocimiento de sus características tal vez logremos desconectar a la red. En 2001 Reuven Cohen et. al [18] realizó un análisis sobre la destrucción de Internet a través de ataques intencionados, el resultado fue sorprendente pues, a pesar de que estas redes que tienen un $\alpha \leq 3$ y por lo tanto son muy resistentes a fallos, son muy sensibles a ataques intencionales. Basta con atacar un número muy pequeño de concentradores para que la red se divida en componentes aislados.

2.4. Simulación basada en agentes

La simulación por computadora es un medio de investigación que ha sido usado con mayor frecuencia en la actualidad debido a que es menos complejo que un modelo analítico, además de que existen algunos modelos analíticos que no tienen solución, y es menos costoso que una implementación real. La simulación es útil para el estudio de fenómenos biológicos y tecnológicos ya que se puede observar la dinámica de cada componente del sistema simulado. Los sistemas complejos se forman de interacciones entre individuos que actúan bajo un conjunto de reglas locales y reaccionan al ambiente de acuerdo a las mismas, por lo que la simulación basada en agentes es de mucha utilidad.

La importancia de la simulación basada en agentes se observa en el trabajo de Joshua M. Epstein [19]. Epstein nos presenta una simulación basada en agentes que se desarrolla en un plano en dos dimensiones llamado *Sugarscape*, este plano cuenta con un recurso: azúcar. El recurso se encuentra distribuido por todo el plano. Como reglas iniciales un agente necesitará consumir esta azúcar para sobrevivir. A lo largo del experimento se describen reglas muy sencillas de movimiento, muerte, reproducción, enfermedad, amistad y

comercio, por mencionar algunas. La ejecución a lo largo de un tiempo de esta simulación permite observar comportamientos globales totalmente inesperados, se dice que estos comportamientos emergen a partir de un conjunto de sencillas reglas locales de interacción. Como ejemplo de estos comportamientos globales emergentes tenemos: la formación de grupos, el combate, el comercio, la difusión de enfermedades y la definición de territorios para cada grupo formado. En este trabajo se han utilizado las simulaciones basadas en agentes para estudiar fenómenos como la distribución de la riqueza, que sigue una distribución de ley de potencias.

En 2013 Tadahiko Murata et.al [20] estudiaron el fenómeno de la migración tomando en cuenta sólo la red social de contactos para averiguar los espacios más ricos en recursos. Se analiza el problema bajo los distintos modelos de redes complejas, al final todos los individuos logran llegar a los lugares con mejores recursos. Los agentes se mueven frecuentemente cuando la red es altamente conectada y no es un centro económico obvio.

Las necesidades más importantes en la simulación de redes complejas son: manejo de grafos con cantidades masivas de nodos, herramientas de creación de redes, herramientas de visualización y una sencilla simulación de eventos en la red.

2.5. Aplicaciones

Las propiedades que ofrecen las redes complejas son útiles para el área de las Tecnologías de la Información y las Comunicaciones, si bien la mayoría de las investigaciones se aplican a problemas sociales, biológicos o físicos, existen muchas aplicaciones en tecnologías de la información que utilizan los beneficios de las redes complejas.

Un ejemplo en el área de sistemas paralelos y distribuidos se describe en el trabajo de Gurmeet Singh Manku et. al [2]. Una red par a par (P2P) dinámica contiene Tablas Hash Distribuidas, que contienen información actualizada del estado de la red. Cada vez que ocurre un cambio en la estructura de la red, ya sea por la adición o eliminación de un equipo, se deben actualizar las tablas. Dichas actualizaciones generan costos grandes y limitan la escalabilidad en la red. La solución que proponen es conectar los equipos de la red para que formen una estructura de red de mundos pequeños. Como resultado se tuvo una red que necesitaba pocos enlaces hacia otros nodos de la red, y un costo bajo en la actualización de las tablas pues se aprovecha el fenómeno de mundos pequeños para agilizar este proceso.

En conclusión se logra que un red de este tipo sea escalable.

Recientemente se ha incrementado el uso de redes complejas en el estudio y creación de *redes inalámbricas de sensores* [3]. Por ejemplo, para disminuir el tráfico urbano, se han implementado distintos *sistemas de transporte inteligentes*. Para la instalación de un circuito cerrado de televisión, se coloca un sensor en las intersecciones que cuentan con cámaras de circuito cerrado para revisar constantemente el tráfico. Cuando el sensor requiere enviar los datos captados en un punto se necesita que esto se haga de manera rápida si es el caso de una emergencia. Para optimizar las características de este sistema se propuso incrementar la potencia de algunos sensores y calcular la mejor distribución de estos de manera que se tenga una topología de red de mundos pequeños. Con esta topología se reduce el número de saltos para entregar la información, se incrementa la velocidad de transmisión y aumenta la eficiencia del sistema de transporte inteligente.

2.6. Trabajos relacionados

Nuestro interés en este trabajo es estudiar cómo cambian las propiedades estructurales de una red a través del tiempo, ya sea bajo procesos de formación, como de degradación de la red. Además queremos observar qué tipos de redes emergen utilizando distintos parámetros en su construcción.

En el trabajo de Wei Qi et. al [21] se analizan las características de una red libre de escala en forma dinámica, tomando un modelo de red distinto al modelo Barabási-Albert. En el nuevo modelo, durante la formación normal de una red libre de escala, se agregan además aristas entre los nodos ya existentes en la red. Se analizó el coeficiente de agrupamiento, el grado de los vértices y la longitud de trayectoria promedio. Las medidas de estas características se tomaron paso a paso conforme aumentaba el tamaño de la red. Ambos modelos, Barabási-Albert normal y Barabási-Albert con enlaces entre nodos existentes, fueron contrastados con una red de comercio electrónico real en la que se observa como los clientes que se suscriben en ella forman enlaces con otros clientes del portal. El resultado de este experimento indica que en la medida en que se incrementaba la escala de la red, el número de vértices con un alto grado se mantiene sin cambios, mientras que el coeficiente de agrupamiento y la longitud de trayectoria promedio disminuyen hasta volverse estables. Además se observó que el modelo propuesto es más parecido a las redes del mundo real. El análisis realizado en [21] nos permite saber cuáles características nos pueden dar una mayor información

de la red, además es un ejemplo de cómo el estudio desde una perspectiva dinámica nos otorga una gran cantidad de información para conocer las propiedades de la red.

Ahmed Helmy [22] estudia el fenómeno de las redes de mundos pequeños dentro de redes inalámbricas de sensores. Estas son redes espaciales que tienden a tener un coeficiente de agrupamiento más alto que el de las redes aleatorias, así como una longitud de trayectoria promedio del orden de $O(\log n)$. La experimentación se realizó sobre una área de $1km^2$ en la que se colocaron los nodos formando diversas topologías iniciales. Se realizaron dos tipos de experimentos, en el primero se recableó el enlace de un nodo hacía otro elegido con alguna probabilidad. En el segundo experimento se agregaron enlaces entre dos nodos seleccionados con cierta probabilidad. Encontraron que añadiendo un número relativamente pequeño de nuevos enlaces(entre 25 % y 40 % de los nodos totales de la red), que formen atajos o enlaces de largo alcance, se puede disminuir drásticamente el diámetro de la red.

En el trabajo de Tao Jia et. al [23] se estudia una variante de las redes de mundos pequeños en el que se agrega un parámetro, el alcance máximo de los enlaces. Las redes obtenidas bajo este modelo tienen una distribución de grados similar a los grafos de mundos pequeños. Observaron que, si la red tiene una gran cantidad de nodos, la longitud de trayectoria promedio disminuye en mayor medida. Se encontró que no es necesario tener enlaces de gran longitud para conseguir que se cumpla la propiedad de mundos pequeños. Se tienen comportamientos emergentes, uno de ellos es el comportamiento no monotónico de la longitud de trayectoria promedio. Para el estudio del modelo propuesto se utilizó la simulación por computadora. A diferencia del modelo que proponemos, en [23] el recableado se realiza completamente al azar y esta decisión es tomada de manera centralizada.

Capítulo 3

Metodología

En este trabajo se estudia la estructura de las redes complejas de manera dinámica, para esto es importante establecer un modelo de construcción de red que permita obtener datos de la estructura del grafo durante su ejecución. El modelo propuesto está inspirado en la formación de vínculos o enlaces entre personas. Como sucede en estos casos una persona sólo está ligada a su comunidad inmediata, como su familia, sus vecinos y compañeros de trabajo. Sin embargo, al pasar el tiempo estos mismos vínculos iniciales le ayudan a construir lazos de *larga distancia*. Por ejemplo, un estudiante de posgrado cuenta con enlaces locales hacia miembros de su área de conocimiento en la universidad. Después de cierto tiempo comienza a estudiar un tema particular en el que su asesor de tesis le provee artículos y medios necesarios para su investigación. Si el asesor cuenta con un contacto de otro país facilitará al estudiante información sobre su trabajo. Al avanzar el tiempo el estudiante requerirá ponerse en contacto directamente con ese conocido lejano, esto agilizará su intercambio de información.

Basados en este ejemplo, en el modelo propuesto, cada nodo utiliza a sus vecinos para encontrar a aquellos nodos de la red que le resultan útiles para establecer enlaces de largo alcance. A continuación se describe el modelo propuesto, su construcción y parámetros.

3.1. Descripción del modelo propuesto

El modelo comienza con una red base con estructura de malla de dos dimensiones. Una malla de $n \times m$ es un grafo que consta de n filas y m columnas de vértices, de este modo podemos tomar en cuenta la posición de cada nodo en la malla usando sus coordenadas. Cada nodo cuenta con

un conjunto de vecinos locales, un vecino es aquel nodo con el que se tiene una arista en común. El vecindario de un nodo es el conjunto de todos sus vecinos, en este trabajo cada vértice cuenta con un vecindario denominado de Von Neumann que consta de los nodos a la derecha, izquierda, arriba y abajo como se muestra en la Figura 3.1.

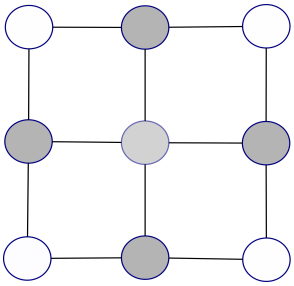
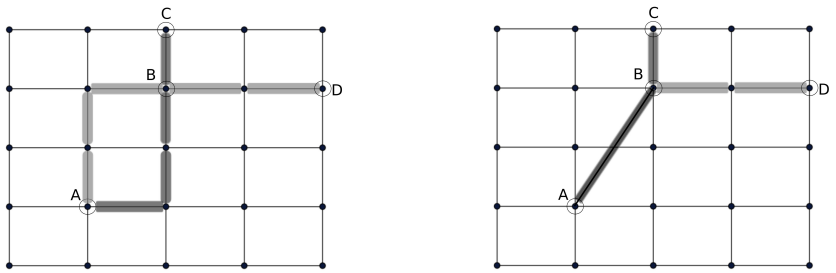


Figura 3.1: Vecindario de Von Neumann

Cada uno de los nodos enviará paquetes a destinos aleatorios con el fin de encontrar aquellos nodos que le convengan como vecinos, podemos observar esta idea en la Figura 3.2. Cuando un nodo termina de enviar sus paquetes puede *observar* las rutas que siguieron, por ejemplo, en la Figura 3.2a, el nodo A envía paquetes a los nodos C y D, cada paquete ocupa 4 y 5 saltos respectivamente para llegar a su destino, ambos atraviesan por el nodo B. Esto indica que B es un posible candidato a ser un vecino ya que tener un enlace conectado hacia B agiliza el envío de paquetes. Si el nodo A decide agregar una arista hacia el nodo B como se observa en la Figura 3.2b, el número de saltos que ocupa cada paquete disminuye y el número de aristas en la red aumenta. Por otro lado si decidimos tomar una de las aristas existentes entre el nodo y alguno de sus vecinos para conectarla hacia el nodo B tenemos una acción de *recableado*, esta acción evita incrementar el número de aristas en la red y de igual manera permite reducir la ruta seguida por los paquetes.

Si queremos realizar un recableado es necesario saber qué enlace moveremos. Con las rutas recogidas por un nodo se puede conocer a su vecino menos utilizado para poder desconectarse de él y mover ese enlace hacia el nodo que más utilizó en la red. El vecino menos usado debe tener una utilidad menor que la del nodo candidato.



(a) Envío de paquetes (b) Creando enlaces

Figura 3.2: Idea principal del modelo propuesto.

3.2. Construcción de la simulación

Para descubrir los nodos más útiles en la red cada nodo envía un conjunto de paquetes con destinos elegidos al azar. Cuando llegan a su destino se devuelve el paquete como acuse de recibo al nodo origen, por la misma ruta que siguió el paquete para llegar. La estructura de un paquete enviado se muestra en la Figura 3.3. El paquete contiene el identificador del nodo origen, las coordenadas del nodo origen, el identificador del nodo destino, una lista en la que se almacena la ruta que sigue el paquete para llegar a su destino y una lista en la que se almacenan los nodos que no le sirven para llegar al destino. El llenado de la lista de nodos descartados se explica en la Subsección 3.2.2.

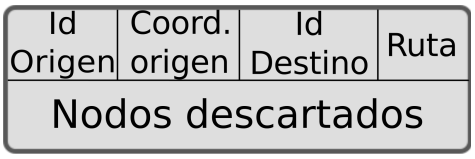


Figura 3.3: Estructura de un paquete

En el simulador definimos como *ciclo* al intervalo de tiempo durante el cual se realizan todas las operaciones necesarias para efectuar un cambio en la estructura de la red. El simulador sigue las siguientes reglas de comportamiento.

3.2.1. Reglas de comportamiento

Dentro del modelo existen comportamientos locales que son realizados por cada nodo de la red y comportamientos globales que describen cada etapa que sigue la simulación.

Comportamiento local en cada nodo:

- Cada nodo genera un número determinado de paquetes que es direccionado hacia un nodo escogido aleatoriamente.
- Cada paquete almacena la ruta por la que viaja, al volver al nodo de origen se hace un conteo de los nodos que fueron visitados en la red y de los enlaces locales que se utilizaron.
- Se selecciona el nodo que resulta más útil de acuerdo a un umbral fn, este nodo es candidato a ser un nuevo vecino.
- Se selecciona nuestro enlace menos utilizado de acuerdo a un umbral fe, este enlace es candidato a ser recableado.
- Al recablear cada enlace, se realiza el intercambio necesario de mensajes entre los nodos involucrados en la desconexión y conexión para saber si están de acuerdo con el cambio.

Un ciclo de simulación se compone de distintas etapas que permitirán generar un cambio en la estructura. Estas etapas definen el comportamiento global del simulador.

Comportamiento global del simulador:

- Etapa de envío de paquetes: en esta etapa cada nodo envía su conjunto de paquetes y espera el regreso de todos anotando a los nodos que cada paquete incluye en sus rutas.
- Sincronización para concluir la etapa de envío de paquetes.
- Etapa de recableado: en esta etapa cada nodo analiza los datos obtenidos en la etapa de envío de paquetes para crear o recablear enlaces.
- Sincronización para concluir la etapa de recableado de enlaces.
- Se guarda cada cambio de la etapa de recableado y se comienza el nuevo ciclo sobre la nueva estructura.

3.2.2. Algoritmo de encaminamiento

En la etapa de envío de paquetes se seleccionan al azar los destinos para cada paquete. Para direccionar un paquete a su destino se implementó el algoritmo Compass Routing propuesto por E. Kranakis et. al [24] que ayuda a elegir el siguiente nodo a quien se enviará el mensaje tomando en cuenta la posición geográfica. El nodo que se elige es el que espacialmente se encuentra más cerca de el destino. El algoritmo marca una linea recta principal entre origen y destino. Se compara la recta formada entre origen y cada vecino con la recta principal. El mejor candidato para enviar el mensaje será el que genere un menor grado de separación entre la recta principal y la recta con el vecino.

Si bien por si mismo Compass Routing nos da buenos resultados tiene una desventaja, la formación de ciclos en el encaminamiento. Para solucionar esto se tomó en cuenta la ruta que lleva guardada el paquete para evitar enviar el paquete a los nodos que visitó anteriormente. Si este camino no nos lleva al destino regresamos al punto de decisión anterior y elegimos otro hasta encontrar el que nos lleve al destino. Los nodos que ocasionan un retroceso en el paquete, son agregados a la lista de nodos descartados para que no se tomen en cuenta cuando se deba elegir un nuevo camino.

Si la red se ha separado en diversos componentes existirán paquetes que nunca podrán llegar a su destino, las modificaciones al algoritmo de encaminamiento permiten al paquete regresar al nodo origen, para que pueda continuar con su envío de paquetes.

Dentro del simulador es posible modificar el algoritmo de encaminamiento de manera sencilla. Al final el paquete guarda la ruta que le permitió llegar al destino, en esta ruta no se guardan los caminos que se visitaron y no fueron útiles para el paquete, es decir, los que ocasionan el retroceso de un paquete.

3.2.3. Etapa de recableado

Dentro de la etapa de recableado cada integrante de la red elegirá al nodo que más visitó enviando paquetes, al que denominaremos candidato. Además determinará cuál de sus enlaces ha sido el menos usado, para poder desconectarlo y reconectarlo hacia el sitio más visitado. Cada nodo debe tomar en cuenta lo siguiente:

- Cada enlace tiene un tamaño definido lo que genera un alcance limi-

tado, por lo que un nuevo vecino debe estar dentro de su alcance para ser un candidato a nuevo vecino.

- La frecuencia de visita hacia el candidato debe ser mayor a una frecuencia fn. Es decir, que nos conviene establecer enlaces directos hacia los nodos que más visitamos en la red.
- La frecuencia de visita de un enlace debe ser menor que una frecuencia fe. Es decir, que si todavía es un enlace utilizado no nos conviene modificarlo.

En cada ciclo todos los nodos intentarán recablear uno de sus enlaces pero sólo algunos lo conseguirán. Cada enlace que ha sido recableado exitosamente se almacena en una traza en la que se separan los cambios dependiendo del ciclo en el que se realizaron, esto con el fin de calcular las medidas que necesitamos.

Consideraciones especiales

Dentro del simulador podemos establecer que los nodos sean dueños de un conjunto de enlaces o ninguno. Las acciones tomadas por parte de cada nodo para llevar a cabo el recableado tienen casos especiales que se describen a continuación.

Recableado de enlaces sin dueño. Una acción de recableado se compone de dos partes: la desconexión de un vecino y la conexión a un nuevo nodo.

Cuando un nodo decide desconectarse de un vecino, para conectarse a otro nodo en la red, existe la posibilidad de que su vecino decida desconectarse de él en el mismo instante de tiempo. Para desconectarse de un vecino, un nodo debe enviar una petición de desconexión. Si el vecino no permite la desconexión contesta con un acuse de recibo negativo para indicarlo, en caso contrario contesta con un acuse de recibo positivo.

Este caso especial se observa en la Figura 3.4, para resolver esta situación, cada nodo almacena el identificador del nodo que le envió una petición de desconexión para compararlo con su identificador. El nodo con menor identificador se queda con el enlace, en el ejemplo, el nodo 1 se queda con el enlace por lo que envía un acuse de recibo negativo para informar a 2 que no debe desconectarse de él. El nodo 2, al tener la misma condición, envía

un acuse de recibo positivo a 1 para permitir que el nodo se desconecte y se quede con el enlace. Al recibir un acuse de recibo positivo un nodo puede continuar con la segunda etapa de recableado, la conexión. Al recibir un acuse de recibo negativo el nodo termina su recableado sin generar ningún cambio en la red.

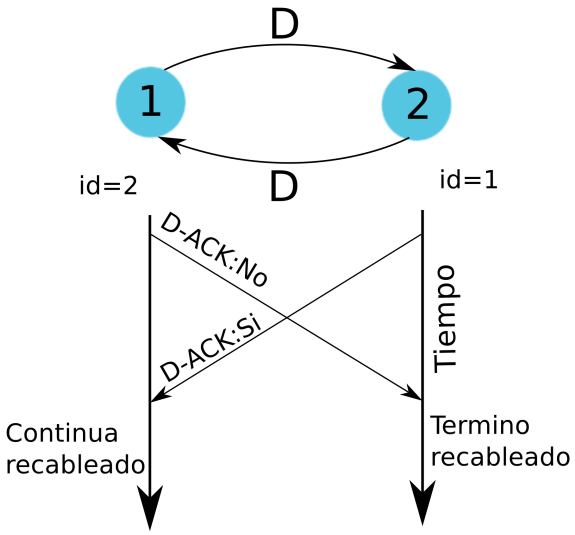


Figura 3.4: Desconexión de un enlace en común

Recableado de enlaces con dueño. Si tenemos el caso de creación de enlaces, cada nodo es dueño de los enlaces que genere por lo que no se presenta el caso especial anterior, ya que cada nodo es libre de desconectar sus propios enlaces. En cambio se presenta un caso similar para la conexión, cuando dos nodos intentan conectarse entre sí al mismo tiempo, este caso se resuelve igual que en el caso anterior. El nodo con menor identificador podrá crear ese enlace, mientras que el otro nodo tendrá que elegir a otro nodo para conectarse.

De la Figura 3.5 podemos observar que, cuando el nodo 1 desea recablear el enlace que tiene con el nodo 3 envía una petición de desconexión, el nodo 3 lo agrega en una lista de peticiones recibidas y le informa al nodo 1. El nodo 1 tiene que asegurarse que podrá conectar el enlace por lo que envía un mensaje de conexión al nodo 2. En este punto surgen dos casos.

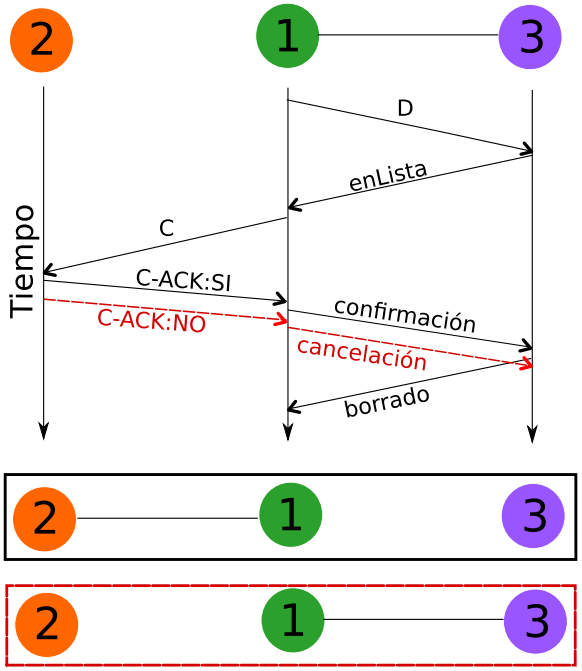


Figura 3.5: Recableado de enlaces con dueño

Caso 1: el nodo 2 responde que no, por lo tanto el nodo 1 envía un mensaje de cancelación a 3 y termina su etapa de recableado.

Caso 2: el nodo 2 contesta que sí, entonces el 1 envía una confirmación a 3 para que lo borre de sus vecinos. Cuando 3 completa la desconexión avisa a 1 que ha sido borrado. Al recibir este mensaje 1 actualiza a sus vecinos agregando a 2 y borrando a 3, con esto completa su etapa de recableado.

3.2.4. Sincronización

Todos los nodos actúan de manera distribuida por lo que es necesario establecer un mecanismo de sincronización en la etapa de envío de paquetes, ya que se necesita que todos los nodos terminen para tener la información completa antes de pasar a la siguiente etapa. La sincronización también se hace al finalizar la etapa de recableado, pues es indispensable que todos los cambios en la estructura terminen para poder comenzar un nuevo ciclo.

Definimos un nodo para coordinar la sincronización. El coordinador emplea un algoritmo de propagación de información con retroalimentación o

PIF [25], al recibir respuesta de todos los nodos de la red el coordinador da por concluida la etapa de correspondiente. Para informar de esta situación a todo el grafo se ejecuta un algoritmo de propagación de información o PI [25] que además indica el comienzo de la siguiente etapa o ciclo.

3.3. Herramientas utilizadas

Para poder realizar el modelo propuesto es necesario contar con una herramienta que permita obtener datos en cada momento de cambio de la red. Cada nodo trabaja de manera distribuida por lo que el modelo se desarrolló utilizando la simulación de eventos discretos.

Para la construcción del modelo se utilizó el lenguaje de programación Python gracias a que se contaba de antemano con un simulador de eventos discretos [8] realizado en este lenguaje que nos facilita su programación. Además Python cuenta con una gran variedad de funciones que facilitan el manejo de listas y eso fue de gran ayuda para el modelo. El simulador de eventos discretos nos falicita la tarea de manejar un ambiente distribuido con el uso de funciones sencillas para el envío y recepción de mensajes. Finalmente, en Python contamos con la librería para análisis de grafos llamada NetworkX que nos permite calcular diversas características que se desean estudiar en el modelo.

La idea básica de construcción de la simulación se muestra en la Figura 3.6. Comienza con la creación de una malla en Python, después se ejecuta el simulador de eventos discretos en el cual se programa el comportamiento local de cada nodo mencionado anteriormente, al final se obtiene una traza que contiene cada acción de recableado realizado en la red.

Para el cálculo de las características de red que deseamos observar se utiliza NetworkX, esta librería lee la traza generada por la simulación generando los cambios especificados en la red y obtiene las características deseadas que se guardan para su posterior análisis. Por cada ciclo se almacena la matriz de adyacencia y la distribución de grados.

Como un módulo opcional se utilizó la librería Matplotlib para generar una animación de la simulación. Esta herramienta es útil para redes con un tamaño pequeño; pero ineficiente para redes con un número mayor de nodos. Cada matriz de adyacencia de los grafos generados se almacena en un formato estándar que puede ser leído por alguna plataforma de visualización de grafos.

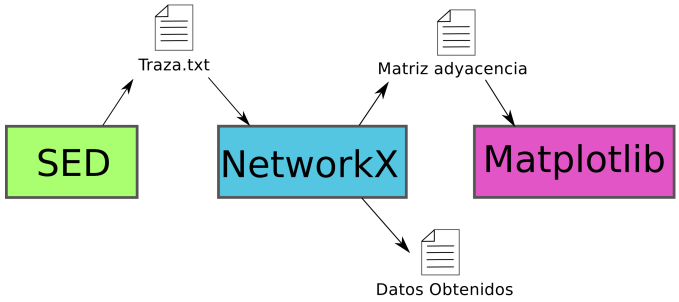


Figura 3.6: Esquema de la comunicación entre herramientas

3.4. Parámetros del simulador

El simulador fue programado tomando en cuenta diversos parámetros que son importantes para la construcción de la red, ya que pueden influir en gran manera en la red resultante. Los parámetros de simulación se muestran en la Tabla 3.1.

Tabla 3.1: Parámetros del simulador

Parámetro	**Descripción**
· Enlaces existentes o nuevos	De este parámetro parten dos modelos importantes, por un lado una simulación en la que cada nodo sólo recablea sus enlaces existentes y por otro lado una simulación en la que se crean nuevos enlaces.
· Nodo cooperador	Esto indica que un nodo puede cooperar para que se establezcan nuevos enlaces hacia el o en caso contrario indica que este nodo no es un cooperador.
· Longitud del enlace	Indica el alcance máximo de un enlace, de este parámetro depende el alcance que tiene el nodo para conectarse con sus candidatos.
· Número de enlaces soportados	Indica el número máximo de enlaces que soporta cada nodo
· Frecuencia de nodos (fn)	Indica la frecuencia mínima de visita para que un nodo sea considerado como el más visitado.
· Frecuencia de enlace (fe)	Indica el uso mínimo de un enlace, por debajo del cual se considera *sub-utilizado*.

Capítulo 4

Evaluación

Para poder observar la dinámica estructural de las redes formadas bajo el modelo propuesto, se proponen tres escenarios que toman en cuenta distintos parámetros al momento de crear una red. En este trabajo estudiamos el efecto de la longitud del enlace dentro del modelo, comparando las medidas de desempeño obtenidas de cada escenario.

4.1. Escenarios estudiados

En los primeros dos escenarios se estudia el parámetro de la longitud de enlace, en un modelo que únicamente recablea enlaces existentes. Los resultados obtenidos de estos primeros escenarios permiten establecer la configuración del tercero. En este se establece una longitud de enlace intermedia a la de los anteriores y se agrega una característica más, la creación de nuevos enlaces. Por cada escenario se realizan experimentos en mallas con 100, 400, 900, 1600 y 2500 nodos.

4.1.1. Primer escenario: Longitud de enlace acotada

Para observar la importancia del tamaño de enlace en la simulación el primer escenario propuesto utiliza como parámetro una longitud acotada de tamaño 3. Podemos ver que una longitud de enlace igual a 1 lleva a un nodo a visitar únicamente a sus vecinos iniciales por lo que, una longitud de 3 permite al nodo llegar a los vecindarios cercanos al suyo. Tomando en cuenta la longitud del enlace, un nodo que sea candidato para ser un nuevo vecino debe estar dentro del alcance del nodo que desea recablear uno de sus enlaces. Como se muestra en la Figura 4.1 el nodo intenta mover su enlace e_A

menos usado hacia alguno de los nodos dentro del área sombreada. En este escenario se presenta el caso especial de recableado con enlaces sin dueño explicado en la Subsubsección 3.2.3.

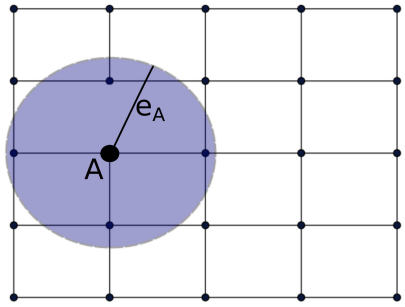


Figura 4.1: Primer escenario: Longitud de enlace acotada

4.1.2. Segundo escenario: Longitud de enlace ilimitada

En este escenario se establece una longitud de enlace infinita, es decir, un nodo puede conectar uno de sus enlaces menos utilizados hacia cualquier nodo que le sea útil sin importar su posición en la malla. En la Figura 4.2 observamos este escenario, el enlace e_A puede ser conectado a cualquier nodo de la red. En este escenario se presenta el caso especial de recableado con enlaces sin dueño explicado en la Subsubsección 3.2.3.

4.1.3. Tercer escenario: Longitud de enlace media con enlaces extra

En los escenarios anteriores cada nodo puede desconectar sus enlaces de sus vecinos y conectarlos a otros nodos, lo que puede ocasionar que algunos nodos queden aislados de la red agrupados en pequeños subgrafos. Para solucionar esta situación en el tercer escenario los vecinos iniciales, indicados por la malla, quedan fijos y dejan de ser posibles enlaces para recablear, lo que evita la descomposición de la red. Además de los enlaces fijos cada nodo tiene un conjunto extra de enlaces que en un inicio no están ligados a un nodo, como se muestra en la Figura 4.3, la longitud de cada enlace extra

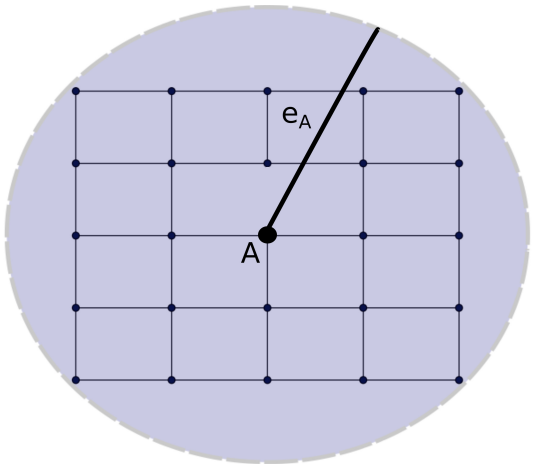


Figura 4.2: Segundo escenario: Longitud de enlace ilimitada

es igual a la distancia entre el centro del grafo y el punto horizontal más alejado a éste. En este caso el nodo central no alcanza a los nodos de las esquinas, pero si a la mayor parte de los nodos de la red.

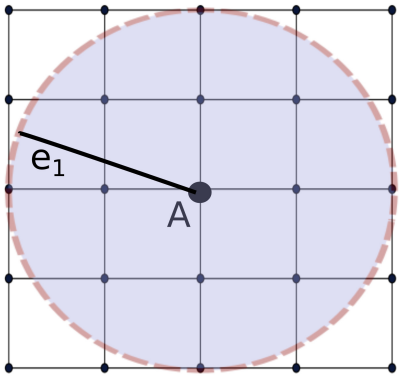


Figura 4.3: Tercer escenario: Longitud de enlace media con enlaces extra

Al comenzar la simulación cada nodo intenta conectar sus enlaces extra hacia el nodo que le resulta más útil en la red. Cuando termine de conectarlos comenzará a recablear, como en los escenarios anteriores, pero esta vez sólo

puede mover a sus enlaces extra. En este escenario se presenta el caso especial de recableado con enlaces con dueño explicado en la Subsubsección 3.2.3.

4.2. Configuración de parámetros

Los experimentos se realizaron usando la configuración de parámetros de la Tabla 4.1. Para determinar el número de ciclos adecuado, se realizaron experimentos previos dejando el parámetro de número de ciclos libre y especificando un criterio de paro. Este criterio toma en cuenta el número de enlaces que son recableados por ciclo. En todas las simulaciones se observó que el número de enlaces recableados totales en la red disminuye, por lo que para detener el experimento el número de enlaces recableados en toda la red debe ser menor que el 1 % del total de enlaces que existen en el grafo. Encontramos que para el primer escenario las simulaciones pararon antes de 30 ciclos. El segundo escenario los experimentos terminaron en promedio en el ciclo 34 para las redes con mayor cantidad de nodos. Observando los movimientos que se realizaron en los últimos ciclos, se encontró que el número de enlaces recableados en cada ciclo era similar y que todos los nodos que movían un enlace lo recableaban desde un nodo concentrador a otro y en el siguiente ciclo regresaban al nodo anterior. Se estableció la duración de los experimentos en 30 ciclos, pues en todos los casos el comportamiento se mantenía igual después de este punto y para simulaciones más grandes se limita el tiempo de los experimentos.

Por cuestiones de tiempo, el número de paquetes enviados por nodo es 20 para no tener un tiempo de simulación tan grande en este conjunto de experimentos. Los demás datos se fijaron para obtener un primer conjunto de experimentos a analizar, en trabajo futuro se analizará el impacto de los mismos.

4.3. Medidas de desempeño

Por cada red obtenida medimos el coeficiente de agrupamiento promedio, el diámetro de la red, la longitud de trayectoria promedio y el número de enlaces recableados en cada ciclo de cambio. Además se obtuvo la distribución de grados de la red final. Estas medidas se obtuvieron usando networkX. Al leer la traza de los cambios realizados por ciclo, se genera un grafo por cada ciclo y se le toman las medidas antes mencionadas.

Tabla 4.1: Configuración de parámetros

Parámetro	**Valor**
· Nodo cooperador	Todos cooperan.
· Longitud del enlace	· 1er escenario = 3. · 2do escenario = $infinito$ · 3er escenario = $no.columnas/2$
· No. paquetes por ciclo	20
· No. ciclos	30
· Frecuencia de nodos (fn)	50 % de los paquetes
· Frecuencia de enlace (fe)	20 % de los paquetes
· No. de enlaces extra	2

En caso de que exista una separación del grafo en varios componentes, las medidas de desempeño se obtienen del componente con mayor número de nodos. El componente con mayor número de nodos suele llamarse *componente gigante* y es un subgrafo de la malla original. Se tomó esta medida ya que la cantidad de nodos que se encuentran fuera del componente gigante es pequeña.

4.4. Resultados

De las simulaciones realizadas se observa una disminución en el diámetro, longitud de trayectoria promedio (LTP) y un aumento en el coeficiente de agrupamiento. En la Tabla 4.2 se observan los datos que tiene una red inicial con estructura de malla. En las simulaciones del primer escenario podemos observar una gran disminución en las medidas, tomando en cuenta que el alcance de los enlaces es muy corto. En la Tabla 4.3 podemos observar los datos de las redes resultantes del primer escenario en los distintos tamaños de red. Debido a lo longitud de los enlaces se pueden formar algunos grupos por lo que el coeficiente de agrupamiento aumenta.

En la Tabla 4.4 observamos los datos finales del segundo escenario, recordemos que en este escenario la longitud del enlace es infinita. En estas simulaciones se observó la formación de topologías muy cercanas a la de es-

trella. Como podemos ver en la Tabla 4.4, tanto el diámetro como la longitud de trayectoria promedio, disminuyen casi al mínimo en todos los tamaños de red. Si bien las características de una red de este tipo son buenas, las mismas ocasionan que estas redes sean muy vulnerables a ataques que provoquen su total desconexión. En este caso, el coeficiente de agrupamiento es más alto que el del primer escenario.

Tabla 4.2: Datos iniciales

Orden del grafo	**Coeficiente agrupamiento**	**Diámetro**	**LTP**
100	0	18	6.66666666667
400	0	38	13.3333333333
900	0	58	20
1600	0	78	26.6666666667
2500	0	98	33.3333333333

Tabla 4.3: Resultados del primer escenario

	Escenario 1		
Orden del grafo	**Coeficiente agrupamiento**	**Diámetro**	**LTP**
100	0.2426512932	7.4	3.7604067691
400	0.2161459235	15.8	6.9843663753
900	0.2347174542	22.8	9.8554513931
1600	0.2514162032	31	12.9262977712
2500	0.2513801687	38.5	15.8001327758

En la Tabla 4.5 observamos los datos de las redes obtenidas con el tercer escenario. En este modelo, además de recablear, incrementa el número de enlaces en la red. En este caso se obtuvieron coeficientes de agrupamiento muy bajos con respecto a los escenarios anteriores. La malla inicial tiene un coeficiente de agrupamiento igual a 0. Lo único que influye para aumentar el coeficiente de agrupamiento es el agregar nuevos enlaces. Un nodo puede llegar más allá de su vecindario inicial o el de sus vecinos por lo que, se vuelve difícil que se generen conjuntos de nodos completamente conectados entre

sí. Se puede observar que, el diámetro y la longitud promedio disminuyen gracias a los atajos de largo alcance que forman los enlaces extra.

Tabla 4.4: Resultados del segundo escenario

	Escenario 2		
Orden del grafo	**Coeficiente agrupamiento**	**Diámetro**	**LTP**
100	0.5217448186	2.2	1.9654594513
400	0.5335923862	3	1.9966659799
900	0.5198852309	5.25	2.0066866194
1600	0.510897505124	6	2.01015842435
2500	0.5290121117	6	2.0148677027

Tabla 4.5: Resultados del tercer escenario

	Escenario 3		
Tamaño red	**Coeficiente agrupamiento**	**Diámetro**	**LTP**
100	0.2457294914	5.6	2.7741818182
400	0.1685665948	7	3.6116215539
900	0.157353622	8	4.1226023977
1600	0.1551783897	8.8	4.5146297686
2500	0.1531514882	9.2	4.8131561104

Para mostrar mejor la dinámica de los datos durante la ejecución del simulador, se tomó un ejemplo de un red de 2500 nodos para los tres escenarios. En este ejemplo se puede observar como cambian las características de la red durante la simulación.

En la Figura 4.4 vemos como cambia el coeficiente de agrupamiento, siendo el segundo escenario el que consigue el valor más alto, lo que puede traducirse en un aumento en la tolerancia a fallos de la red por parte de este modelo. Mientras que con el tercer escenario, a pesar de tener un mayor numero de enlaces, se tiene el coeficiente de agrupamiento más pequeño.

De la Figura 4.5 observamos una disminución en el diámetro de la red,

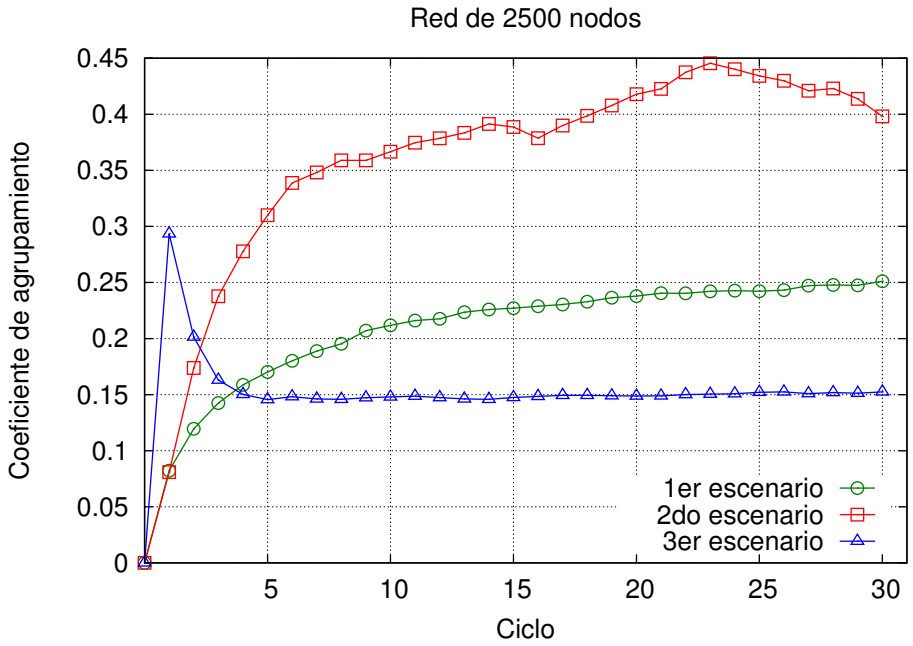


Figura 4.4: Coeficiente de agrupamiento por cada ciclo de una red de 2500 nodos en los 3 escenarios

el segundo escenario muestra una mayor disminución mientras que el primer escenario termina con un mayor diámetro. En todos los casos se observa una mayor disminución hacia los primeros ciclos de simulación, mientras que, en ciclos finales la disminución no es tan grande.

De la Figura 4.6 observamos que la longitud de trayectoria promedio tiene un comportamiento similar al del diámetro, recordemos que el diámetro por definición es consecuencia de las trayectorias existentes en la red. En estos resultados observamos que, los saltos que separan a todos los nodos de la red son pocos, en el caso del segundo escenario se obtienen las longitudes más cortas.

La distribución de grados se obtuvo de la red final que contiene todos los cambios realizados durante la simulación. En la Figura 4.7 vemos que en el primer escenario no existen concentradores muy grandes, pero si existen algunos concentradores pequeños que tienen entre 8 y 12 vecinos. La mayor parte de nodos tienen 4 enlaces.

De la Figura 4.8 observamos que existe un nodo concentrador que tiene

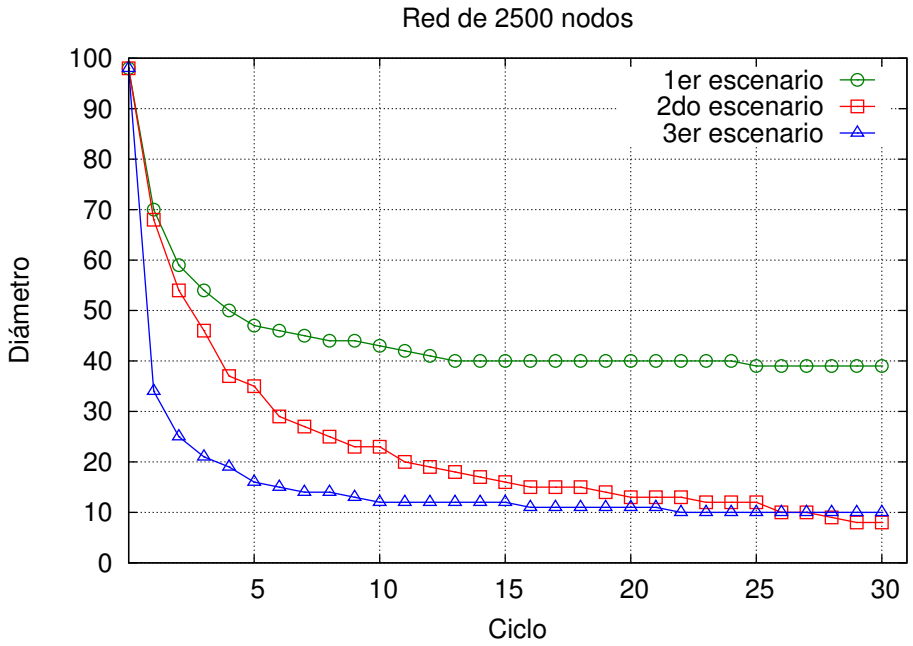


Figura 4.5: Diámetro por cada ciclo de una red de 2500 nodos en los 3 escenarios

2365 vecinos, esto explica la drástica disminución tanto en la longitud de trayectoria como del diámetro. El nodo concentrador permite una buena comunicación entre sus vecinos. La mayor parte de los nodos tienen entre 1 y 2 enlaces.

En la Figura 4.9 tenemos la distribución de grados del tercer escenario. En este caso no existen nodos de grado 0, 1, 2 y 3. Los nodos que tienen un grado 4 son los de las esquinas que tienen dos enlaces fijos y dos extra, esto en caso de no tener más vecinos que se conecten a ellos. En este caso los concentradores son más grandes que en el primer escenario y mucho menores que el tercer escenario. La mayor parte de los nodos tienen 6 enlaces lo que nos dice que estos nodos lograron colocar bien a sus dos enlaces extra, pero aun así no son concentradores.

Se midió también el número de enlaces recableados en los ejemplos anteriores, de 2500 nodos, el resultado se muestra en la Figura 4.10. En el tercer escenario se cuentan como enlaces recableados a los enlaces extra que se conectan al inicio de la simulación, por lo que el número de enlaces mostrado

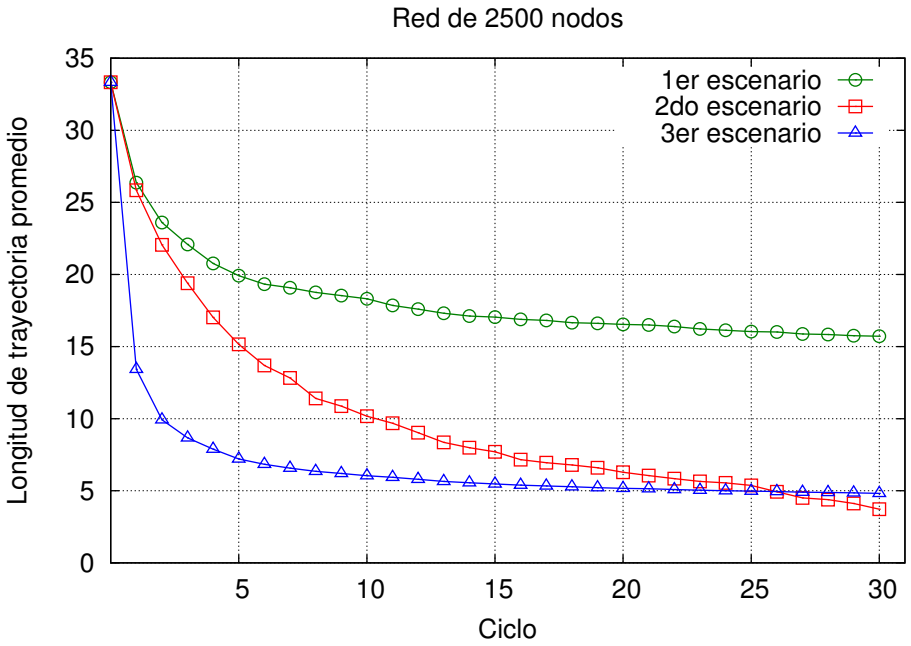


Figura 4.6: Longitud de trayectoria promedio por cada ciclo de una red de 2500 nodos en los 3 escenarios

en la gráfica en el primer ciclo es muy grande. A partir del tercer ciclo se generan solamente acciones de recableado. Después del cuarto ciclo se nota que el segundo escenario realiza por ciclo la mayor cantidad de acciones de recableado que los demás escenarios. Esto debido a que no tiene una restricción en cuanto el enlace, lo que ocasionó que estas simulaciones fueran las más tardadas. A pesar de tener las mejores condiciones de red, el segundo escenario es muy costoso en cuanto el envío de mensajes.

El primer escenario es el menos costoso, ya que un nodo sólo es capaz de recablear enlaces hacia nodos cercanos a él. El tercer escenario se encuentra entre ambos, aun así presenta unas condiciones no tan buenas con respecto al primero.

Con ayuda del módulo para la animación creamos la Figura 4.11 en donde, podemos observar un ejemplo de creación de un grafo de 900 nodos formado con el modelo propuesto en el primer escenario, enlaces con un alcance acotado. En estos casos encontramos la formación de muchos concentradores, que son los que permiten la disminución en las medidas

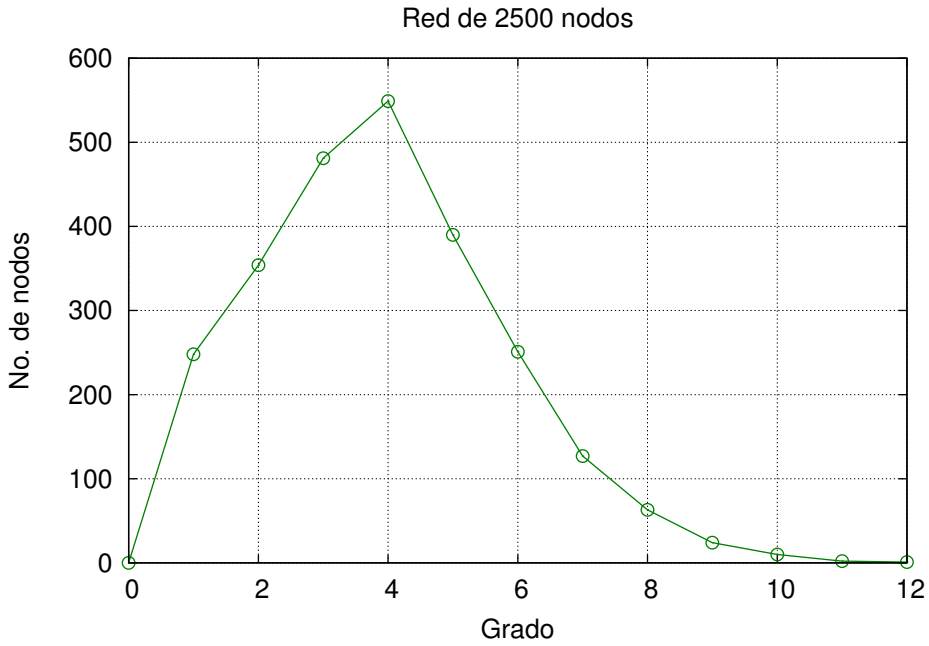
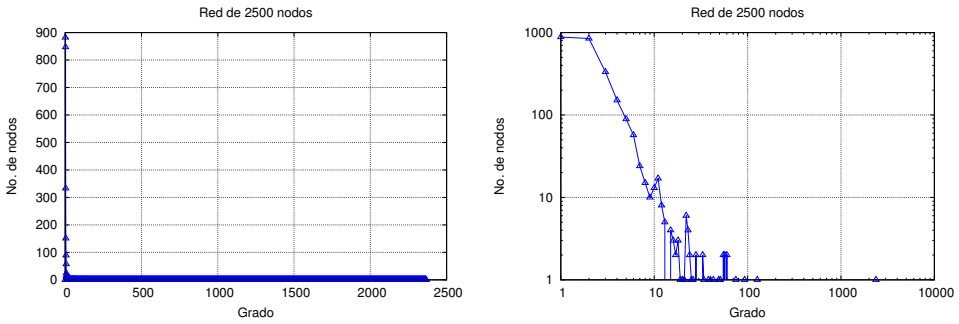


Figura 4.7: Ejemplo de la distribución de grados del primer escenario

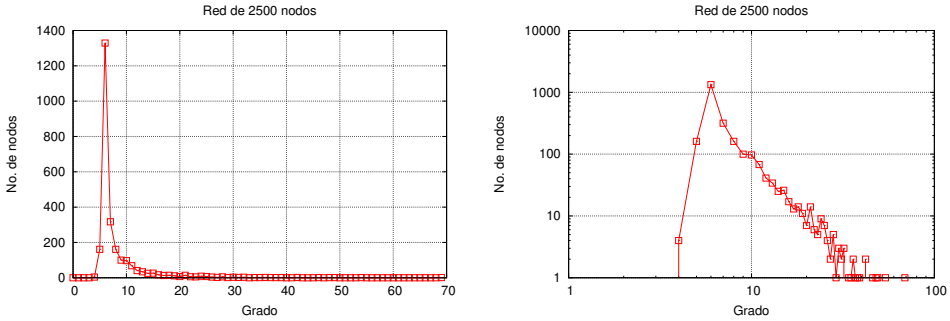


(a) Escala normal (b) Escala logarítmica

Figura 4.8: Ejemplo de la distribución de grados del segundo escenario

tomadas.

En la Figura 4.12 tenemos una red con la misma cantidad de nodos pero sometida al segundo escenario. En este caso podemos ver que el concentrador más rico se encuentra cerca del centro de la red. Vemos que se comienza a formar una estrella y algunos enlaces aislados.



(a) Escala normal (b) Escala logarítmica

Figura 4.9: Ejemplo de la distribución de grados del tercer escenario

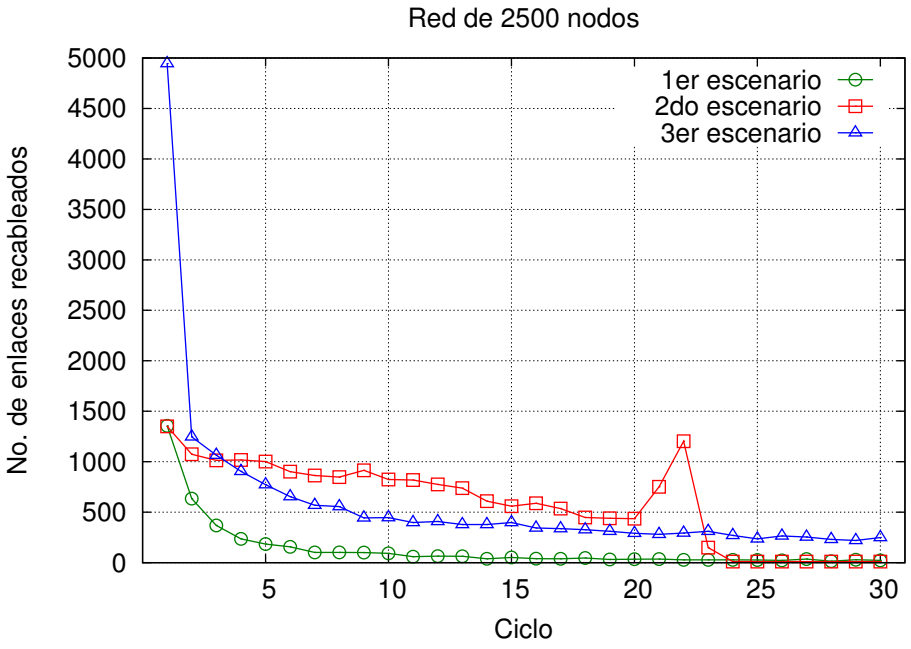


Figura 4.10: No. de enlaces recableados por ciclo en una red con 2500 nodos en los 3 escenarios

En la Figura 4.13, tomando en cuenta que cada nodo tiene enlaces fijos, se puede ver que no existen nodos aislados y aun puede observarse la malla. En el dibujo original no se podían apreciar los enlaces extras conectados por lo que se dibujo cada nodo en distinto tamaño, los nodos más grandes

cuentan con un mayor grado. Se observan algunos concentradores de menor grado que se comunican con el concentrados principal y ayudan a alcanzar a los nodos más alejados del centro que cuentan con un menor grado.

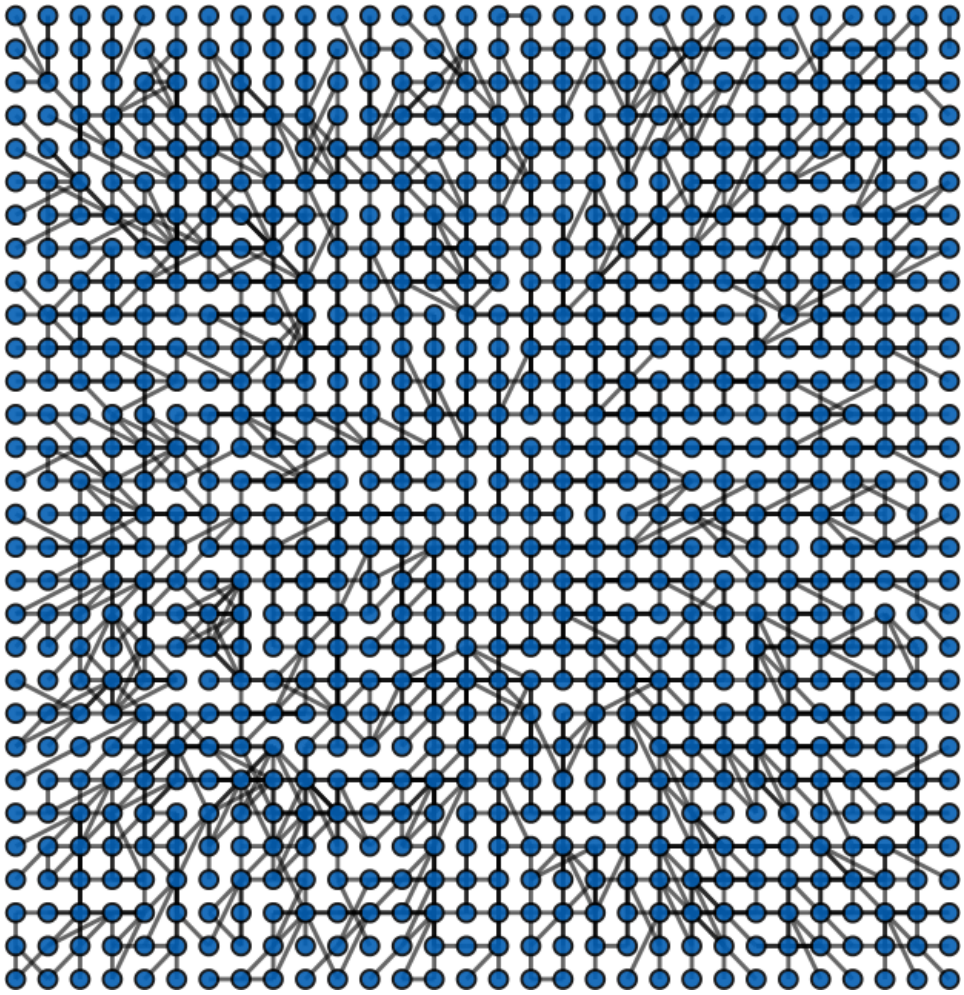


Figura 4.11: Ejemplo de una red de 900 nodos formada con el primer escenario

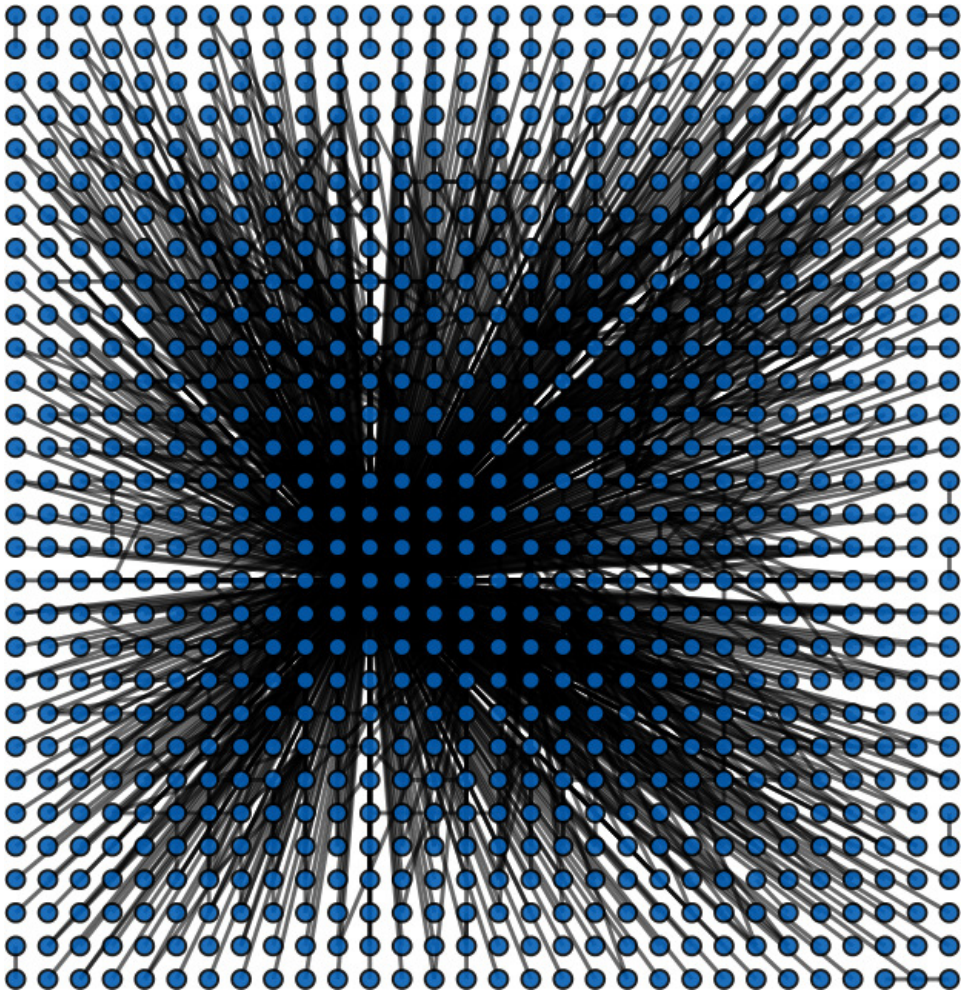


Figura 4.12: Ejemplo de una red de 900 nodos formada con el segundo escenario

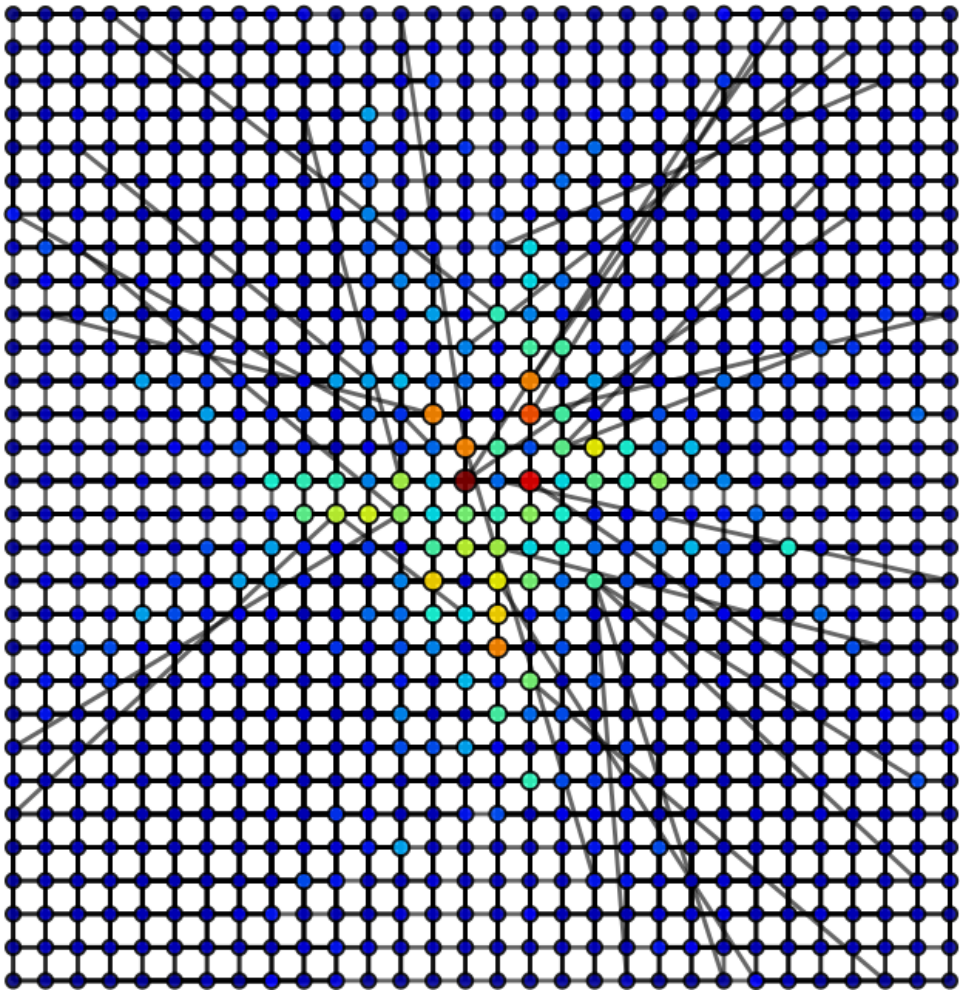


Figura 4.13: Ejemplo de una red de 900 nodos formada con el tercer escenario

Capítulo 5

Conclusión

En este trabajo, se presentó un modelo para la construcción de redes que permite estudiar su estructura de manera dinámica. Este modelo permite modificar distintos parámetros locales para observar las redes resultantes y sus propiedades. Los objetivos planteados en el trabajo se cumplieron de la siguiente manera.

El primer objetivo específico se cumplió, ya que el conjunto de parámetros tomado para las mediciones brinda una buena caracterización de las redes obtenidas. El coeficiente de agrupamiento, la longitud promedio de la trayectoria y el diámetro de la red, tienen cambios importantes dentro de los primeros ciclos de la simulación, por lo que en pruebas futuras se puede disminuir el número de ciclos. En los resultados se observa una disminución del diámetro y longitud promedio de la trayectoria, con el tiempo ambos parámetros se estabilizan. Para experimentos con nodos que tienen enlaces infinitos tanto el diámetro como la longitud de trayectoria promedio bajan drásticamente, por lo que el envío de paquetes se da en pocos pasos, sin embargo, este comportamiento no es conveniente pues en una red de comunicaciones el nodo central tendrá una carga muy alta de paquetes por lo que el flujo de paquetes no será eficiente. En el escenario 1 se puede ver que, con tamaños de enlaces cortos el grado de los nodos se mantiene controlado, es decir, a pesar de que existen concentradores estos no son demasiado grandes.

El segundo objetivo específico se cumplió ya que se propuso un mecanismo de recableado tomando en cuenta la utilidad de los nodos de la red para formar redes. Podemos observar que es posible crear diversas topologías de red a partir de la modificación de sencillas reglas locales. Una de las primeras topologías emergentes que observamos es la de estrella.

Los parámetros elegidos ayudaron a cumplir el tercer objetivo y con esto

se evaluaron las redes obtenidas. En los primeros dos escenarios se observó la formación de componentes aislados dentro de los grafos formados, a pesar de esto el componente mayor contiene a la mayor parte de los nodos por lo que se pueden medir las características de éste. En el último escenario se evitó la desconexión de la red, las medidas disminuyeron gracias al aumento de enlaces en la red, si bien se aumenta el número de enlaces es interesante poder disminuir el diámetro y longitud de trayectoria promedio en la red sin tener que llegar al extremo de conectar completamente a los nodos.

5.1. Trabajo futuro

Como trabajo futuro se puede obtener más información a partir de las matrices de adyacencia almacenadas. Una manera de utilizar las matrices de adyacencia es calculando la matriz laplaciana o laplaciano del grafo [26]. El laplaciano contiene información que nos da una idea de la topología del grafo, por ejemplo la conectividad de la red.

En el modelo propuesto la estructura inicial es una malla de dos dimensiones, en experimentos futuros se puede modificar la estructura inicial de la red para analizar el impacto que ésta tiene en los grafos finales. Se puede experimentar con topologías conocidas o con algunas estructuras inspiradas en la naturaleza, por ejemplo, en el trabajo de Nikko Ström [27] se realiza la conexión de redes neuronales inspirada en la organización de las neuronas del nervio auditivo, esta estructura es llamada tonotópica. En la estructura del nervio auditivo las neuronas se ordenan por su respuesta a altas o bajas frecuencias. Podemos dar a nuestros nodos distintas probabilidades de recableado y ordenarlos como en el caso de las neuronas para tener zonas con distintas probabilidades de reconectar enlaces y así poder observar si se tiene un impacto en la estructura final.

Además se puede experimentar con las distintas configuraciones de parámetros para descubrir otro conjunto de grafos.

De las simulaciones obtenidas se puede estudiar a fondo la tolerancia a fallos y ataques de cada tipo de red generado.

Podemos someter las redes a diversos mecanismos de degradación de la red para analizar su vulnerabilidad y estudiar que modelos son más convenientes para diversos sistemas de comunicaciones.

Es importante llevar a cabo un análisis profundo de las redes obtenidas, por lo que se pueden tomar en cuenta los trabajos de modularidad menciona-

dos en la Sección 2.1 para encontrar módulos dentro de los grafos existentes o de los que se obtengan en futuros experimentos.

Apéndice A

Ejemplos de grafos finales

En la Sección 4.4 se mostraron algunos ejemplos de grafos resultantes, en estos grafos podemos observar estructuras diferentes en cada escenario. En este apéndice se muestran más ejemplos de experimentos con los mismos parámetros pero con diferentes semillas. Todas las redes son de 900 nodos. Los nodos de menor tamaño tienen un grado de vértice menor mientras que los de gran tamaño tienen un grado mayor.

A.1. Primer escenario

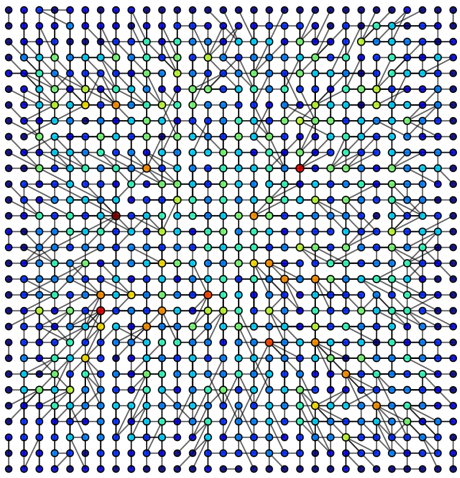


Figura A.1: Ejemplo 1 de una red formada con el 1er escenario

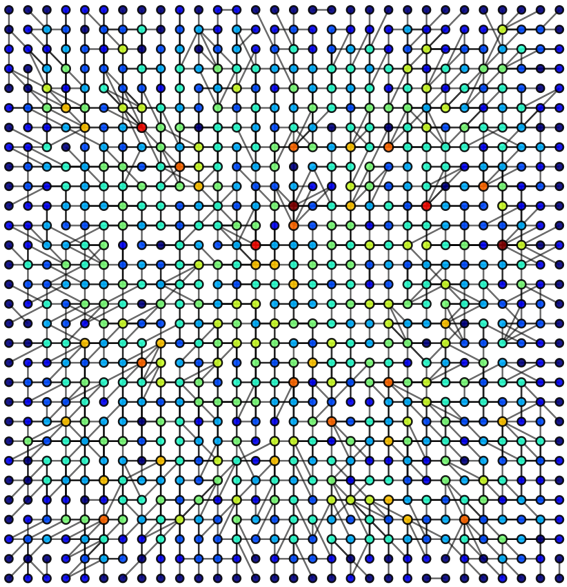


Figura A.2: Ejemplo 2 de una red formada con el 1er escenario

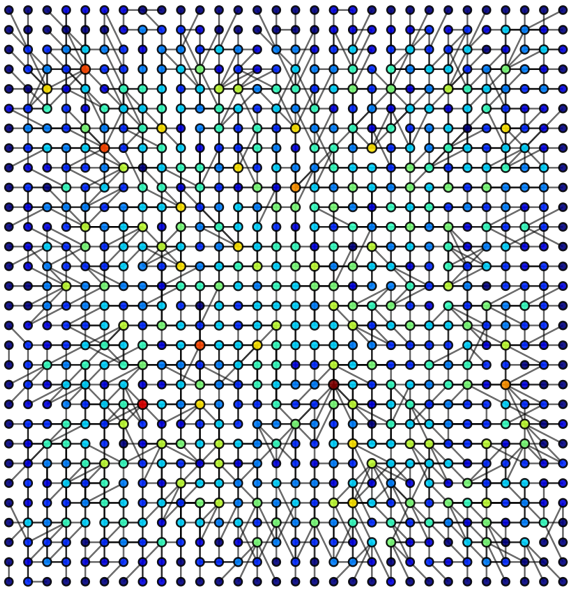


Figura A.3: Ejemplo 3 de una red formada con el 1er escenario

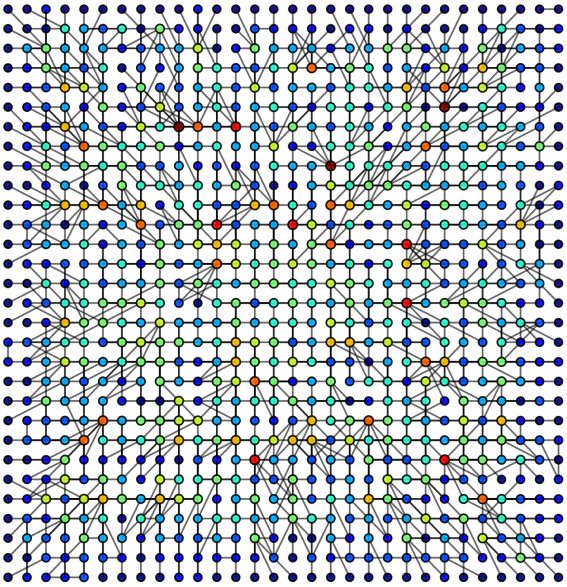


Figura A.4: Ejemplo 4 de una red formada con el 1er escenario

A.2. Segundo escenario

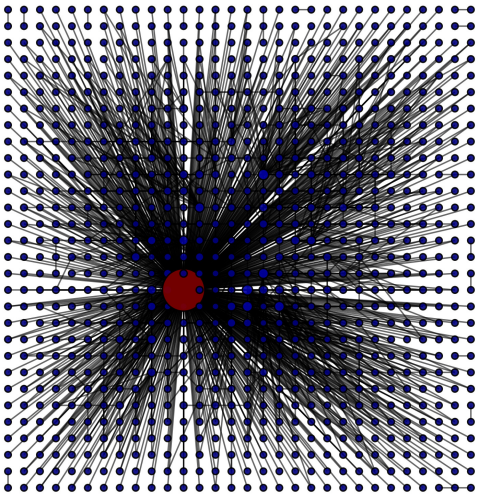


Figura A.5: Ejemplo 1 de una red formada con el 2do escenario

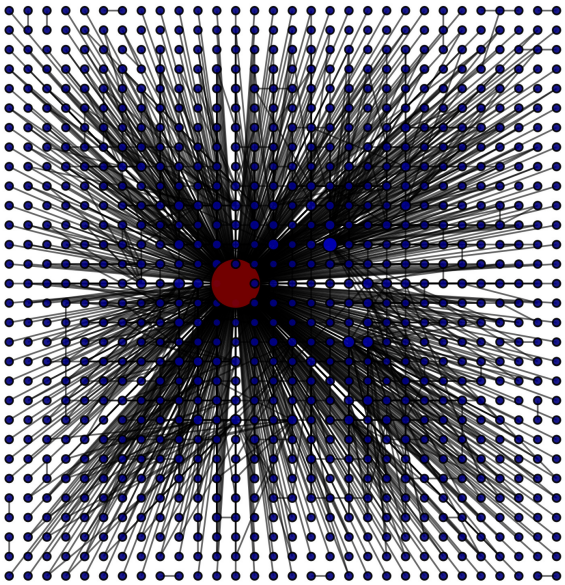


Figura A.6: Ejemplo 2 de una red formada con el 2do escenario

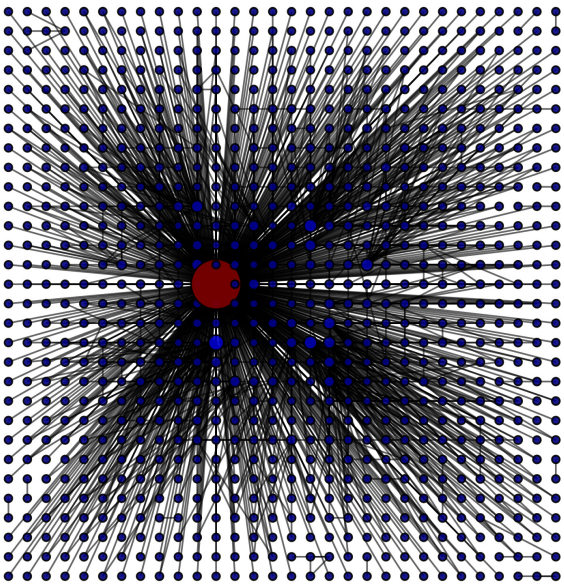


Figura A.7: Ejemplo 3 de una red formada con el 2do escenario

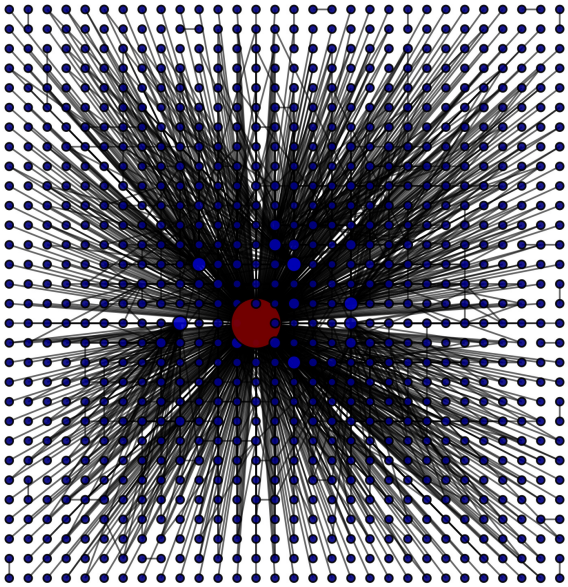


Figura A.8: Ejemplo 4 de una red formada con el 2do escenario

A.3. Tercer escenario

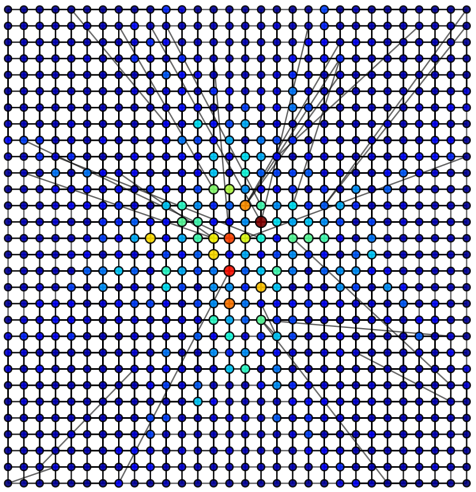


Figura A.9: Ejemplo de una red de 900 nodos formada con el 3er escenario

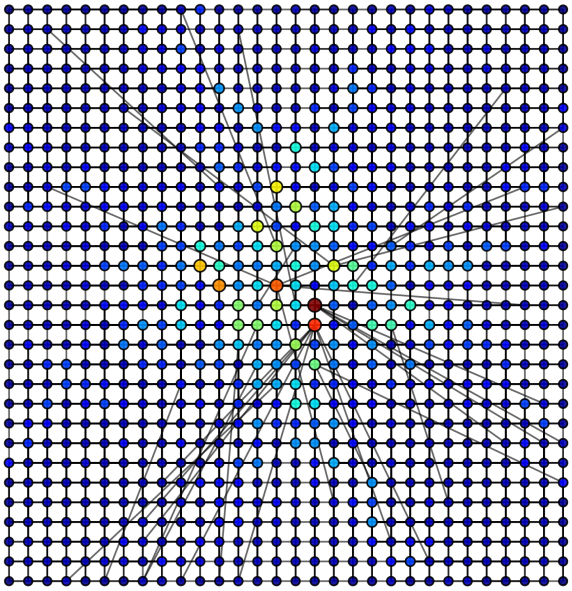


Figura A.10: Ejemplo de una red de 900 nodos formada con el 3er escenario

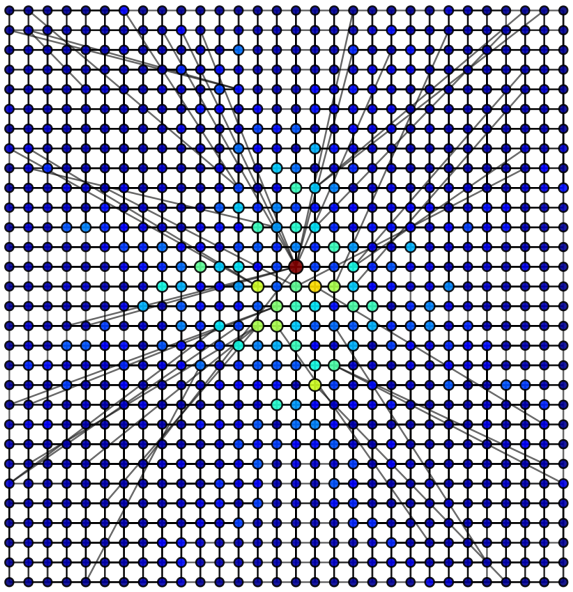


Figura A.11: Ejemplo 3 de una red de 900 nodos formada con el 3er escenario

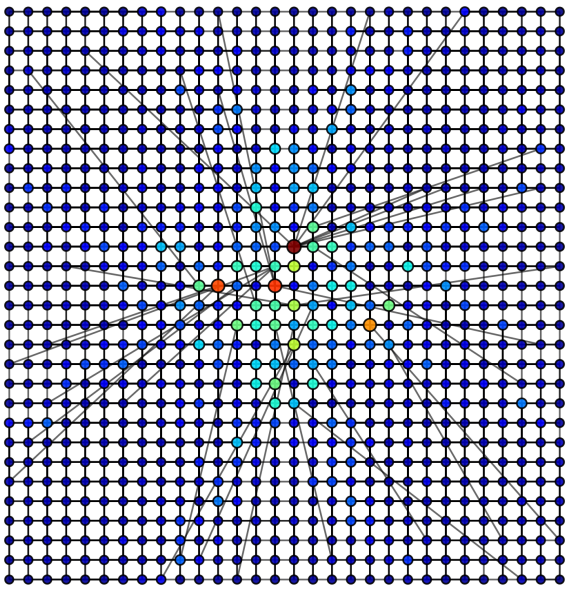


Figura A.12: Ejemplo 4 de una red de 900 nodos formada con el 3er escenario

Apéndice B

Simulador: Escenario 1 y 2

Con este archivo se crean los escenarios 1 y 2, lo que cambia entre ellos es el parámetro que especifica el tamaño de enlace, *tamEnlace*.

```
1
2  # Este archivo sirve de modelo para la creacion de
       aplicaciones, i.e. algoritmos concretos
3  """Implementa la simulacion del modelo de creacion de una
       red compleja"""
4  import sys
5  from event import Event
6  from model import Model
7  from simulation import Simulation
8  import parametros
9  import funciones
10
11 class simulacionRecableado(Model):
12   def init(self):
13     """ Aqui se definen e inicializan los atributos
           particulares del algoritmo (atributos de cada nodo)
           """
14     self.coord=funciones.coordenadas(self.id)
15     self.frec_enlaces = [0]*len(self.neighbors) ##Frecuencia
           de visita a nodos vecinos(utilidad enlaces)
16     self.frec_nodos = []       ##Frecuencia de visita a los
           demas nodos de la red
17     self.frec_nodos.append([])
18     self.frec_nodos.append([])
19     self.paqEnviados=0
20     self.paqRegreso=0
21     self.ciclo=1
22     self.contPaq=0       #Contador de los paquetes que debo
           esperar
23     self.terminoPaq=0    #Bandera que indica que termine mis
           paquetes y me pongo alerta a los PIF
```

```
24      self.terminoRec=0   #Bandera que indica que termine de
            recablear mis enlaces y me pongo alerta a los PIF
25      self.comRec=1     #Contador del PIF para comenzar el
            recableado
26      self.padreRec=self.id
27      self.visitadoR=False
28      self.visitadoRPI=False
29      self.comCiclo=1     #Contador del PIF para comenzar el
            nuevo ciclo
30      self.padreCiclo=self.id
31      self.contRecab=1      #Contador para saber en que paso
            estoy del recableado
32      self.visitadoC=False
33      self.visitadoCPI=False
34      self.etapaRec=0     #1 si estoy en etapa de recableado,
            0 en otro caso
35      self.rec=0        #Variable que indica si ya acabe y
            tengo que responder a a mi padre el REWIRE-PIF
36      self.cic=0        #Variable que indica si termine de
            recablear y puedo continuar el PIF
37      self.bufferRec=[]   #Buffer de mensajes para comenzar el
             recableo
38      self.neighborsSinc=self.neighbors[:]  #Conjunto de
            vecinos que ayudan en la sincronizacion
39      self.numRec=0     #numero de reconexiones
40      self.maxTray=0      #trayectoria maxima que siguieron
            mis paquetes
41      self.iDesc=-1     #En esta variable se guarda el id del
            nodo a quien envie un mensaje de desconexion
42
43    def limpiarListas(self):
44      del self.frec_enlaces[:]
45      del self.frec_nodos[:]
46      self.frec_enlaces = [0]*len(self.neighbors) ##Frecuencia
             de visita a nodos vecinos(utilidad enlaces)
47      self.frec_nodos = []    ##Frecuencia de visita a los
            demas nodos de la red
48      self.frec_nodos.append([])
49      self.frec_nodos.append([])
50
51    def recablea(self,event):
52      self.etapaRec=1
53      if len(self.bufferRec)>0:
54        for i in range(len(self.bufferRec)):
55          mBuffer=self.bufferRec[i]
56          if mBuffer[0][1] == "D":
57            if(len(self.neighbors)>1):
58              if mBuffer[0][2] in self.neighbors:
59                indVec=self.neighbors.index(mBuffer[0][2])
```

```
60                del self.neighbors[indVec]
61                del self.frec_enlaces[indVec]
62                if self.cic==1:
63                  self.comCiclo-=1
64
65              newevent = Event([["R-ACK","D",self.id,"SI",
                    mBuffer[0][3],mBuffer[0][4]],[]],event.
                    getTime() + 1.0,mBuffer[0][2],self.id)
66              self.transmit(newevent)
67            else:
68              newevent = Event([["R-ACK","D",self.id,"NO",
                    mBuffer[0][3],mBuffer[0][4]],[]],event.
                    getTime() + 1.0,mBuffer[0][2],self.id)
69              self.transmit(newevent)
70          if mBuffer[0][1] == "C":
71            if mBuffer[0][2] not in self.neighbors:
72              self.neighbors.append(mBuffer[0][2])
73              self.frec_enlaces.append(0)
74
75            newevent = Event([["R-ACK","C",self.id,"SI"],[]],
                  event.getTime() + 1.0,mBuffer[0][2],self.id)
76            self.transmit(newevent)
77        del self.bufferRec[:]
78
79      self.contRecab-=1
80      if self.frec_nodos[0]!=[] and self.frec_enlaces!=[]:
81        #Busco el 1er nodo mas visitado
82        indMax=self.frec_nodos[1].index(max(self.frec_nodos
              [1]))
83        max1=int(self.frec_nodos[0][indMax])
84        #Si existe mas de 1 elemento en la lista busco el 2do
              nodo mas visitado
85        if (len(self.frec_nodos[1])>1):
86          cpfrec=self.frec_nodos[1][:]
87          cpfrec[indMax]=0
88          indMax2=cpfrec.index(max(cpfrec))
89          max2=int(self.frec_nodos[0][indMax2])
90        #Busco el enlace menos usado
91        indMin=self.frec_enlaces.index(min(self.frec_enlaces))
92        if(int(self.frec_nodos[1][indMax])>parametros.frecNodo
              and max1 not in self.neighbors and funciones.
              distancia(self.id,max1)<=parametros.tamEnlace):
93          self.iDesc=self.neighbors[indMin]
94          newevent = Event([["R","D",self.id,"C",max1],[]],
                event.getTime()+1.0, self.neighbors[indMin],
                self.id)
95          self.transmit(newevent)
96          self.contRecab+=1
```

```
97      if (self.contRecab==0 and self.terminoRec==0):  #Si ya
            termine de recablear
98        self.terminoRec=1
99        #Si soy el coordinador comienzo el PIF para recablear
100       if self.id == parametros.coordinador:
101         newevent = Event([["CICLO","PIF",0],[]], event.
              getTime() + 1.0,self.id, self.id)
102         self.transmit(newevent)
103       if self.visitadoC==True and self.cic==0:
104         #Activo mi bandera para indicar que estoy dentro del
              PIF
105         self.cic=1
106         for n in self.neighbors:
107           if n != self.padreCiclo:
108             newevent = Event([["CICLO","PIF",0],[]], event.
                  getTime()+1.0,n, self.id)
109             self.transmit(newevent)
110             self.comCiclo += 1
111         if self.comCiclo == 0:
112           if (self.id != parametros.coordinador):
113             newevent = Event([["CICLO", "PIF", self.numRec],
                    []], event.getTime()+1.0, self.padreCiclo,
                  self.id)
114           else:
115             newevent = Event([["CICLO", "PI"], []], event.
                  getTime()+1.0, self.padreCiclo, self.id)
116           self.transmit(newevent)
117
118   def receive(self, event):
119     M=event.getName() # Recibo el mensaje
120     if M[0][0] == "START":   ##Comienzo el envio de un
          paquete.
121       self.contPaq=self.contPaq+1
122       new_package=funciones.newPackage(self.id)
123       newM=[new_package[:],[]]
124       next = funciones.Compass_Routing(self.id, self.
            neighbors, new_package[2], new_package[5:])
125       newevent = Event(newM, event.getTime()+1.0, next, self
            .id)
126       self.transmit(newevent)
127       self.paqEnviados=self.paqEnviados+1
128
129     if M[0][0] == "P":
130       if (M[0][2] != self.id):
131         auxN=self.neighbors[:]
132         auxN.remove(event.getSource())
133         for n in M[1]:
134           if n in auxN:
135             auxN.remove(n)
```

```
136           ruta =M[0][5:]
137           for m in ruta:
138             if m in auxN:
139               auxN.remove(m)
140           #Comenzando backtraking
141           if len(auxN) == 0:  #Si la lista queda vacia
                  entonces me agrego a la lista de nodos que no
                  llevan al destino
142             M[1].append(self.id)
143             #Envio el paquete regreso a la fuente(o mi padre)
144             B=[M[0][:],M[1][:]]
145             B[0][0]="B"
146             newevent = Event(B, event.getTime()+1.0, event.
                    getSource(), self.id)
147             self.transmit(newevent)
148           else:
149             self.contPaq=self.contPaq+1
150             next = funciones.Compass_Routing(self.id, auxN, M
                    [0][2], ruta)
151             M[0].append(self.id)
152             newevent = Event(M, event.getTime()+1.0, next,
                    self.id)
153             self.transmit(newevent)
154         else:
155           #Envio el paquete de vuelta al nodo origen
156           newM=[M[0][:],M[1][:]]
157           newM[0].append(self.id)
158           ruta = newM[0][5:]
159           newM[0][0]="ACK"
160           newevent = Event(newM, event.getTime() + 1.0, ruta[
                  len(ruta)-2], self.id)
161           self.transmit(newevent)
162       if M[0][0] == "B": #Backtrack
163         auxN=self.neighbors[:]  #Creo una copia de mi lista de
                vecinos
164         if self.id != M[0][1]:
165           if M[0][len(M[0])-1] in auxN:
166             auxN.remove(M[0][len(M[0])-1])  #Borro de la lista
                    a mi "padre"
167
168         for n in M[1]:  #Descarto de la copia a todos los
                vecinos por los que ya paso el paquete
169           if n in auxN:
170             auxN.remove(n)
171         ruta =M[0][5:]
172         for m in ruta:
173           if m in auxN:
174             auxN.remove(m)
```

```
175         if len(auxN) == 0:  #Si la lista queda vacia entonces
                me agrego a la lista de nodos que no llevan al
                destino
176           self.contPaq=self.contPaq-1
177           if self.id == M[0][1]:
178             self.paqRegreso=self.paqRegreso+1
179             if(self.paqRegreso  < parametros.num_paquetes):
180               newevent = Event([["START", self.id], []], event
                      .getTime() + 1.0, self.id, self.id)
181               self.transmit(newevent)
182           else:
183             if self.id==M[0][len(M[0])-1]:
184               del M[0][len(M[0])-1]
185             M[1].append(self.id)
186             #Envio el paquete regreso a la fuente(o mi padre)
187             newevent = Event(M, event.getTime() + 1.0,M[0][len
                    (M[0])-1],self.id)
188             self.transmit(newevent)
189           # Verifico si ya termine de enviar mis paquetes
190           if (self.contPaq==0 and self.paqRegreso ==
                 parametros.num_paquetes and self.terminoPaq==0):
191             self.terminoPaq=1 #Indico que termine de la etapa
                    de envio de paquetes
192             #Si soy el coordinador comienzo el PIF para
                    recablear
193             if (self.id==parametros.coordinador):
194               newevent = Event([["REWIRE","PIF",0],[]], event.
                      getTime() + 1.0,self.id, self.id)
195               self.transmit(newevent)
196             if (self.visitadoR==True and self.rec==0):
197               self.rec = 1
198               for n in self.neighborsSinc:
199                 if n != self.padreRec:
200                   newevent = Event([["REWIRE","PIF",0],[]],
                          event.getTime()+1.0,n, self.id)
201                   self.transmit(newevent)
202                   self.comRec += 1
203               if self.comRec == 0:
204                 if (self.id != parametros.coordinador):
205                   newevent = Event([["REWIRE", "PIF", self.
                          maxTray], []], event.getTime()+1.0, self
                          .padreRec, self.id)
206                 else:
207                   newevent = Event([["REWIRE", "PI"],[]],
                          event.getTime()+1.0, self.padreRec, self
                          .id)
208                 self.transmit(newevent)
209       else:
210         P=[M[0][:],M[1][:]]
```

```
211             P[0][0]="P"
212             next = funciones.Compass_Routing(self.id, auxN, M
                    [0][2], ruta)
213             newevent = Event(P, event.getTime() + 1.0, next,
                    self.id)
214             self.transmit(newevent)
215
216         if M[0][0] == "ACK":
217           self.contPaq -= 1
218           if((M[0][1]) == self.id):
219             self.paqRegreso = self.paqRegreso+1
220             ruta = M[0][6:]
221             if len(ruta) > self.maxTray:
222               self.maxTray = len(ruta)
223             k=self.neighbors.index(ruta[0])
224             self.frec_enlaces[k] = self.frec_enlaces[k] + 1
225             for i in range(1,len(ruta)):
226               if (ruta[i] in self.frec_nodos[0]):
227                 j=self.frec_nodos[0].index(ruta[i])
228                 self.frec_nodos[1][j]=self.frec_nodos[1][j]+1
229               else:
230                 self.frec_nodos[0].append(ruta[i])
231                 self.frec_nodos[1].append(1)
232
233             if(self.paqRegreso < parametros.num_paquetes):
234               newevent = Event([["START",self.id],[]], event.
                      getTime() + 1.0,self.id,self.id)
235               self.transmit(newevent)
236           else:
237             ruta = M[0][5:]
238             indice=ruta.index(self.id)
239             newevent = Event(M, event.getTime() + 1.0, ruta[
                    indice-1], self.id)
240             self.transmit(newevent)
241           if (self.contPaq==0 and self.paqRegreso == parametros.
                  num_paquetes and self.terminoPaq==0):
242             self.terminoPaq=1 #Termine el envio de paquetes
243             #Si soy el coordinador comienzo el PIF para
                    recablear
244             if (self.id==parametros.coordinador):
245               newevent = Event([["REWIRE", "PIF", 0], []], event
                      .getTime() + 1.0,self.id, self.id)
246               self.transmit(newevent)
247             if (self.visitadoR==True and self.rec==0):
248               self.rec=1  #Entro a la etapa de recableado
249               for n in self.neighborsSinc:
250                 if n != self.padreRec:
251                   newevent = Event([["REWIRE","PIF",0],[]],
                          event.getTime()+1.0,n, self.id)
```

```
252                 self.transmit(newevent)
253                 self.comRec += 1
254             if self.comRec == 0:
255               if (self.id != parametros.coordinador):
256                 newevent = Event([["REWIRE", "PIF", self.
                        maxTray], []], event.getTime()+1.0, self.
                        padreRec, self.id)
257               else:
258                 newevent = Event([["REWIRE", "PI"], []], event
                        .getTime()+1.0, self.padreRec, self.id)
259               self.transmit(newevent)
260
261       """Sincronizacion para comenzar la etapa de recableado
              """
262       if M[0][0] == "REWIRE":
263         if M[0][1] == "PIF":
264           self.comRec -= 1
265           if self.visitadoR == False:
266             self.visitadoR = True
267             self.padreRec = event.getSource()
268             if (self.paqRegreso == parametros.num_paquetes and
                     self.terminoPaq == 1):
269               self.rec = 1
270               for n in self.neighborsSinc:
271                 if n != self.padreRec:
272                   newevent = Event([["REWIRE", "PIF", 0], []],
                          event.getTime()+1.0, n, self.id)
273                   self.transmit(newevent)
274                   self.comRec += 1
275           if event.getSource()!=self.padreCiclo:  #Sumatoria
                de cambios
276             if M[0][2]>self.maxTray:
277               self.maxTray=M[0][2]
278           if self.comRec == 0 and self.terminoPaq==1:
279             if (self.id != parametros.coordinador):
280               newevent = Event([["REWIRE", "PIF", self.maxTray
                      ], []], event.getTime()+1.0, self.padreRec,
                      self.id)
281             else:
282               newevent = Event([["REWIRE", "PI"], []], event.
                      getTime()+1.0, self.padreRec, self.id)
283             self.transmit(newevent)
284
285         if M[0][1]=="PI":
286           if self.visitadoRPI == False:
287             self.visitadoRPI = True
288             self.visitadoC = False
289             self.visitadoCPI = False
290             self.comCiclo = 1
```

```
291             self.cic = 0
292             #Comienzo el recableado
293             for n in self.neighborsSinc:
294               newevent = Event([["REWIRE", "PI"], []], event.
                      getTime()+1.0, n, self.id)
295               self.transmit(newevent)
296             self.recablea(event)
297
298       """Sincronizacion para comenzar el siguiente ciclo"""
299       if M[0][0] == "CICLO":
300         if M[0][1] == "PIF":
301           self.comCiclo -= 1
302           if self.visitadoC == False:
303             self.visitadoC = True
304             self.contPaq = 0
305             self.limpiarListas()
306             self.paqEnviados = 0
307             self.paqRegreso = 0
308             self.padreCiclo = event.getSource()
309             if (self.contRecab == 0 and self.terminoRec == 1):
310               self.cic = 1    #Estoy dentro del PIF
311               for n in self.neighborsSinc:
312                 if n != self.padreCiclo:
313                   newevent = Event([["CICLO", "PIF", 0],[]],
                          event.getTime()+1.0, n, self.id)
314                   self.transmit(newevent)
315                   self.comCiclo += 1
316           if event.getSource()!=self.padreCiclo:  #Realizamos
                la sumatoria de el numero de cambios
317             self.numRec=self.numRec+M[0][2]
318           if (self.comCiclo == 0 and self.terminoRec==1):
319             if (self.id != parametros.coordinador):
320               newevent = Event([["CICLO", "PIF",  self.numRec
                      ], []], event.getTime()+1.0, self.padreCiclo
                      , self.id)
321             else:
322               newevent = Event([["CICLO", "PI"], []], event.
                      getTime()+1.0, self.padreCiclo, self.id)
323             self.transmit(newevent)
324
325         if M[0][1]=="PI":
326           if self.visitadoCPI == False:
327             #Aqui el coordinador debe revisar la variable self
                    .numRec y ejecutar la condicion de paro
328             if self.ciclo<parametros.ciclos:
329               self.visitadoCPI = True
330               self.rec = 0
331               self.visitadoR = False
332               self.visitadoRPI = False
```

```
333             self.etapaRec = 0
334             self.contRecab = 1
335             self.comRec = 1
336             self.terminoPaq = 0
337             self.terminoRec = 0
338             self.numRec = 0
339             self.maxTray = 0
340             self.iDesc = -1
341             self.ciclo += 1
342             for n in self.neighborsSinc:
343               newevent = Event([["CICLO", "PI"], []], event.
                      getTime()+1.0, n, self.id)
344               self.transmit(newevent)
345             newevent = Event([["START",self.id],[]], event.
                    getTime() + 1.0,self.id,self.id)
346             self.transmit(newevent)
347
348     if M[0][0] == "R":
349       if(self.etapaRec==0):
350         self.bufferRec.append(M)
351       else:
352         if M[0][1] == "D":
353           if(len(self.neighbors)>1 and (event.getSource()!=
                  self.iDesc or  (event.getSource()==self.iDesc
                  and event.getSource()<self.id))):
354             if event.getSource() in self.neighbors:
355               indVec = self.neighbors.index(event.getSource
                      ())
356               del self.neighbors[indVec]
357               del self.frec_enlaces[indVec]
358             newevent = Event([["R-ACK", "D",self.id, "SI", M
                    [0][3], M[0][4]], []],event.getTime()+1.0,
                    event.getSource(), self.id)
359             self.transmit(newevent)
360           else:
361             newevent = Event([["R-ACK", "D", self.id, "NO",
                    M[0][3],M[0][4]], []], event.getTime()+1.0,
                    event.getSource(), self.id)
362             self.transmit(newevent)
363         elif M[0][1]== "C":
364           if event.getSource() not in self.neighbors:
365             self.neighbors.append(event.getSource())
366             self.frec_enlaces.append(0)
367           newevent = Event([["R-ACK", "C", self.id, "SI"
                  ],[]], event.getTime()+ 1.0, event.getSource()
                  , self.id)
368           self.transmit(newevent)
369
370     if M[0][0] == "R-ACK":
```

```
371         self.contRecab -= 1
372         if M[0][1] == "D" and M[0][3] == "SI":
373           if event.getSource() in self.neighbors:
374             indVec=self.neighbors.index(event.getSource())
375             del self.neighbors[indVec]
376             del self.frec_enlaces[indVec]
377           newevent = Event([["R", "C", self.id],[]], event.
                getTime()+1.0, M[0][5], self.id)
378           self.transmit(newevent)
379           self.contRecab += 1
380           self.numRec += 1
381         if M[0][1] == "C" and M[0][3]=="SI":
382           if event.getSource() not in self.neighbors:
383             self.neighbors.append(event.getSource())
384             self.frec_enlaces.append(0)
385         if (self.contRecab == 0 and self.terminoRec == 0):
386           self.terminoRec = 1
387           if(self.id==parametros.coordinador):
388             #Comienza el PIF
389             newevent = Event([["CICLO", "PIF", 0], []], event.
                  getTime() + 1.0, self.id, self.id)
390             self.transmit(newevent)
391           if (self.visitadoC==True and self.cic==0):
392             self.cic=1 #Estoy dentro de PIF
393             for n in self.neighborsSinc:
394               if n != self.padreCiclo:
395                 newevent = Event([["CICLO", "PIF", 0],[]],
                      event.getTime()+1.0, n, self.id)
396                 self.transmit(newevent)
397                 self.comCiclo += 1
398             if self.comCiclo == 0:
399               if (self.id != parametros.coordinador):
400                 newevent = Event([["CICLO", "PIF", self.numRec
                      ], []], event.getTime()+1.0, self.
                      padreCiclo, self.id)
401               else:
402                 newevent = Event([["CICLO", "PI"], []], event.
                      getTime()+1.0, self.padreCiclo, self.id)
403               self.transmit(newevent)
404 #
      ------------------------------------------------------------

405 # "main()"
406 #
      ------------------------------------------------------------

407 # construye una instancia de la clase Simulation recibiendo
      como parametros el nombre del
```

```
408 # archivo que codifica la lista de adyacencias de la grafica
        y el tiempo max. de simulacion
409 if len(sys.argv) != 2:
410     print "Please␣supply␣a␣file␣name"
411     raise SystemExit(1)
412 experiment = Simulation(sys.argv[1], sys.maxint)
413
414 # asocia un pareja proceso/modelo con cada nodo de la
        grafica
415 for i in range(1,len(experiment.graph)+1):
416     m = simulacionRecableado()
417     experiment.setModel(m, i)
418
419 # inserta un evento semilla en la agenda y arranca
420 seed=[]
421 for i in range(1,len(experiment.graph)+1):
422     seed.append(Event([["START",i],[]], 0.0, i,i))
423     experiment.init(seed[i-1])
424
425 experiment.run()
```

Apéndice C

Simulador: Escenario 3

Con este archivo se crea el escenario 3, lo que cambia con respecto al archivo anterior es el manejo de enlaces fijos, el recableado se da únicamente entre enlaces extra.

```
1  # This Python file uses the following encoding: utf-8
2  # Este archivo sirve de modelo para la creacion de
       aplicaciones, i.e. algoritmos concretos
3  # Autora: Magali Alexander Lopez Chavira
4  # Abril-2014
5  """ Implementa la simulacion del modelo de creacion de una
       red compleja con recableado,
6  usando vecinos permanentes para evitar la desconexion de la
       red"""
7  import sys
8  from event import Event
9  from model import Model
10 from simulation import Simulation
11 import parametros
12 import funciones
13 class simulacionRecableado(Model):
14   def init(self):
15     self.frec_enlaces = [0]*len(self.neighbors) #fe
16     self.frec_enlacesExtra=[]
17     self.frec_nodos = [[],[]] # fn
18     self.neighborsPerm=self.neighbors[:]  #Vecinos
           permanentes
19     self.extrasLibres=parametros.nEnlExt  #Enlaces extra
           ocupados
20     self.idSolicitudEnlExtra=[]   #aqui almaceno el id de
           los nodos que son candidatos a ser mis vecinos por
           enlace extra y que aun no me contestan la peticion
21     self.auxSolEnv=0      #Auxiliar para contar el numero de
            solicitudes que he enviado para mis enlaces extra
```

```
22      self.idEnlaceExtra=[] #Identificador del nodo al que
            tengo conectado un enlace extra
23      self.paqEnviados=0
24      self.paqRegreso=0
25      self.ciclo=1
26      self.contPaq=0      #Contador de los paquetes que debo
            esperar
27      self.terminoPaq=0   #Bandera que indica que termine mis
            paquetes y me pongo alerta a los PIF
28      self.terminoRec=0   #Bandera que indica que termine de
            recablear mis enlaces y me pongo alerta a los PIF
29      self.comRec=1       #Contador del PIF para comenzar el
            recableado
30      self.padreRec=self.id
31      self.visitadoR=False
32      self.visitadoRPI=False
33      self.comCiclo=1     #Contador del PIF para comenzar el
            nuevo ciclo
34      self.padreCiclo=self.id
35      self.contRecab=1      #Contador para saber en que paso
            estoy del recableado
36      self.visitadoC=False
37      self.visitadoCPI=False
38      self.etapaRec=0     #1 si estoy en etapa de recableado,
            0 en otro caso
39      self.rec=0          #Indica si ya acabe y tengo que
            responder a a mi padre el REWIRE-PIF
40      self.cic=0          #Variable que indica si termine de
            recablear y puedo continuar el PIF
41      self.bufferRec=[]   #Buffer de mensajes para comenzar el
             recableo
42      self.numRec=0       #numero de reconexiones
43      self.maxTray=0      #trayectoria maxima que siguieron
            mis paquetes
44      self.iDesc=-1       #Guardo el id del nodo a quien envie
            un mensaje de desconexion// es igual a -1 cuando no
            he enviado una peticion de desconexion
45      self.iConex=-1      #Guardo el id del nodo a quien me
            conectare en un recableado// es igual a -1 cuando no
             he enviado una peticion de conexion en recableado
46      self.idsDesc=[]     #Aqui guardo los ids de quien me
            manda peticion de desconexion
47      self.pifsCiclo=[]   #ids de los nodos que me enviaron
            pifs para sincronizar ciclos
48
49    def limpiarListas(self):
50      del self.frec_enlaces[:]
51      del self.frec_nodos[:]
```

```
52      self.frec_enlaces = [0]*len(self.neighbors) ##Frecuencia
            de visita a nodos vecinos(utilidad enlaces)
53      self.frec_nodos = [[],[]] ##Frecuencia de visita a los
            demas nodos de la red
54      del self.frec_enlacesExtra[:]
55      self.frec_enlacesExtra=[]
56      del self.idsDesc[:]
57      self.idsDesc=[]
58      del self.idSolicitudEnlExtra[:]
59      self.idSolicitudEnlExtra=[]
60
61    def recablea(self,event):
62      self.etapaRec=1        #Estoy dentro del recableado
63      if len(self.bufferRec)>0:
64        for i in range(len(self.bufferRec)):
65          mBuffer=self.bufferRec[i]
66          if mBuffer[0][1] == "D":
67            if (mBuffer[0][2] in self.neighbors) and (mBuffer
                  [0][2] not in (self.idEnlaceExtra)) and (
                  mBuffer[0][2] not in (self.neighborsPerm)):
68              self.idsDesc.append(mBuffer[0][2])
69              newevent = Event([["R-ACK", "D", self.id, "SI",
                    mBuffer[0][3], mBuffer[0][4], "enLista"],
                    []], event.getTime() + 1.0, mBuffer[0][2],
                    self.id)
70              self.transmit(newevent)
71            else:
72              newevent = Event([["R-ACK", "D", self.id, "NO",
                    mBuffer[0][3], mBuffer[0][4], 0], []], event
                    .getTime() + 1.0, mBuffer[0][2], self.id)
73              self.transmit(newevent)
74          elif mBuffer[0][1] == "C":
75            if ((mBuffer[0][2] not in self.neighbors) and ((
                  mBuffer[0][2] not in self.idSolicitudEnlExtra)
                   or ((mBuffer[0][2] in self.
                  idSolicitudEnlExtra) and  (mBuffer[0][2]<self.
                  id)))):
76              self.neighbors.append(mBuffer[0][2])
77              self.frec_enlaces.append(0)
78              if (mBuffer[0][2] in self.idSolicitudEnlExtra)
                    and (mBuffer[0][2]<self.id):
79                #elimino el id del nodo de mis solicitudes
                      enviadas
80                self.idSolicitudEnlExtra.remove( mBuffer[0][2]
                      )
81              newevent = Event([["R-ACK", "C", self.id, "SI"
                    ],[]], event.getTime()+ 1.0, mBuffer[0][2],
                    self.id)
82              self.transmit(newevent)
```

```
83            else:
84              newevent = Event([["R-ACK", "C", self.id, "NO"
                    ],[]], event.getTime()+ 1.0, mBuffer[0][2],
                    self.id)
85              self.transmit(newevent)
86        del self.bufferRec[:]
87
88      self.contRecab -= 1
89      #Si tengo enlaces extra sin usarlos conecto al nodo mas
            visitado en la ronda de envio de paquetes
90      if (self.extrasLibres > 0):
91        # Ordeno mi frecuencia de nodos de forma descendente
92        funciones.burbujaFrec_Nodos(self.frec_nodos)
93        while ((self.extrasLibres > self.auxSolEnv) and (len(
              self.frec_nodos[0])>=self.auxSolEnv)):
94          if((self.frec_nodos[0][self.auxSolEnv] not in self.
                neighbors) and (funciones.distancia(self.id,self
                .frec_nodos[0][self.auxSolEnv])<=parametros.
                tamEnlace)):
95            #agrego el id del nodo a la lista de solicitudes
                  enviadas
96            self.idSolicitudEnlExtra.append( self.frec_nodos
                  [0][self.auxSolEnv] )
97            #Envio peticion de conexion
98            newevent = Event([["R", "C", self.id], []], event.
                  getTime()+1.0, self.frec_nodos[0][self.
                  auxSolEnv], self.id)
99            self.transmit(newevent)
100           if self.contRecab == 0:
101             self.contRecab += 1
102         self.auxSolEnv+=1
103     else:
104       #Busco el nodo mas visitado
105       if self.frec_nodos[0]!=[] and self.frec_enlaces!=[]:
106         #Busco el 1er nodo mas visitado
107         indMax = self.frec_nodos[1].index(max(self.
                frec_nodos[1]))
108         max1=int(self.frec_nodos[0][indMax])
109       #Busco el enlace menos usado
110       for x in range(len(self.idEnlaceExtra)):
111         indX = self.neighbors.index(self.idEnlaceExtra[x])
112         self.frec_enlacesExtra.append(self.frec_enlaces[indX
                ])
113       indMin = self.frec_enlacesExtra.index(min(self.
              frec_enlacesExtra))
114       if((int(self.frec_nodos[1][indMax])>=parametros.
              frecNodo) and (max1 not in self.neighbors) and (
              funciones.distancia(self.id,max1)<=parametros.
              tamEnlace)):
```

```
115             self.iDesc = self.idEnlaceExtra[indMin]
116             self.iConex = max1
117             newevent = Event([["R","D", self.id,"C",max1],[]],
                    event.getTime()+1.0, self.iDesc, self.id)
118             self.transmit(newevent)
119             self.contRecab += 1
120         if (self.contRecab==0 and self.terminoRec==0):  #Si ya
                termine de recablear
121           self.terminoRec = 1
122           if self.id == parametros.coordinador:   #Si soy el
                coordinador comienzo el PIF para recablear
123             newevent = Event([["CICLO","PIF",0],[]], event.
                    getTime() + 1.0,self.id, self.id)
124             self.transmit(newevent)
125           if self.visitadoC==True and self.cic==0:
126             self.cic=1  #Estoy dentro del PIF
127             for n in self.neighbors:
128               if n != self.padreCiclo:
129                 newevent = Event([["CICLO","PIF",0],[]], event.
                        getTime()+1.0,n, self.id)
130                 self.transmit(newevent)
131                 self.comCiclo += 1
132             if self.comCiclo == 0:
133               if (self.id != parametros.coordinador):
134                 newevent = Event([["CICLO","PIF",self.numRec
                        ],[]], event.getTime()+1.0, self.padreCiclo,
                         self.id)
135               else:
136                 newevent = Event([["CICLO","PI"],[]], event.
                        getTime()+1.0, self.padreCiclo, self.id)
137               self.transmit(newevent)
138
139     def receive(self, event):
140       """ Aqui se definen las acciones concretas que deben
              ejecutarse cuando se recibe un evento """
141       M=event.getName()    # Recibo el mensaje
142       if M[0][0] == "START":  #Comienzo el envio de un paquete
143         self.contPaq=self.contPaq+1
144         new_package=funciones.newPackage(self.id)
145         newM=[new_package[:],[]]
146         next = funciones.Compass_Routing(self.id, self.
                neighbors, new_package[2], new_package[5:])
147         newevent = Event(newM, event.getTime() + 1.0, next,
                self.id)
148         self.transmit(newevent)
149         self.paqEnviados=self.paqEnviados+1
150
151       if M[0][0] == "P":
152         if (M[0][2] != self.id):
```

```
153         auxN=self.neighbors[:]
154         auxN.remove(event.getSource())
155         for n in M[1]:  #Descarto de la copia a todos los
                vecinos por los que ya paso el paquete
156           if n in auxN:
157             auxN.remove(n)
158
159         ruta =M[0][5:]
160         for m in ruta:
161           if m in auxN:
162             auxN.remove(m)
163                 #Comenzando backtraking
164         if len(auxN) == 0:    #Si la lista queda vacia
                entonces me agrego a la lista de nodos que no
                llevan al destino
165           M[1].append(self.id)
166           #Envio el paquete regreso a la fuente(o mi padre)
167           B=[M[0][:],M[1][:]]
168           B[0][0]="B"
169           newevent = Event(B, event.getTime() + 1.0, event.
                getSource(), self.id)
170           self.transmit(newevent)
171         else:
172           self.contPaq=self.contPaq+1
173           next = funciones.Compass_Routing(self.id, auxN, M
                [0][2], ruta)
174           M[0].append(self.id)
175           newevent = Event(M, event.getTime() + 1.0, next,
                self.id)
176           self.transmit(newevent)
177       else:
178         #Envio el paquete de vuelta al nodo origen
179         newM=[M[0][:],M[1][:]]
180         newM[0].append(self.id)
181         ruta = newM[0][5:]
182         newM[0][0]="ACK"
183         newevent = Event(newM, event.getTime() + 1.0, ruta[
              len(ruta)-2], self.id)
184         self.transmit(newevent)
185     if M[0][0] == "B": #Backtrack
186       auxN=self.neighbors[:]    #Creo una copia de mi lista
            de vecinos
187       if self.id != M[0][1]:
188         if M[0][len(M[0])-1] in auxN:
189           auxN.remove(M[0][len(M[0])-1])  #Borro de la lista
                a mi "padre"
190       for n in M[1]:  #Descarto de la copia a todos los
            vecinos por los que ya paso el paquete
191         if n in auxN:
```

```
192             auxN.remove(n)
193         ruta =M[0][5:]
194         for m in ruta:
195           if m in auxN:
196             auxN.remove(m)
197         if len(auxN) == 0:    #Si la lista queda vacia
              entonces me agrego a la lista de nodos que no
              llevan al destino
198           self.contPaq=self.contPaq-1
199           if self.id == M[0][1]:
200             self.paqRegreso=self.paqRegreso+1
201             if(self.paqRegreso  < parametros.num_paquetes):
202               newevent = Event([["START",self.id],[]], event.
                    getTime() + 1.0,self.id,self.id)
203               self.transmit(newevent)
204           else:
205             if self.id==M[0][len(M[0])-1]:
206               del M[0][len(M[0])-1]
207             M[1].append(self.id)
208             #Envio el paquete regreso a la fuente(o mi padre)
209             newevent = Event(M, event.getTime() + 1.0,M[0][len
                  (M[0])-1],self.id)
210             self.transmit(newevent)
211           #Verifico si ya termine de enviar mis paquetes
212           if (self.contPaq==0 and self.paqRegreso ==
                parametros.num_paquetes and self.terminoPaq==0):
213             self.terminoPaq=1
214             if (self.id==parametros.coordinador):
215               newevent = Event([["REWIRE","PIF",0],[]], event.
                    getTime() + 1.0,self.id, self.id)
216               self.transmit(newevent)
217             if (self.visitadoR==True and self.rec==0):
218               self.rec=1
219               for n in self.neighbors:
220                 if n != self.padreRec:
221                   newevent = Event([["REWIRE","PIF",0],[]],
                        event.getTime()+1.0,n, self.id)
222                   self.transmit(newevent)
223                   self.comRec += 1
224               if self.comRec == 0:
225                 if (self.id != parametros.coordinador):
226                   newevent = Event([["REWIRE", "PIF", self.
                        maxTray], []], event.getTime()+1.0, self
                        .padreRec, self.id)
227                 else:
228                   newevent = Event([["REWIRE","PI"],[]], event
                        .getTime()+1.0, self.padreRec, self.id)
229                 self.transmit(newevent)
230         else:
```

```
231             P = [M[0][:],M[1][:]]
232             P[0][0] = "P"
233             next = funciones.Compass_Routing(self.id,auxN,M
                    [0][2],ruta)
234             newevent = Event(P, event.getTime() + 1.0,next,self.
                    id)
235             self.transmit(newevent)
236
237         if M[0][0] == "ACK":
238           self.contPaq -= 1
239           if((M[0][1]) == self.id):
240             self.paqRegreso=self.paqRegreso+1
241             ruta = M[0][6:]
242             if len(ruta) > self.maxTray:
243               self.maxTray = len(ruta)
244             k = self.neighbors.index(ruta[0])
245             self.frec_enlaces[k] = self.frec_enlaces[k] + 1
246             for i in range(1,len(ruta)):
247               if (ruta[i] in self.frec_nodos[0]):
248                 j = self.frec_nodos[0].index(ruta[i])
249                 self.frec_nodos[1][j]=self.frec_nodos[1][j]+1
250               else:
251                 self.frec_nodos[0].append(ruta[i])
252                 self.frec_nodos[1].append(1)
253             if(self.paqRegreso < parametros.num_paquetes):
254               newevent = Event([["START", self.id], []], event.
                      getTime() + 1.0, self.id, self.id)
255               self.transmit(newevent)
256           else:
257             ruta = M[0][5:]
258             indice=ruta.index(self.id)
259             newevent = Event(M, event.getTime() + 1.0, ruta[
                    indice-1], self.id)
260             self.transmit(newevent)
261           if (self.contPaq==0 and self.paqRegreso == parametros.
                  num_paquetes and self.terminoPaq==0):
262             self.terminoPaq=1
263             if (self.id==parametros.coordinador):
264               newevent = Event([["REWIRE","PIF",0],[]], event.
                      getTime() + 1.0,self.id, self.id)
265               self.transmit(newevent)
266             if (self.visitadoR==True and self.rec==0):
267               self.rec=1
268               for n in self.neighbors:
269                 if n != self.padreRec:
270                   newevent = Event([["REWIRE", "PIF", 0],[]],
                          event.getTime()+1.0,n, self.id)
271                   self.transmit(newevent)
272                   self.comRec += 1
```

```
273             if self.comRec == 0:
274               if (self.id != parametros.coordinador):
275                 newevent = Event([["REWIRE", "PIF", self.
                        maxTray], []], event.getTime()+1.0, self.
                        padreRec, self.id)
276               else:
277                 newevent = Event([["REWIRE","PI"],[]], event.
                        getTime()+1.0, self.padreRec, self.id)
278               self.transmit(newevent)
279
280       """Sincronizacion para comenzar la etapa de recableado
              """
281       if M[0][0] == "REWIRE":
282         if M[0][1]=="PIF":
283           self.comRec-=1
284           if self.visitadoR==False:
285             self.visitadoR = True
286             self.padreRec = event.getSource()
287             if (self.paqRegreso == parametros.num_paquetes and
                    self.terminoPaq==1):
288               self.rec = 1
289               for n in self.neighbors:
290                 if n != self.padreRec:
291                   newevent = Event([["REWIRE","PIF",0],[]],
                          event.getTime()+1.0,n, self.id)
292                   self.transmit(newevent)
293                   self.comRec += 1
294           if event.getSource()!=self.padreCiclo:  #sumatoria
                de el numero de cambios
295             if M[0][2]>self.maxTray:
296               self.maxTray=M[0][2]
297           if self.comRec == 0 and self.terminoPaq==1:
298             if (self.id != parametros.coordinador):
299               newevent = Event([["REWIRE", "PIF", self.maxTray
                      ], []], event.getTime()+1.0, self.padreRec,
                      self.id)
300             else:
301               newevent = Event([["REWIRE", "PI"], []], event.
                      getTime()+1.0, self.padreRec, self.id)
302             self.transmit(newevent)
303
304         if M[0][1] == "PI":
305           if self.visitadoRPI == False:
306             self.visitadoRPI = True
307             self.visitadoC = False
308             self.visitadoCPI=False
309             self.comCiclo=1
310             self.cic=0
311             for n in self.neighbors:
```

```
312             newevent = Event([["REWIRE", "PI"], []], event.
                    getTime()+1.0, n, self.id)
313             self.transmit(newevent)
314           self.recablea(event)
315
316       """Sincronizacion␣para␣comenzar␣el␣siguiente␣ciclo"""
317       if M[0][0]=="CICLO":
318         if M[0][1]=="PIF":
319           self.comCiclo -= 1
320           if (self.terminoRec==0):
321             self.pifsCiclo.append(event.getSource())
322
323           if self.visitadoC == False:
324             self.visitadoC = True
325             self.contPaq=0
326             self.paqEnviados=0
327             self.paqRegreso=0
328             self.padreCiclo = event.getSource()
329             if (self.contRecab==0 and self.terminoRec==1):
330               self.cic=1 #Estoy dentro del PIF
331               for n in self.neighbors:
332                 if n != self.padreCiclo:
333                   newevent = Event([["CICLO", "PIF", 0],[]],
                          event.getTime()+1.0, n, self.id)
334                   self.transmit(newevent)
335                   self.comCiclo += 1
336           if event.getSource()!=self.padreCiclo:  #Sumatoria
                del numero de cambios
337             self.numRec=self.numRec+M[0][2]
338           if (self.comCiclo == 0 and self.terminoRec==1):
339             if (self.id != parametros.coordinador):
340               newevent = Event([["CICLO", "PIF", self.numRec],
                      []], event.getTime()+1.0, self.padreCiclo,
                     self.id)
341             else:
342               newevent = Event([["CICLO", "PI"], []], event.
                    getTime()+1.0, self.padreCiclo, self.id)
343             self.transmit(newevent)
344
345         if M[0][1]=="PI":
346           if self.visitadoCPI == False:
347             if self.ciclo < parametros.ciclos:
348               self.visitadoCPI = True
349               self.rec = 0
350               self.visitadoR = False
351               self.visitadoRPI = False
352               self.etapaRec = 0
353               self.contRecab = 1
354               self.comRec = 1
```

```
355             self.terminoPaq = 0
356             self.terminoRec = 0
357             self.numRec = 0
358             self.maxTray = 0
359             self.iDesc = -1
360             self.iConex = -1
361             self.auxSolEnv = 0
362             self.limpiarListas()
363             self.ciclo += 1
364             for n in self.neighbors:
365               newevent = Event([["CICLO", "PI"], []], event.
                      getTime()+1.0, n, self.id)
366               self.transmit(newevent)
367             ##---Aqui envio mensaje a mi mismo para comenzar
                    el siguiente ciclo---##
368             newevent = Event([["START", self.id], []], event
                    .getTime() + 1.0, self.id, self.id)
369             self.transmit(newevent)
370
371     if M[0][0] == "R":
372       if(self.etapaRec==0):
373                 self.bufferRec.append(M)
374       else:
375         # Recibo mensaje de desconexion
376         if M[0][1] == "D":
377           # Recibo peticion de desconexion
378           if M[0][3] =="C":
379             if ((event.getSource() in self.neighbors) and (
                    event.getSource() not in (self.idEnlaceExtra
                    )) and (event.getSource() not in (self.
                    neighborsPerm))):
380               self.idsDesc.append(event.getSource())
381               newevent = Event([["R-ACK", "D", self.id, "SI"
                      , M[0][3], M[0][4], "enLista"], []], event
                      .getTime() + 1.0, event.getSource(), self.
                      id)
382             else:
383               newevent = Event([["R-ACK", "D", self.id, "NO"
                      , M[0][3], M[0][4],], []], event.getTime()
                       + 1.0, event.getSource(), self.id)
384             self.transmit(newevent)
385           ## Recibo cancelacion de desconexion
386           elif M[0][3] == "cancelar":
387             self.idsDesc.remove(event.getSource())
388         ## Recibo mensaje de conexion
389         elif M[0][1]== "C":
390           if ((event.getSource() not in self.neighbors)and
                  ((event.getSource() not in self.
                  idSolicitudEnlExtra) or ((event.getSource() in
```

```
                self.idSolicitudEnlExtra) and  (event.
               getSource()<self.id)))):
391             self.neighbors.append(event.getSource())
392             self.frec_enlaces.append(0)
393             if (event.getSource() in self.
                    idSolicitudEnlExtra) and  (event.getSource()
                    <self.id):
394               self.idSolicitudEnlExtra.remove( event.
                      getSource() )
395             newevent = Event([["R-ACK", "C", self.id, "SI"],
                     []], event.getTime()+ 1.0, event.getSource
                    (), self.id)
396             self.transmit(newevent)
397             if self.cic == 1:
398               self.comCiclo += 1
399               newevent = Event([["CICLO", "PIF", 0],[]],
                      event.getTime()+1.0, event.getSource(),
                      self.id)
400               self.transmit(newevent)
401           else:
402             newevent = Event([["R-ACK", "C", self.id, "NO"],
                     []], event.getTime()+ 1.0, event.getSource
                    (), self.id)
403             self.transmit(newevent)
404
405     if M[0][0] == "R-ACK":
406       if M[0][1] == "D":
407         if M[0][3] == "NO":
408           self.contRecab -= 1
409         elif M[0][3]=="SI":
410           if M[0][6]== "enLista": #el nodo que me contesta
                  si puede borrarme de sus vecinos
411             if M[0][5] not in self.neighbors:
412               self.idSolicitudEnlExtra.append(M[0][5])
413               newevent = Event([["R", "C", self.id],[]],
                      event.getTime()+1.0, M[0][5], self.id)
414               self.numRec += 1
415               self.transmit(newevent)
416             else:
417               self.contRecab -= 1
418               newevent = Event([["R", "D", self.id, "
                      cancelar"], []], event.getTime()+1.0, self
                      .iDesc, self.id)
419               self.transmit(newevent)
420           elif M[0][6]== "borrado":
421             self.contRecab-=1
422             indVec = self.neighbors.index( event.getSource()
                    )
423             del self.neighbors[indVec]
```

```
424             del self.frec_enlaces[indVec]
425             self.idEnlaceExtra.remove(event.getSource())
426             if event.getSource() in self.pifsCiclo:
427               self.comCiclo+=1
428           ## Recibo confirmacion de desconexion
429           elif M[0][6] == "confirmacion":
430             indVec = self.neighbors.index(event.getSource())
431             del self.neighbors[indVec]
432             del self.frec_enlaces[indVec]
433             self.idsDesc.remove(event.getSource())
434             if self.cic == 1:
435               self.comCiclo -= 1
436               if self.comCiclo == 0:
437                 if (self.id != parametros.coordinador):
438                   newevent = Event([["CICLO", "PIF", self.
                        numRec], []], event.getTime()+1.0,
                        self.padreCiclo, self.id)
439                 else:
440                   newevent = Event([["CICLO", "PI"], []],
                        event.getTime()+1.0, self.padreCiclo,
                        self.id)
441                 self.transmit(newevent)
442             newevent = Event([["R-ACK", "D", self.id, "SI",
                  0, 0, "borrado"], []], event.getTime() +
                  1.0, event.getSource(), self.id)
443             self.transmit(newevent)
444
445       if M[0][1] == "C":
446         if M[0][3]=="SI":
447           self.neighbors.append(event.getSource())
448           self.frec_enlaces.append(0)
449           if event.getSource() in self.idSolicitudEnlExtra:
450             self.idEnlaceExtra.append(event.getSource())
451             self.idSolicitudEnlExtra.remove( event.getSource
                  () )
452           if self.iConex == -1:
453             self.contRecab -= 1
454             self.extrasLibres=self.extrasLibres-1
455           elif (event.getSource() == self.iConex) and (self.
                numRec>0):
456             newevent = Event([["R-ACK", "D", self.id, "SI",
                  0, 0, "confirmacion"], []], event.getTime()
                  +1.0, self.iDesc, self.id)
457             self.transmit(newevent)
458             if (self.visitadoC==True and self.cic==0):
459               if self.iDesc==self.padreCiclo:
460                 self.padreCiclo=self.id
461                 self.visitadoC==False
462                 self.comCiclo+=1
```

```
463             elif M[0][3]=="NO":
464               if event.getSource() in self.idSolicitudEnlExtra:
465                 self.idSolicitudEnlExtra.remove( event.getSource
                        () )
466               self.contRecab -= 1
467               if event.getSource() == self.iConex and (self.
                      numRec>0):
468                 self.numRec -= 1
469                 newevent = Event([["R", "D", self.id, "cancelar"
                        ], []], event.getTime()+1.0, self.iDesc,
                        self.id)
470                 self.transmit(newevent)
471             if (len(self.idSolicitudEnlExtra)>0) and (self.
                    iConex== -1):
472               self.contRecab += 1
473           if (self.contRecab==0 and self.terminoRec==0):
474             self.terminoRec=1
475             del self.pifsCiclo[:]
476             self.pifsCiclo=[]
477             if(self.id==parametros.coordinador):
478               newevent = Event([["CICLO", "PIF", 0], []], event.
                      getTime() + 1.0, self.id, self.id)
479               self.transmit(newevent)
480             if (self.visitadoC==True and self.cic==0):
481               self.cic = 1 #Estoy dentro del PIF
482               for n in self.neighbors:
483                 if n != self.padreCiclo:
484                   newevent = Event([["CICLO", "PIF", 0],[]],
                          event.getTime()+1.0, n, self.id)
485                   self.transmit(newevent)
486                   self.comCiclo += 1
487               if self.comCiclo == 0:
488                 if (self.id != parametros.coordinador):
489                   newevent = Event([["CICLO","PIF",self.numRec
                          ],[]], event.getTime()+1.0, self.
                          padreCiclo, self.id)
490                 else:
491                   newevent = Event([["CICLO","PI"],[]], event.
                          getTime()+1.0, self.padreCiclo, self.id)
492                 self.transmit(newevent)
493 #
        ----------------------------------------------------------------------

494 # "main()"
495 #
        ----------------------------------------------------------------------

496 # construye una instancia de la clase Simulation recibiendo
        como parametros el nombre del
```

```
497 # archivo que codifica la lista de adyacencias de la grafica
        y el tiempo max. de simulacion
498 if len(sys.argv) != 2:
499     print "Please␣supply␣a␣file␣name"
500     raise SystemExit(1)
501 experiment = Simulation(sys.argv[1], sys.maxint)
502
503 # asocia un pareja proceso/modelo con cada nodo de la
        grafica
504
505 for i in range(1,len(experiment.graph)+1):
506     m = simulacionRecableado()
507     experiment.setModel(m, i)
508
509 # inserta un evento semilla en la agenda y arranca
510
511 seed=[]
512 for i in range(1,len(experiment.graph)+1):
513     seed.append(Event([["START",i],[]], 0.0, i,i))
514     experiment.init(seed[i-1])
515
516 experiment.run()
```

Referencias

[1] A.-L. Barabási, *Linked: How Everything Is Connected to Everything Else and What It Means for Bussiness, Science, and Everyday Life.* Plume, 2004.

[2] G. S. Manku, M. Bawa, P. Raghavan, and V. Inc, "Symphony: Distributed hashing in a small world," in *In Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, pp. 127–140, 2003.

[3] Z. Gengzhong and L. Qiumei, "A survey of wireless sensor networks based on small world network model," in *Electrical and Control Engineering (ICECE), 2010 International Conference on*, pp. 2647–2650, IEEE, 2010.

[4] S. Tisue and U. Wilensky, "Netlogo: A simple environment for modeling complexity," in *in International Conference on Complex Systems*, pp. 16–21, 2004.

[5] J. C. Neumann, *The Book of GNS3.* San Francisco, CA, USA: No Starch Press, 1st ed., 2014.

[6] V. Batagelj and A. Mrvar, "Pajek – analysis and visualization of large networks," in *GRAPH DRAWING SOFTWARE*, pp. 77–103, Springer, 2003.

[7] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan, "Mason: A multiagent simulation environment," *Simulation*, vol. 81, pp. 517–527, July 2005.

[8] R. Marcelín-Jiménez, R. Esquivel-Villafaña, and S. Rajsbaum, "A flexible simulator for distributed algorithms," in *Computer Science, 2003. ENC 2003. Proceedings of the Fourth Mexican International Conference on*, pp. 176–181, IEEE, September 2003.

[9] R. Marcelín-Jiménez, "Estrellas de rock de internet," *Revista ¿Cómo ves?*, p. 16, Diciembre 2012.

[10] S. Milgram, "The small world problem," *Psychology Today*, vol. 2, pp. 60–67, 1967.

[11] M. V. Steen, *Graph Theory and Complex Networks*. On Demand Publishing, LLC-Create Space, 2010.

[12] D. Meunier, R. Lambiotte, and E. T. Bullmore, "Modular and hierarchically modular organization of brain networks," *Frontiers in neuroscience*, vol. 4, 2010.

[13] R. Guimera and L. A. N. Amaral, "Functional cartography of complex metabolic networks," *Nature*, vol. 433, no. 7028, pp. 895–900, 2005.

[14] M. E. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical review E*, vol. 69, no. 2, p. 026113, 2004.

[15] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, pp. 440–442, June 1998.

[16] A.-L. Barabási and R. Albert, "Emergence of Scaling in Random Networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[17] R. Cohen, K. Erez, D. ben Avraham, and S. Havlin, "Resilience of the Internet to random breakdowns," *Physical Review letters*, vol. 85, pp. 4626–4628, November 2000.

[18] R. Cohen, K. Erez, D. ben Avraham, and S. Havlin, "Breakdown of the Internet under Intentional Attack," *Physical Review letters*, vol. 86, pp. 3682–3685, April 2001.

[19] J. M. Epstein and R. L. Axtell, *Growing artificial societies: social science from the bottom up*. A Bradford book, Brookings Institution Press, 1996.

[20] B. Jiang, C. Yang, T. Yamada, and T. Terano, "The role of social network in migration and economic aggregation through a brownian agent model," in *Agent-Based Approaches in Economic and Social Complex Systems VII* (T. Murata, T. Terano, and S. Takahashi, eds.), vol. 10 of *Agent-Based Social Systems*, pp. 163–181, Springer Japan, 2013.

[21] W. Qi, L. Zhong, and L. Wang, "Dynamic evolution analysis of network structure of transactional community," in *Systems and Informatics (ICSAI), 2012 International Conference on*, pp. 2614–2618, IEEE, 2012.

[22] A. Helmy, "Small worlds in wireless networks.," *IEEE Communications Letters*, vol. 7, no. 10, pp. 490–492, 2003.

[23] T. Jia and R. V. Kulkarni, "On the structural properties of small-world networks with range-limited shortcut links," *Physica A: Statistical Mechanics and its Applications*, vol. 392, no. 23, pp. 6118–6124, 2013.

[24] E. Kranakis, H. Singh, and J. Urrutia, "Compass routing on geometric networks," in *IN PROC. 11 TH CANADIAN CONFERENCE ON COMPUTATIONAL GEOMETRY*, pp. 51–54, 1999.

[25] A. Segall, "Distributed network protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 1, pp. 23–34, 1983.

[26] B. Mohar and Y. Alavi, "The laplacian spectrum of graphs," *Graph theory, combinatorics, and applications*, vol. 2, pp. 871–898, 1991.

[27] N. Strom, "A tonotopic artificial neural network architecture for phoneme probability estimation," in *Automatic Speech Recognition and Understanding, 1997. Proceedings., 1997 IEEE Workshop on*, pp. 156–163, IEEE, 1997.

Casa abierta al tiempo
UNIVERSIDAD AUTÓNOMA METROPOLITANA

Estudio de la dinámica estructural de las redes complejas usando simulación basada en agentes

Idónea Comunicación de Resultados para obtener el grado de

Maestra en Ciencias y Tecnologías de la Información

Presentado por:
Lic. Magali Alexander López Chavira

Asesor: Dr. Ricardo Marcelín Jiménez

Jurado Calificador:

Presidente: Dr. Víctor Manuel Landassuri Moreno

Secretario: Dr. Ricardo Marcelín Jiménez

Vocal: Dr. Michael Pascoe Chalke

México, D.F. a 17 de Octubre del 2014