UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

ESTUDIO DE ALGORITMOS DE RUTEO CONSIDERANDO RESTRICCIONES DE TIEMPO REAL

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRA EN CIENCIAS

P R E S E N T A:

MAGALI ARELLANO VÁZQUEZ

DIRECTOR DE LA TESIS:

DR. HÉCTOR BENÍTEZ PÉREZ

MÉXICO, D.F. **2011.**

A mis padres, por darme todas las herramientas para seguir mis sueños.

AGRADECIMIENTOS

- A mis padres por su apoyo incondicional.
- A la Universidad Nacional Autónoma de México y en particular al Posgrado en Ciencia e Ingeniería de la Computación.
- Al programa de becas nacionales de CONACYT [1].
- A la DGAPA-UNAM, por el apoyo recibido por parte del proyecto de investigación PAPIIT No. IN 103310.
- Al Dr. Héctor Benítez Pérez por dirigir mi tesis, sobre todo por su paciencia y apoyo durante la realización de este trabajo.
- A mis sinodales, Dr. Javier Gómez, Dr. Jorge Ortega Arjona, Dr. Víctor Rangel Licea e Ing. Mario Rodríguez, por el tiempo dedicado a la revisión de este trabajo.
- A mis compañeros del DISCA-IIMAS por sus enseñanzas y amistad.
- A Paul Erick Méndez Monroy por su apoyo, paciencia y asesoría en momentos de pánico.
- A mis amigos y compañeros de clases de maestría: Diego Enrique, Esmeralda, Fede, Jaime, Jimmy, Laura, Memo, Naim, Paty, Ricardo, Rodrigo, Ruben, Toño y Toño Zacatecas. Por compartir conmigo su amistad y hecho de esta ciudad mi segunda casa.
- A mis amigos: Casandra, Melina, Perla, Iván, Cesarunix, Daniel, Maricruz, Pilar, Carlos, César Guadarrama, Magda, Mirella, Johnny, Caro y Amed. Por apoyarme y seguir tan cercanos a mi, a pesar de la distancia y las largas ausencias.
- A la faculdad de ciencias de la UAEMor y al Dr. Miguel Robles por impulsarme a continuar mis estudios de posgrado.
- Al personal administrativo del Posgrado en Ciencia e Ingeniería de la Computación.
- A todas las personas que contribuyeron directa e indirectamente con la realización de esta tesis.

Mag ='-'=

[1] No. CVU 265975

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

INTRODUCCIÓN

Si al franquear una montaña en la dirección de una estrella,
el viajero se deja absorber demasiado por los problemas de
la escalada, se arriesga a olvidar cual es la estrella que lo guía.
Antoine de Saint-Exupery

En la actualidad las redes de computadoras han alcanzado un gran auge y han cobrado vital importancia en todos los ámbitos de la sociedad.

La transmisión de datos a través de redes móviles es una de las áreas de telecomunicaciones que experimentan un crecimiento muy notable en los últimos años, debido a que las computadoras portátiles han reducido su tamaño y precio, por lo tanto lo usuarios demandan soporte para servicios como el correo electrónico, la mensajería instantánea y la transferencia de datos. Esto ha ocasionado que algunos países el tráfico de datos haya superado al tráfico de voz.

Por lo tanto mejorar los algoritmos actuales de comunicación entre los dispositivos que conforman las redes permite contribuir a este gran fenómeno.

1.1. Problema

La problemática que presentan los sistemas distribuidos móviles es que debido al cambio constante de topología de este tipo de sistema es difícil establecer y mantener una ruta de comunicación entre 2 nodos, cuando ésta no es directa. El método más usado para encontrar una ruta de un nodo a otro es calcular una imagen global de la red para poder escoger una ruta adecuada, sin embargo al cambiar tanto la topología esto se vuelve costoso.

1.2. Metas

Con base a los conceptos de sistemas de tiempo real y a la gran variedad de algoritmos de enrutamiento adaptativo y parcialmente adaptativo, se desea:

Diseñar un algoritmo que nos permita construir una ruta en el tiempo en un sistema distribuido móvil contemplando restricciones derivadas del uso de tiempo real de manera local que repercutan en el estado global del sistema. Definir rutas de acceso de manera dinámica, esto mediante la coordinación y cooperación entre los nodos. Garantizar la conectividad en la red.

1.3. Objetivo

Proponer un algoritmo que contemple las restricciones de tiempo real locales, que puedan ser usadas para determinar el estado global del sistema con el mínimo de comunicación buscando encontrar una ruta conexión en un sistema distribuido móvil.Este algoritmo sería una aportación al ruteo en sistemas distribuidos móviles, porque podríamos garantizar la conectividad de una red, tomando en cuenta sus características instantáneas, lo cual permitiría la comunicación a pesar de que la red cambie constantemente.

1.4. Método

Durante la primera etapa se revisaron los textos más relevantes y actuales relacionados con los algoritmos de enrutamiento adaptativos y parcialmente adaptativos, sistemas de tiempo real y representación de redes, posteriormente se seleccionaron algunos que se adaptaban a las características del problema.

Se contemplaron las restricciones de los algoritmos de planificación de tareas en sistemas de tiempo real con base al nodo local, para la definición e implementación de una función de conexión.

Finalmente, se realizó un profundo estudio de las matrices de adyacencia de la red en distintas condiciones, estas condiciones varían a través de una función que contempla parámetros de la red, además de parametros de cada nodo. A partir de las matrices de adyacencia y utilizando el algoritmo de Floyd-Warshall encontramos las rutas más cortas en el tiempo actual, con esta información construimos una ruta que se construye a cada instante de tiempo que la topología cambia.

1.5. Logros

Se desea hacer una primera aproximación de un algoritmo que nos permitirá reducir los recursos utilizados para la comunicación, actualización y consulta de tablas de ruteo.

1.6. Organización del documento

En el capítulo 2 se presentan los antecedentes teóricos que sirvieron como base para la investigación, en particular conceptos de sistemas de tiempo real (como algoritmos de planificación), redes de computadoras (con un énfasis especial en algoritmos de ruteo adaptativo) y teoría de grafos (como representación de redes y algoritmos de rutas más cortas). Los detalles del problema se tratan en el capítulo 3. La explicación del método desarrollado, así como la construcción paso a paso del algoritmo se explica en el capítulo 4. Los resultados obtenidos de los experimentos y el análisis de éstos se presentan en el capítulo 5. Finalizamos con las conclusiones en la capítulo 6.

Capítulo 2

ANTECEDENTES

...Y el que es versado en la ciencia de los números puede hablaros de las regiones de peso y la medida, pero no puede conducirnos a ellas. Porque la visión de un hombre no presta sus alas a otro hombre...

El profeta, Gibran Jalil

Para la mejor comprensión de este trabajo es necesario conocer algunas definiciones sobre sistemas de tiempo real, ruteo en redes y representación de redes como grafos.

Los conceptos de tiempo real son importantes para el caso de estudio que se tratará. En el caso de estudio se considera que cada nodo en la red se comporta como un ruteador y a su vez cada nodo tiene un planificador de tareas. El servicio de enrutamiento entra a la lista de espera del planificador , ya que la prioridad del servicio de enrutamiento no es superior al de las tareas del nodo, de esta manera depende mucho de la disponibilidad del procesador del nodo el tiempo que se tarda en transmitir un paquete.

Los conceptos de teoría de grafos son fundamentales, ya que necesitamos una representación matemática del comportamiento del sistema.

2.1. Sistemas de tiempo real

Un sistema en tiempo real es aquel sistema digital que interactúa activamente con un entorno con dinámica conocida en relación con sus entradas, salidas y restricciones temporales, para darle un correcto funcionamiento de acuerdo con los conceptos de predictibilidad, estabilidad y control.

Los sistemas en tiempo real están presentes en nuestra vida diaria, prácticamente en todo lo que nos rodea; en los medios de transporte, en los medios de comunicación, en los electrodomésticos, en los

teléfonos celulares y en las centrales telefónicas digitales. Son un elemento indispensable para garantizar la generación, transmisión y distribución de la energía eléctrica y para asegurar la calidad y la seguridad de procesos industriales.

2.1.1. Definiciones

Un sistema consiste de un conjunto de tareas tal que:

$$T = \{\tau_1, \tau_2, \ldots, \tau_n\} \tag{2.1}$$

donde el peor caso es cuando el tiempo de ejecución de cada $\tau_i \in T$ es C_i, donde C_i es el tiempo máximo de computación. El sistema es de tiempo real si existe al menos una tarea $\tau_i \in T$, la cual queda en una de las siguientes categorías:

1. Tarea τ_i es una *tarea de tiempo real duro*. Esto es que la ejecución de la tarea τ_i debe ser completada en un deadline[1] D_i, es decir, $C_i \leq D_i$ [2].
2. Tarea τ_i es una *tarea de tiempo real suave*. Esto es que si la tarea τ_i termina de ejecutarse después de su deadline, no hay ninguna penalización. Una función de penalización $P(\tau_i)$ es definida por la tarea. Si $C_i \leq D_i$, en el caso $P(\tau_i) = 0$. De otra manera $P(\tau_i) > 0$. El valor de $P(\tau_i)$ es una función creciente de $C_i - D_i$ [2].
3. Tarea τ_i es una *tarea de tiempo real firme*. Esto es que si la tarea τ_i termina antes de su deadline D_i se otorga una recompensa como ganancia. La función recompensa $R(\tau_i)$ se define por la tarea. Si $C_i \geq D_i$, en el caso $R(\tau_i) = 0$. De otra manera $R(\tau_i) > 0$. El valor de $R(\tau_i)$ es una función creciente de $D_i - C_i$ [2].

El sistema de tiempo real $T = \{\tau_1, \tau_2, \ldots, \tau_n\}$ puede ser una combinación de tareas de tiempo real duras, suaves y firmes [2].

Planificación de un sistema

Para un conjunto de tareas dadas, el problema de la planificación es definir en que orden se ejecutarán, tal que las restricciones sean cumplidas. Básicamente el problema de la planificación es determinar una secuencia para la ejecución de tareas, tal que todas sean completadas y no se descarten tareas por haberse vencido el deadline [2].

Dado un sistema de tiempo real, el enfoque apropiado de la planificación debe ser diseñado basado en las propiedades del sistema y en la tareas que ocurren en éste. Las propiedades son las siguientes:

Duras / Suaves / Firmes Como se describieron en la sección anterior.

[1] Tiempo en el cual la ejecución de la tarea debe ser concluida, después de este la tarea es descartada

Periódicas / No periódicas Las tareas periódicas son aquellas que se activan regularmente en períodos de tiempo fijos. Normalmente tienen restricciones las cuales indican que instancias de ellas deben ejecutarse por período P. Las tareas no periódicas son las que se activan irregularmente. Las restricciones que se aplican para este tipo de tareas son relativas al deadline [2].

Apropiativa / No apropiativa En algunos sistemas, algunas tareas pueden tener prioridad sobre otras. En el caso de la ejecución de tareas apropiativas, éstas una vez que inician son completadas sin interrupción [2].

Multiprocesador / Sistemas uniprocesador El número de procesadores es uno de los principales factores que se toman en cuenta para la planificación de un sistema de tiempo real. En los sistemas multiprocesador, los algoritmos de planificación previenen el acceso simultáneo a recursos y dispositivos compartidos [2].

Tareas independientes / tareas dependientes Dado un sistema de tiempo real, la tarea que comenzará su ejecución requerirá de recibir información de otra tarea del sistema. Entonces, la ejecución de una tarea podría comenzar después de terminar la ejecución de otra tarea. Este es el concepto de dependencia. Las tareas dependientes usan memoria compartida o comunicación de datos para transferir la información generada por otra tarea y es requerida por otra tarea [2].

2.1.2. Algoritmos de planificación

Existen diversos algoritmos de planificación tales que, se pueden construir relaciones de ejecución validas dependiendo de ciertas condiciones de acceso.

Planificación RM (Rate monotonic)

Asumimos que todas las tareas son periódicas y la prioridad de la tarea τ_i es mayor que la prioridad de la tarea τ_j, donde $i < j$. El algoritmo de planificación RM es un ejemplo de algoritmos de prioridad impulsada con prioridad estática asignada en el sentido que las prioridades de todas las instancias son conocidas aún antes de su llegada. Las prioridades de las instancias de cada tarea son las mismas. Son determinadas sólo por el período de la tarea. Una tarea periódica consiste en una secuencia infinita de instancias con tiempos periódicos, donde el deadline de una petición puede ser menor que, mayor que, o igual que el tiempo que la instancia exitosa [2].

Una tarea periódica τ_i se caracteriza por tres parámetros P_i, el período de la instancia, C_i, el tiempo de ejecución y D_i, el deadline de las tareas. El factor de utilización de un conjunto de n tareas periódicas está definido por

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \tag{2.2}$$

donde $P_1, P_2, P_3, \ldots, P_n$ son períodos y $C_1, C_2, C_3, \ldots, C_n$ son los tiempos de ejecución de las n tareas [2].

Un inconveniente del algoritmo RM es que las prioridades de la tarea son definidas por sus períodos. Debemos cambiar las prioridades de las tareas para asegurar que todas las tareas críticas se

realicen. Supongamos que se nos da un conjunto de tareas que contienen las tareas τ_i y τ_j, donde $P_i < P_j$, pero τ_j es una tarea crítica y τ_i es una tarea no crítica. Revisamos la factibilidad de la planificación del algoritmo RM para las tareas $\tau_1, \tau_2, \tau_3, \ldots, \tau_n$, suponemos que si consideramos el peor tiempo de ejecución de las tareas, no podemos garantizar la planificación de las tareas. Sin embargo, en el caso promedio, todas las tareas son RM planificables. El problema es como acomodar las tareas para que las tareas críticas cumplan sus deadlines bajo el algoritmo de RM, incluso en el peor de los casos, mientras que las tareas no criticas, como τ_i, cumplen sus deadlines en muchos otros casos. La solución es también uno de los siguientes 2 métodos:

- Alargamos el período de tareas no críticas, por ejemplo τ_i, por un factor de k. La tarea original debería además ser reemplazada por k tareas, cada fase por la cantidad apropiada. El parámetro k debería ser elegido tal que obtenemos $P_i^{'} > P_J$.
- Reducimos el período de tareas críticas, por ejemplo τ_j, por un factor de k. Entonces podemos reemplazar la tarea original por una cuyo tiempo de ejecución es también reducido por un factor de k. El parámetro k se elige tal que $P_i^{'} > P_J$ [2].

El algoritmo RM toma $O((N+\alpha)^2)$ en el peor caso, donde N es el número total de las peticiones de cada hiper-período[1] de n tareas periódicas en el sistema y α es el número de tareas no periódicas [2].

Planificación EDF (Early Deadline First)

El algoritmo de planificación EDF es una de las prioridades impulsadas por el algoritmo en el cual la más alta prioridad se asigna a la petición que termina primero y una prioridad alta siempre opaca una prioridad baja. EDF también se conoce como el algoritmo planificador de deadline-monotónico [2].

Supongamos que cada tiempo llega una nueva tarea, que se inserta en una lista de tareas, clasificadas por sus deadlines. Si se utilizan listas ordenadas, el algoritmo EDF toma $O((N+\alpha)^2)$ en el peor caso, donde N es el número total de peticiones en cada hiper-período de n tareas periódicas en el sistema y α es el número de tareas no-periódicas.Aunque el algoritmo EDF es óptimo, no es único, existe otro algoritmo óptimo de planificación de prioridades dinámicas, el cual es el algoritmo de primero el menos laxo. El problema de la laxitud de un proceso, es definir como el deadline el menor tiempo computacional restante [2].

Si todas las tareas son periódicas y tienen fronteras relativas igual a sus períodos, pueden ser factibles de planificar por EDF si y sólo si $\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$ [2].

La laxitud de una tarea es la máxima cantidad de tiempo que la tarea puede esperar y aún así cumplir con su deadline. El algoritmo da la máxima prioridad al trabajo activo con la laxitud mínima. Entonces el trabajo con la máxima prioridad es el primero en ser ejecutado [2].

En el peor caso la complejidad del algoritmo LLF (Least Laxity First) es de $O((N+\alpha)^2)$, en el peor caso, donde N es el número total de peticiones en cada hiper-período de n tareas periódicas en el sistema y α es el número de tareas no periódicas [2].

[1]Definimos hiper-período como el período igual al menor múltiplo común de los períodos $P_1, P_2, \ldots, P_n$ de n tareas periódicas.

Es importante introducir los conceptos de planificación de tareas en tiempo real, ya que dentro de una red móvil cada nodo además de servir como router tiene otras funciones, por lo tanto el nodo cuenta con un planificador de tareas propio, el cual se encarga de asignar memoria y tiempo de procesamiento, por lo tanto es necesario saber como sucede esto a detalle, dado que puede darse el caso que el nodo se encuentre tan ocupado con sus tareas que no pueda servir como router.

Como el contexto de nuestro problema se ubica en un sistema distribuido móvil, es necesario incorporar los conceptos de tiempo real a los de redes.

2.2. Redes de comunicación

Una red de comunicación está hecha de nodos y enlaces. Dependiendo del tipo de red, los nodos tienen distintos nombres. Por ejemplo, en una red de computadoras existe un nodo llamado ruteador, mientras que en una red telefónica existe uno que cumple la misma función y es llamado central. En una red de comunicación el tráfico fluye desde el nodo origen hacia el nodo destino.

Tres factores son importantes para determinar el desempeño en una red de comunicación, estos factores son la topología, el método de conmutación (*switching*) y el algoritmo de ruteo.

La topología se define como los nodos conectados en la red. La topologías de malla e hipercubo son las más populares empleadas en la topología de computadoras paralelas tal como el cubo cósmico de calTech[2] [17] . El hipercubo es simétrico y regular.

El método de conmutación (*switching*) determina la manera que los mensajes visitan los ruteadores intermedios a lo largo de la ruta a su destino. Una gran cantidad de dispositivos utilizan el esquema de *switching* llamado *wormhole*, debido a su bajos requisitos de memoria y más importante, porque ha hecho la latencia casi independiente de la distancia entre la fuente y los nodos destino. En el swichting *wormhole*, el mensaje es divido en una secuencia de unidades de tamaño fijo, llamadas *flits*. La cabecera de cada *flit* establece la ruta a seguir a través de la red, mientras los datos faltantes siguen una tendencia de línea de ensamble [18].

La comunicación en una red depende de la latencia [3] y ésta a su vez está condicionada por la técnica de conmutación (*switching*) [18]. Existen 2 enfoques para el ruteo de paquetes, basados en las acciones que toma el dispositivo de conmutación (*switch*) cuando comienzan a llegar los *flits*.

1. Almacenar y reenviar
2. Atravesar
    - Atravesar Virtualmente

[2]Red de tipo hipercubo 6. Procesadores 8086 con 128 Kb de memoria. Intel iPSC/1 con procesadores Intel 80286 con 212 Kbytes de memoria, 8 puertos de E/S por cada nodo para formar un hipercubo de dimensión 7, el octavo puerto comunica el nodo con el host por red Ethernet

[3]Se denomina latencia a la suma de retardos temporales dentro de una red. Un retardo es producido por la demora en la propagación y transmisión de paquetes dentro de la red y el tiempo de reenvio, almacenamiento de los nodos intermedios.

- Wormhole

Por ejemplo el algoritmo *wormhole* , para la comprensión de este algoritmo es necesario introducir los conceptos de *phit* y *flit*. Se entiende como *phit* a la unidad más pequeña de información en la capa física, la cual se transfiere a través de un enlace físico en un ciclo; en cambio, un *flit* es una pequeña unidad de información en la capa de enlace del tamaño de una o varias palabras. El algoritmo *wormhole* funciona de la siguiente manera: Un paquete es dividido en n *flits*. El *flit* que encabeza el paquete gobernará la ruta, los *flits* restantes seguirán en pipeline a la cabecera (Figura 2.1) [4].

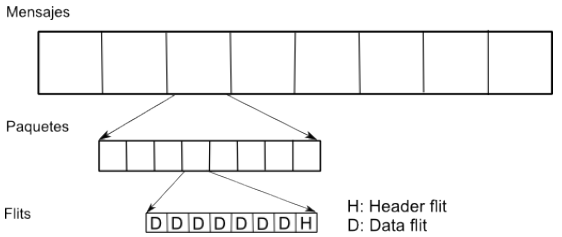

Figura 2.1: Esquema de fragmentación de paquetes para ser enviados por la red [11].

Por lo tanto el mensaje se asemeja a un gusano a través de la red, finalmente se reensambla en el destino. Los paquetes son trasmitidos desde un nodo como *flits*, se identifican 2 tipos de *flits*: *Flits* cabecera y *flits* de datos. El *flit* cabecera trata de obtener un canal de comunicación mientras que los *flits* de datos son transmitidos por el canal que ya se ha obtenido. Un canal se libera sólo cuando el último flit del mensaje ha pasado a través de él. Los flits de 2 mensajes distintos no pueden ser intercalados. Si la cabecera no puede continuar avanzando por la red, debido a que los canales de salida están ocupados, entonces la ruta queda bloqueada por los flits estancados, esto bloquea otras posibles comunicaciones. Este algoritmo es muy susceptible a presentar bloqueos mutuos. En caso de no presentarse, la latencia de la comunicación es

$$T_{WH} = d(t_r + t_w + t_m) + max(t_w, t_m)M \tag{2.3}$$

Donde:

$d =$ Longitud de la ruta

$\mu =$ la tasa máxima en $bits/segundo$ en el que los bits pueden ser transferidos a través de cada cable de un canal físico

$B = \mu(phits/segundo) =$ ancho de banda $= w\mu(bits/segundo)$

$w =$ tamaño del $flit =$ ancho de banda en bits

$M =$ Tamaño del paquete en $flits$

$t_r =$ Tiempo de decisión de enrutamiento en un solo router$(segundos)$

$t_w =$ latencia interna de conmutación en el router, una vez que se ha tomado la decisión de enrutamiento $(segundos/phit)$

$t_m = 1/B =$ latencia entre el canal y el router$(segundos/flits)$

como se ve en la figura 2.2 [4].

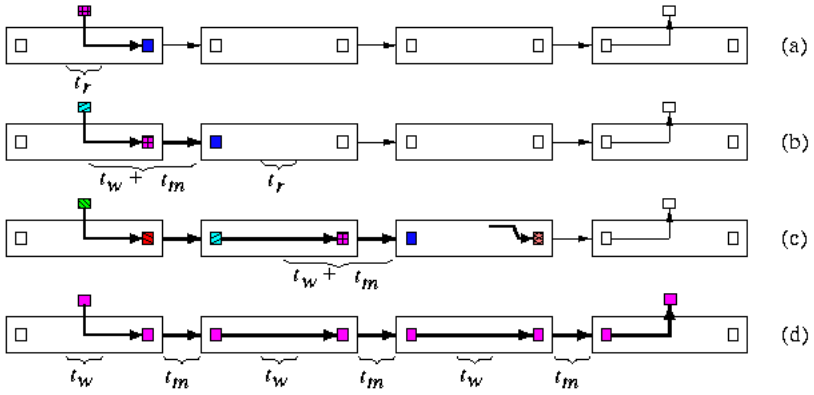

Figura 2.2: Latencia de la comunicación **(a)**La cabecera es copiada en el *buffer* de salida después de tomar una decisión sobre la ruta a seguir. **(b)** El *flit* cabecera es trasferido a un segundo ruteador y otro *flit* lo sigue. **(c)** El *flit* cabecera llega a un ruteador con el canal de salida ocupado y la cadena completa de *flits* se detiene, bloqueando ese canal. **(d)** La cadena de *flits* en caso de no tener conflicto de canales disponibles, establece un *wormhole* (hoyo de gusano) a través de los ruteadores [4].

Eventualmente los paquetes son encaminados desde una fuente hacia un destino. Tales paquetes pueden necesitar atravesar muchos puntos de cruce, similares a las intersecciones de calles en una red de transporte. Estos puntos de cruce en Internet se llaman ruteadores.

Las funciones de los ruteadores son leer la dirección de destino, marcar el paquete IP como paquete entrante para consultar su información e identificar el enlace al cual el paquete será reenviado y reenvía el paquete [10].

Haciendo una analogía entre la red de comunicación y una de transporte, el número de carriles y el límite de velocidad es similar al enlace de red que conecta 2 ruteadores, éste es limitado por cuanta información puede ser transmitida por unidad de tiempo, llamada ancho de banda o capacidad de enlace; esto es representado por una tasa de datos (por ejemplo 1.54 mega bits por segundo). Una red entonces carga tráfico en su enlace y a través de sus ruteadores hacia su eventual destino; el tráfico en la red se refiere a los paquetes generados por diferentes aplicaciones, tales como el Internet o el correo electrónico[10].

Si suponemos que el tráfico aumenta de repente, entonces los paquetes generados pueden ser agregados a la cola de un ruteador o pueden ser descartados, puesto que el ruteador tiene una cola de capacidad finita, conocida como *buffer*, para temporalmente guardar paquetes acumulados. Como el protocolo TCP/IP permite la posibilidad de que un paquete IP no sea entregado o sea descartado durante el camino, el *buffer* finito del ruteador no es problema. Desde el punto de vista de la eficiencia, es deseable no tener perdida de paquetes (o que sea mínima) durante el tránsito [10].

2.2.1. Ruteo

Pensemos en una red telefónica, cuando llega una llamada a la oficina se debe determinar si existe una ruta por la cual la llamada pueda ser conectada a su destino. Si la ruta se encuentra y si se decide usarla o no; si no hay una ruta disponible, debe decidirse que hacer con la llamada. Cada

uno de estos pasos es relativo a los aspectos de ruteo. La selección de una ruta disponible constituye un problema de ruteo en el sentido estricto, requiriendo que las rutas disponibles para cada flujo sean definidas. Esta definición puede ser el conjunto de todas las rutas permitidas por la estructura de la red o un subconjunto conveniente. Esto debe incluir una descripción de un procedimiento para encontrar tal ruta y para seleccionar cual ruta usar si hay más de una disponible. La decisión de conectar o no una llamada sobre una ruta disponible, es llamada control de flujo [5].

Una característica muy importante de una red de comunicación es el camino desde un nodo fuente al nodo destino. La ruta puede ser establecida manualmente, por lo que esa ruta seria una *ruta estática*. En general, es deseable usar un algoritmo de ruteo para determinar la ruta [10].

El objetivo del algoritmo de enrutamiento es, en general, dictado por la exigencia de la red de comunicación y el proveedor de servicios quiere imponer sobre sí mismo. Mientras que las metas pueden ser distintas para diferentes tipos de redes de comunicación, éstas pueden ser clasificadas en 2 categorías generales:

- Orientadas al usuario.
- Orientadas a la red.

Que una red esté orientada al usuario, significa que la red provee un buen servicio a cada usuario, tal que el tráfico puede moverse desde la fuente al destino rápidamente para ese usuario. Esto no debe ser para un usuario específico a expensas de otros usuarios entre la fuente o el destino de otros nodos de la red. El objetivo de esta red, es direccionar de manera eficiente y justa para que la mayoría de los usuarios tengan un servicio bueno y aceptable, en vez de dar el mejor servicio a un solo usuario. Existen 2 algoritmos con un impacto muy importante, éstos son los algoritmos de *Bellman-Ford* y el algoritmo de *Dijkstra*, éstos se detallaran más adelante [10].

La IP asume que la computadora está directamente conectada a una red local (por ejemplo, una LAN Ethernet) y que puede enviar paquetes directamente a cualquier otra computadora sobre esa misma red; si la dirección de destino es en la red local, IP simplemente accede al medio físico de transmisión y envía el paquete [10].

El problema aparece cuando la dirección de destino queda en otra red, en cuyo caso TCP/IP recurrirá a un *gateway* para enviar indirectamente los paquetes. Se denomina *gateway* a un dispositivo (ya sea otra computadora o un dispositivo específicamente diseñado a tal efecto) que está conectado a más de una red y tiene la capacidad para redirigir (forward) paquetes entre esas redes [10].

Para determinar a qué *gateway* deberán enviarse los paquetes, TCP/IP extrae la dirección de red del nodo de destino y consulta una tabla, denominada tabla de ruteo, en la cual se listan las redes conocidas y los *gateways* que pueden utilizarse para alcanzarlas [10].

Por ejemplo:

El asterisco indica que no es necesario ningún *gateway* para llegar a la red en cuestión dado que la máquina tiene conexión directa a la misma. El ruteo hacia redes remotas se realiza en base a direcciones de red (esto es, la parte de host de la dirección IP se ignora); además, la tabla de ruteo especifica tanto la

Red	Gateway
170.25.1.0	170.25.3.254
170.25.2.0	170.25.3.254
170.25.3.0	*
170.25.4.0	170.25.3.253

Tabla 2.1: Ejemplo de tabla de ruteo

red local como las remotas, y que para el caso de éstas últimas, indica la dirección IP de alguna máquina en la red local que puede ser utilizada para alcanzarla [19].

La ruta que seguirán los paquetes desde el origen hasta su destino se va decidiendo a medida que los mismos viajan por la red. Cada nodo es responsable de determinar cual es el próximo *salto* en dirección al nodo final en función del contenido de su tabla de ruteo. Este modelo de ruteo asume que si el nodo de destino no pertenece a la red local, deberá haber una entrada en la tabla de ruteo que especifique el *gateway* a utilizar. En otras palabras, asume que todos los nodos están al tanto de la estructura de la red [19].

En consecuencia, cada vez que la estructura de la red cambia (por ejemplo, cuando se agrega o elimina una subred), el administrador debería actualizar las tablas de ruteo en todos los nodos. Igualmente, si la red se interconectara a otra red, una nueva entrada deberá agregarse en las tablas de ruteo de cada máquina [19].

Siguiendo con este razonamiento, a medida que la red crece y se interconecta a otras redes las tablas de ruteo se hacen mas largas y complejas; inclusive sería posible que fueran virtualmente imposibles de construir o mantener, en especial si la red se conecta a Internet (formada por miles de redes independientes)[19].

Por supuesto, existen previsiones para enfrentar estos problemas: el ruteo dinámico o adaptativo y las rutas por defecto [19].

Tipos de ruteo

El algoritmo de ruteo indica el camino que un mensaje debe tomar para alcanzar sus destino, seleccionando el canal de salida adecuado. Este canal puede seleccionar de un conjunto de posibles opciones de acuerdo al tamaño del conjunto, los algoritmos de ruteo son clasificados en 3 categorías:

1. Ruteo determinista. Asignan un camino simple entre cada par de fuente y destino. Esta forma de ruteo es popular debido a que se evitan situaciones de bloqueo mutuo, resultando una aplicación simple. Sin embargo, en un mensaje que se trasmite por este método no se pueden utilizar rutas alternativas para evitar la congestión de canales a lo largo del camino hacia el destino, por lo que el rendimiento de la red es bajo [13].

2. Algoritmos de ruteo completamente adaptativo. En este tipo de algoritmos le permite al mensaje explorar los caminos disponibles, es decir el conjunto de opciones llega a su máximo en estos

algoritmos, por lo tanto es posible hacer un mejor uso de los recursos de la red pero implica mayor complejidad para el bloqueo mutuo [13].

3. Algoritmos de ruteo parcialmente adaptativo. Estos algoritmos tratan de combinar las ventajas de las otras 2 categorías para producir un encaminamiento con limitada adaptabilidad y establecer un equilibrio entre el rendimiento y la complejidad del ruteador. Estos algoritmos permiten seleccionar una ruta de un subconjunto de todas las posibles rutas, es decir, limitan el tamaño del conjunto de las posibles opciones. La mayoría de estos algoritmos están basados en algoritmos planares adaptativos, sobre todo han sido propuestos para redes de malla y de hipercubo [13].

2.2.2. Algoritmos de ruteo

La tabla de ruteo de un host puede construirse de dos maneras. Una posibilidad consiste en que el administrador (por medio de *scripts* que se ejecutan al inicializar el sistema, o por medio de comandos ejecutados interactivamente) introduzca manualmente las entradas de la tabla. Esta técnica se denomina ruteo estático, debido a que la tabla de ruteo se construye cuando la computadora se prende y no varía con el tiempo [19].

La otra posibilidad es ejecutar en cada host un programa que actualice automática y periódicamente la tabla de ruteo. Dichos programas se basan en el hecho de que una computadora siempre tiene acceso a otras computadoras conectadas a la red local; esto se traduce en que las tablas de ruteo contienen inicialmente al menos las direcciones de las redes locales [19].

Así, si todos los nodos de la red ejecutan un programa de estas características (llamado demonio de ruteo) al cabo de cierto tiempo habrán *descubierto* por si mismas la estructura de la red y construido sus tablas automáticamente. Mas aún, si se produjera algún cambio en la estructura de la red, bastaría con que alguna de las computadoras lo detectara para que en pocos segundos esa nueva información se propagará por toda la red [19].

Por ejemplo, la ruta utilizada para enviar un mensaje del sitio A al sitio B sólo se elige cuando se envía un mensaje. Debido a que la decisión se toma dinámicamente, a distintos mensajes se les pueden asignar rutas diferentes. El sitio A tomará una decisión para enviar el mensaje al sitio C; éste a su vez, decidirá enviarlo al sitio D y así sucesivamente. Con el tiempo, un sitio entregará el mensaje al sitio B. En general, un sitio envía un mensaje a otro sitio sobre el enlace menos utilizado en ese momento en particular. Los mensajes pueden llegar en cualquier orden, este problema se soluciona anexando un número de secuencia a cada mensaje [19].

Esta estrategia se denomina ruteo dinámico o adaptativo y tiene la ventaja de que, al ser automático, permite eliminar las tareas administrativas relacionadas con el mantenimiento de las tablas de ruteo, sin embargo, este tipo de ruteo es el más complejo de preparar y correr [19].

En redes de comunicación, un termino genérico para referirse a la medida de distancia (sin asignar una unidad) se le denomina costo, costo del enlace, costo de distancia o métrica de enlace [10].

2.2.3. Algoritmos de ruteo parcialmente adaptativo

Estos algoritmos tratan de combinar las ventajas del ruteo dinámico y el estático para producir un encaminamiento con limitada adaptabilidad y establecer un equilibrio entre el rendimiento y la complejidad del ruteador.

Algoritmo basado en el modelo de cambio de dirección

Los bloqueos mutuos en el ruteo *wormhole* son causados por paquetes esperando entre ellos en un ciclo, como se puede ver en la figura 2.3, se muestra una manera en la que ocurre un *deadlock* en una malla bidimensional. Cuatro paquetes viajando en distintas direcciones tratando de dar vuelta a la izquierda y terminan en un ciclo. Si solo uno de los paquetes no girara, el *deadlock* se evitaría. Se han propuesto algoritmos que no tienen estos problemas, como los basado en cambios de dirección [13, 6].

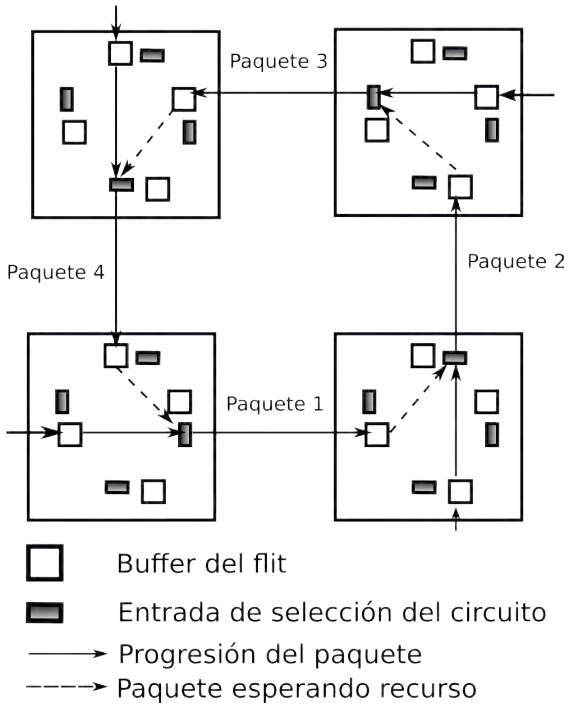

Figura 2.3: Un bloqueo mutuo en que se involucran 4 ruteadores y 4 paquetes en una malla bidimensional [6].

Este modelo implica el análisis de las direcciones en que los paquetes pueden tener en la red y los ciclos que los cambios de dirección pudieran causar y prohibe cambios de dirección que provoquen en los bloqueos mutuos en las dependencias de los ciclos. El algoritmo resultante es libre de bloqueos mutuos(*deadlocks*) y de *livelocks* [4], es mínimo/ no mínimo y altamente adaptativo. Si el algoritmo es mínimo es porque se restringe a que se tomen siempre las rutas más cortas, en caso de no ser mínimo hay más opciones de caminos y más posibilidades de llegar a su destino [13, 6].

[4]Un *livelock* se produce cuando la ruta de un paquete nunca conduce a su destino

Los pasos para diseñar un algoritmo de ruteo basado en el modelo de cambios de dirección se construye como sigue:

Paso 1 Dividir los canales de la red en conjuntos de acuerdo a su dirección.

Paso 2 Identificar los posibles cambios de dirección, ignorando giros de 0 grados. Girar a 0 grados solo es posible cuando hay múltiples canales virtuales en una dirección.

Paso 3 Identificar los ciclos que se pueden formar a través de estos cambios de dirección. En general, identificar los ciclos más simples en cada plano de la topología es adecuado.

Paso 4 Se prohibe un cambio de dirección en cada ciclo para prevenir bloqueos mutuos. El cambio de dirección debe ser escogido cuidadosamente para romper cada posible ciclo, incluyendo ciclos complejos no identificados en el paso 3.

Los algoritmos de ruteo encaminan los paquetes a lo largo de conjuntos de canales identificados en el paso 1 y usan solo los cambios de dirección desde un conjunto a otro, permitido por el paso 4, estos caminos están libres de bloqueos mutuos. Esto es porque rompiendo todos los ciclos, prevenimos la espera circular [13, 6].

Prevenir la espera circular significa que es posible numerar los canales de la red, tal que el algoritmo dirija cada paquete a través de los canales, en orden estrictamente decreciente. Esto es que junto con el hecho de que la red contiene un número finito de canales, significa que el paquete alcanzará su destino después de un limitado número de saltos, por lo que este modelo está libre de *livelocks*. Los algoritmos más importantes basados en el modelo de cambio de dirección son *West first*, *North first* y ruteo *Negative first* para redes de malla 2D y algoritmos de ruteo p-cube para redes de hipercubo [13, 6].

Mallas bidimensionales

Formalmente, una malla $n-$dimensional tiene $k_0 \times k_1 \times \ldots k_{n-1}$ nodos, k_i nodos en cada dimensión i, donde $k_i \geq 2$. Cada nodo en X en una malla $n-$dimensional es identificada por n coordenadas, $(x_0, x_1, \ldots, x_{n-1})$, donde $0 \leq x_i \leq k_i - 1$ para cada dimensión de i. 2 nodos X y Y son vecinos si y solo si $x_i = y_i$ para todo i excepto uno, j, donde $y_i = x_j \pm 1$ [13, 6].

Para redes bidimensionales, las dimensiones 0 y 1 se convierten en x y y respectivamente. Las longitudes de las dimensiones k_0 y k_1 se convierten en m y n. Finalmente las direcciones $-x, +x, -y$, y $+y$ se convierten en *west, east, south* y *north*. Las 4 direcciones permiten 8 giros de $90°$: cambios de dirección a la derecha e izquierda cuando se está viajando al oeste, este, sur o norte. Los 8 cambios de dirección forman 2 ciclos simples, como se muestra en la figura 2.4. El algoritmo xy prohibe 4 cambios de dirección, como en la figura 2.5 [13, 6].

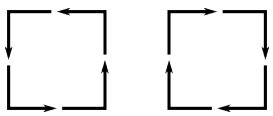

Figura 2.4: Los posibles cambios de dirección y ciclos simples en una malla bidimensional.

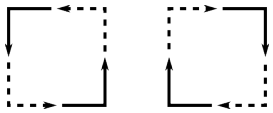

Figura 2.5: Los 4 posibles cambios de dirección permitidos por el algoritmo de ruteo xy

Prohibiendo estos 4 cambios de dirección se previene el bloqueo mutuo, porque los otros 4 giros restantes no pueden formar un ciclo, pero tampoco permiten adaptabilidad en el ruteo. El bloqueo mutuo puede prevenirse prohibiendo menos de 4 cambios de dirección. De hecho, solo 2 cambios de dirección son necesarios para evitar el bloqueo mutuo, uno de cada ciclo, permitiendo así un poco de adaptabilidad en el ruteo [13, 6].

Prohibiendo solo 2 cambios de dirección no se evita el bloqueo mutuo, como se muestra en la figura 2.6. Los 3 cambios de dirección permitidos en la figura 2.6 (a) son equivalentes al giro a la derecha de la figura 2.6(b) y los 3 giros a la derecha permitidos en la figura 2.6(b) son equivalentes a los giros prohibidos a la izquierda en la 2.6(a). Entonces, ambos ciclos permiten que exista un bloqueo mutuo, como se ve en la figura 2.6(c) [13, 6].

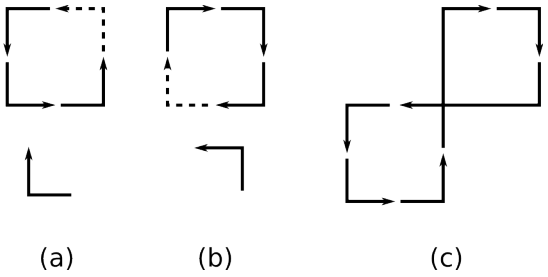

Figura 2.6: Los 6 cambios de dirección que completan ciclos y permiten el bloqueo mutuo **(a)** Los posibles cambios de dirección y ciclos simples en una malla bidimensional. **Inferior:** Los 4 posibles cambios de dirección permitidos por el algoritmo de ruteo xy

Algoritmo planar de ruteo

Es posible reducir el costo de los bloqueos mutuos restringiendo la flexibilidad del ruteo. Los algoritmos planares de ruteo limitan la adaptabilidad a 2 dimensiones en una unidad de tiempo. Reducir la libertad hace posible prevenir bloqueos mutuos con un número fijo de canales virtuales, independientemente del número de dimensiones de la red. Cada ruta es dividida en $n-1$ planos y rutas de paquetes adaptativamente en cada uno de estos planos [13].

$$path_i = A_0 \rightarrow A_1 \rightarrow A_2 \rightarrow \ldots \rightarrow A_{n-3} \rightarrow A_{n-2} \rightarrow A_{n-1} \tag{2.4}$$

$A_i = d_i + d_{i+1}$ El paquete en el plano A_i será enviado en el plano A_{i+1} siempre que el recorra la dimensión d_i. Este algoritmo requiere 3 canales virtuales por cada canal físico para prevenir un bloqueo mutuo. Cada plano, A_i es definido como la combinación de varios conjuntos de canales virtuales, como sigue: $A_i = d_{i,2} + d_{i+1} + d_{i+1,1}$, donde $d_{i,j}$ especifica el conjunto j de canales virtuales de la dimensión d_i. El algoritmo de ruteo descompone la red en planos adaptativos. El encaminamiento en cada plano es libre de bloqueos mutuos y los ciclos no se pueden formar entre los planos. Entonces, el algoritmo es

libre de bloqueos mutuos. Como se muestra en la figura 2.7, cada plano adaptativo A_i es dividido en 2 subredes virtuales las cuales están completamente separadas como: Red decreciente $A_{i+} = d_{i,2+}, d_{i+1,0}$ y la red creciente $A_{i-} = d_{i,2-}, d_{i+1,1}$ [13].

Los paquetes en el plano $A_{i+}(A_{i-})$ puede solo usar el canal virtual 2 de dimensiones $d_{i+}(d_{i-})$. Si suponemos que un ciclo se forma en la red decreciente. Como la malla es rectangular, el ciclo debe tener uno de los A_{i-} canales virtuales pero los paquetes de la red decreciente no pueden usar ninguno de los A_{i-} canales virtuales, entonces no se puede formar un ciclo. Un argumento simétrico se puede usar para la red creciente. Cada plano adaptativo A_i encamina el tráfico a A_{i+1}. Asumamos que un ciclo de tamaño $m + 1$ ocurre: $ciclo = A_{c1}, A_{c2}, \ldots, A_{cm}, A_{cl}$ [13].

Para cada liga en el ciclo, los planos adaptativos son atravesados en orden numérico. Esto significa que en el ciclo debemos tener $C_i < C_{i+1}$ consecuentemente $C_m < C_l$. Sin embargo, el operador $<$ es transitivo y recorriendo el ciclo, resulta que $C_l < C_m$ lo cual es una contradicción. Entonces, no se pueden formar ciclos y el encaminamiento entre planos adaptativos está libre de bloqueos mutuos [13].

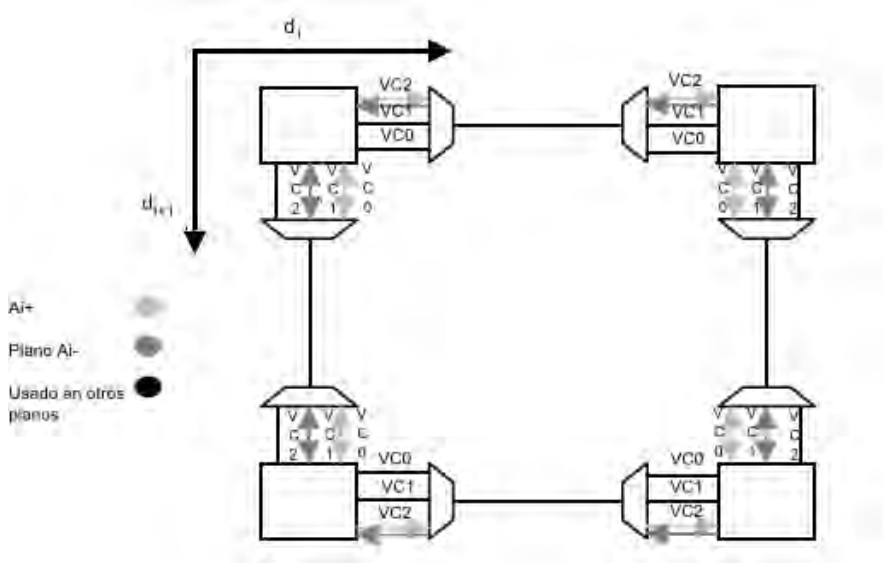

Figura 2.7: Localización de los canales virtuales en el plano A_i en el algoritmo de ruteo adaptativo. Los canales no usados serán usados en los previos y siguientes planos

Conmutación determinista dinámicamente adaptativa

Los ruteadores generalmente se clasifican como deterministas y adaptativos. La técnica de ruteo es llamada adaptativa, sí dada una dirección fuente y destino, la ruta elegida para un paquete en particular depende de las condiciones dinámicas de la red. Los ruteadores adaptativos evitan enlaces congestionados usando rutas alternas, esto incrementa el rendimiento de la red. De cualquier forma, debido a la lógica extra necesaria para encontrar una buena ruta, la latencia se incrementa cuando hay bajos niveles de congestionamiento. El esquema llamado DyAD (*Dinamic Adaptative Deterministic switching*) se basa en el estado actual del congestionamiento de la red; es decir, cada ruteador en la red, monitorea continuamente su red local, su carga y toma decisiones en torno a esta información. Cuando la red no está congestionada, la red trabaja de manera determinista. Sin embargo, cuando la red está congestionada

el DyAD regresa y se evitan los enlaces congestionados, explorando rutas alternativas [7].

DyAD es un paradigma para el diseño de ruteadores para NoC (*Network operation center*), los cuáles aprovechan las ventajas del encaminamiento adaptativo y determinista. Por estas características estos ruteadores pueden construirse utilizando cualquier esquema de encaminamiento adaptativo. Dada su popularidad, la plataforma de estos ruteadores se compone de un arreglo de $n \times n$ celdas las cuales están interconectadas por un red *mesh*. En esta arquitectura regular, cada celda se compone de un elemento de procesamiento y un ruteador. El ruteador se encuentra embebido sobre cada celda, conectada a 4 celdas vecinas y los canales de su elemento de procesamiento. Cada canal consiste en 2 enlaces direccionales punto a punto entre 2 ruteadores o ruteador- elemento de procesamiento [7].

2.2.4. Algoritmos de ruteo adaptativo

Los algoritmos de ruteo dinámicos cambian el camino según los cambios en el tráfico de la red o de la topología. Un algoritmo dinámico puede ser ejecutado ya sea de manera periódica o de manera directa con respuesta a los cambios de topología o cambio en los costos de enlace, estos algoritmos son más susceptibles a problemas tales como oscilaciones en las rutas, y lazos en las rutas.

Algoritmo de Bellman-Ford

Este algoritmo usa la simple idea de calcular las rutas más cortas entre 2 nodos en una forma centralizada. En un ambiente distribuido, un enfoque de vector de distancia es tomado como calcular las rutas más cortas [10, 3].

Usaremos 2 nodos genéricos para explicar el algoritmo, etiquetados como i y j, en una red de N nodos, algunos estarán directamente conectados, como los nodos 4 y 6 de la Figura 2.8. Como se ve en la

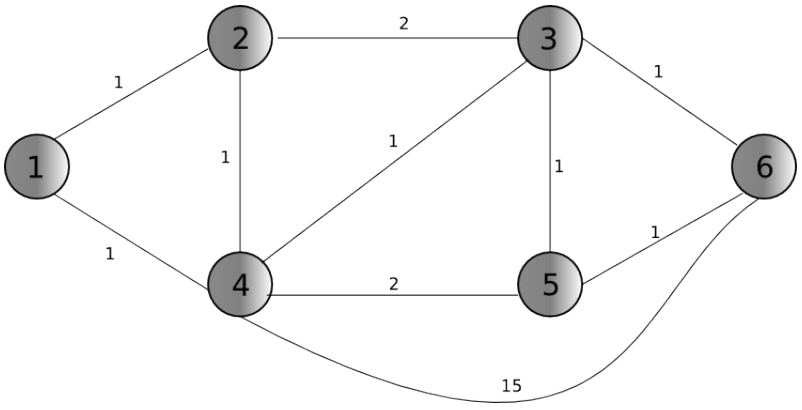

Figura 2.8: Red de 6 nodos

Figura 2.8, no todos los nodos están conectados directamente, en estos casos para encontrar la distancia entre 2 nodos es necesario pasar por otros nodos y enlaces [10].

Definamos la siguiente notación para el costo de los nodos:

$d_{i,j}$ =Costo del enlace entre los nodos i y j

$\overline{D_{i,j}}$ = Costo del cálculo de la ruta mínima desde el nodo i al nodo j.

Si 2 nodos están directamente conectados, entonces, el costo del enlace $d_{i,j}$ toma un valor finito.

Considerando de nuevo la figura 2.8, los nodos 4 y 6 están directamente conectados con un costo de 15, podemos escribir $d_{4,6} = 15$, sin embargo, los nodos 1 y 6 no están directamente conectados, entonces $d_{1,6} = \infty$ [10].

La diferencia entre $d_{i,j}$ y $\overline{D_{i,j}}$, es que los nodos 4 y 6, el costo mínimo actual es de 2, el cual tiene la ruta $4 - 3 - 6$, es decir, $\overline{D_{4,6}} = 2$, mientras que $d_{4,6} = 15$, para los nodos 1 y 6, encontramos que $\overline{D_{1,6}} = 3$, mientras que $d_{1,6} = \infty$ [10].

Por lo tanto, el costo mínimo de la ruta puede ser obtenido entre 2 nodos en una red, sin importar si están o no directamente conectados, al menos que algún nodo esté totalmente aislado de la red. Ahora lo importante es como calcular costo mínimo entre 2 nodos de la red [15].

Consideremos un nodo genérico k en la red, el cual está directamente conectado a cualquier nodo final, asumimos que k esta directamente conectado al nodo j, es decir $d_{k,j}$ tiene un valor finito [10].

Las siguientes ecuaciones, son conocidas como las ecuaciones de Bellman, que deben ser satisfechas por la ruta más corta del nodo i al nodo j:

$$\overline{D_{i,j}} = 0, \forall i \tag{2.5}$$

$$\overline{D_{i,j}} = min_{k \neq j}\{\overline{D_{i,j}} + d_{k,j}\} \text{para } i \neq j \tag{2.6}$$

Una variación del algoritmo se usa cuando el costo mínimo se obtiene por iteraciones del número de saltos. Específicamente definimos el termino para el costo mínimo en términos del número de saltos h, como sigue:

$\overline{D_{i,j}}^{h}$= Costo de la rutas de mínimo costo desde el nodo i al nodo j cuando más de h número de saltos es considerado [10].

Este algoritmo que itera en términos del número de saltos se define como :

Definición 1 ***Algoritmo de Bellman-Ford***
Inicializar todos los nodos i y j en la red:

$$\overline{D_{i,i}}^{(0)} = 0, \forall i;\ \overline{D_{i,j}}^{(0)} = \infty, \textit{para } i \neq j \tag{2.7}$$

Para $h = 0$ hasta $N - 1$, hacer:

$$\overline{D_{i,i}}^{(h+1)} = 0, \forall i \tag{2.8}$$

$$\overline{D_{i,j}}^{(h+1)} = min_{k \neq j}\{\overline{D_{i,k}}^{(h)} + d_{k,j}\}, \textit{para } i \neq j \tag{2.9}$$

Algoritmo de Dijkstra

Este algoritmo calcula la ruta más corta para todos los destinos desde una fuente, en vez de solo para un par especifico de nodos en un tiempo. Consideremos un nodo genérico i en una red de N nodos desde donde queremos calcular las rutas más cortas a todos los nodos en la red. La lista de N nodos se denotará por $N = \{1, 2, \ldots, N\}$. Un destino genérico será denotado como j ($j \neq i$). Usaremos los siguientes términos:

$d_{i,j}$ =Costo del enlace entre los nodos i y j.

$\underline{D}_{i,j}$ = Costo del cálculo de la ruta mínima desde el nodo i al nodo j.

El algoritmo divide la lista de N nodos en 2 listas: comienza con una lista permanente S, la cual representa los nodos ya considerados y una lista tentativa S' de nodos que aun no han sido considerados [10].

Mientras el algoritmo progresa, la lista S se expande con nuevos nodos incluidos, mientras que la lista S' disminuye cuando un nuevo nodo es incluido en S, éste es borrado de S', el algoritmo para cuando la lista S' está vacía. Inicialmente, tenemos $S = \{i\}$ y $S = N \backslash \{i\}$ [10, 3].

El algoritmo tiene 2 partes:

- Como expandir las listas
- Como calcular la ruta mínima a los nodos que son vecinos de los nodos de la lista S.

La lista S se expande en cada iteración, considerando un nodo vecino k del nodo i con la ruta menos costosa desde el nodo i. En cada iteración, el algoritmo considera la vecindad de nodos k, la cual no está aun en S, para ver si el costo mínimo cambia desde la última iteración [10].

Usando el ejemplo de la Figura 2.8. Suponemos que el nodo 1 quiere encontrar las rutas más cortas hacia todos los nodos de la red. Inicialmente $S = \{1\}$ y $S' = \{2, 3, 4, 5, 6\}$ y las rutas más cortas hacia todos los nodos que son vecinos directos de el nodo que pueden ser fácilmente encontrados, mientras que el costo de los nodos restantes es ∞, i.e.

$$\underline{D}_{1,2} = 1, \underline{D}_{1,4} = 1, \underline{D}_{1,3} = \underline{D}_{1,5} = \underline{D}_{1,6} = \infty \tag{2.10}$$

Para la siguiente iteración, notamos que el nodo 1 tiene 2 vecinos directos: el nodo 2 y el nodo 4 con $d_{1,2} = 1$ y $d_{1,4} = 1$ respectivamente, todos los otros nodos no están directamente conectados al nodo 1, entonces el costo directo sigue siendo ∞. Como los nodos 2 y 4 son vecinos con el mismo costo mínimo, podemos escoger cualquiera de los 2. Supongamos que escogemos el nodo 2, entonces tenemos $S = \{1, 2\}$ y $S' = \{3, 4, 5, 6\}$. Ahora preguntamos al nodo 2 el costo a sus vecinos directos que ya están en el conjunto S. En la Figura 2.8 se nota que los vecinos del nodo 2 son el nodo 3 y 4. Entonces, comparamos y calculamos el costo desde el nodo 1 para estos 2 nodos y vemos si hay alguna mejora:

$$\underline{D}_{1,3} = min\{\underline{D}_{1,3}, \underline{D}_{1,2} + d_{2,3}\} = min\{\infty, 1 + 2\} = 3 \tag{2.11}$$

$$\underline{D}_{1,4} = min\{\underline{D}_{1,4}, \underline{D}_{1,2} + d_{2,4}\} = min\{1, 1 + 1\} = 1 \tag{2.12}$$

No hay ninguna mejora en el costo del nodo 4, entonces mantenemos la ruta más corta original. Para el nodo 3, ahora tenemos la ruta mínima : 1-2-3. Para el resto de los nodos, el costo sigue siendo ∞. Esto completa esta iteración. Seguimos con la siguiente iteración y encontramos que el siguiente intermediario es $k = 4$ y el proceso continua como se explicó anteriormente. La generalización de este algoritmo se puede apreciar en la definición 2 [10].

Definición 2 ***Algoritmo de Dijkstra para la ruta más corta***

1. *Comenzar con un nodo fuente i en la lista de nodos permanentes ya considerados, i.e., $S = \{i\}$; el resto de los nodos se ponen en la lista de los nodos tentativos, etiquetada como S'. Iniciar*

$$\underline{D}_{i,j} = d_{i,j}, \forall j \in S' \tag{2.13}$$

2. *Identificar un nodo vecino (intermediario) k (que no esté en la lista S) con el costo mínimo de la ruta desde el nodo i, i.e., encontrar $k \in S'$ tal que*

$$\underline{D}_{j,k} = min_{min \in S'} \underline{D}_{i,m} \tag{2.14}$$

Agregar k a la lista de nodos permanentes S, i.e., $S = S \cup \{k\}$,
Quitar a k de la lista de nodos tentativos S', i.e.,$S = S \backslash \{k\}$,
Si S' está vacía, parar.

3. *Considerar la lista de nodos vecinos, N_k, de los nodos intermediarios k (pero no considerar los que están en S) para revisar por una mejora en el costo de la ruta más corta, i.e., para $j \in N_k \cap S'$*

$$\underline{D}_{i,j} = min\{\underline{D}_{i,j}, \underline{D}_{i,k} + d_{k,j}\} \tag{2.15}$$

[10], [3]

Ruteo óptimo alternante

Considera cualquier nodo fuente y cualquier nodo destino, con n rutas entre ellos, indexados por $i = 1, 2, \cdots, n$. El problema más simple y óptimo de distribuir el tráfico en el nodo de entrada a las rutas disponibles para minimizar el retardo promedio desde el momento que la fuente entra hasta que llega a su destino [1].

En este modelo se asume que los mensajes llegan con una función de densidad de probabilidad de Poisson al nodo fuente con una razón γ por segundo y tienen longitudes distribuidas exponencialmente de $\frac{1}{\mu}$ bits. Los mensajes son puestos en una cola de uno de los n *buffers*, correspondientes a las rutas de salida. Se asume que cada *buffer* tiene capacidad infinita. En cada ruta el el primer enlace cambia la

capacidad de H_i bps, $i = 1, 2, \cdots, n$, el tiempo gastado en el nodo fuente por mensaje asignado a la i-ésima ruta es T_i segundos, este valor es calculado como :

$$T_i = \frac{1}{\mu H_i - \lambda_i} \tag{2.16}$$

Donde λ_i es el radio en el cual el tráfico es asignado a la i-ésima ruta en mensajes por segundo [1].

Adicionalmente a T_i, los mensajes al tomar la i-ésima ruta tienen un retardo adicional de T_i' después de que dejan el nodo fuente. Este retardo incluye el retardo de propagación y procesamiento y retardos en nodos intermedios de la ruta i, entonces la suma de $T_i + T_i'$ es el tiempo de transmisión sobre la ruta i. El retardo de transmisión promedio está dado por la fórmula:

$$T = \frac{1}{\gamma} \sum_{i=1}^{n} \lambda_i (T_i + T_i') \tag{2.17}$$

El problema de ruteo óptimo es minimizar T, sujeto a las restricciones:

$$\gamma = \sum_{i=1}^{n} \lambda_i, 0 \leq \lambda_i \leq \mu H_i, i = 1, 2, \cdots, n \tag{2.18}$$

Para este algoritmo se usa el teorema de Kuhn-Tucker [5] para estudiar las condiciones en las cuales el ruteo adaptable es satisfecho [1].

Al menos que el nodo fuente se encuentre saturado (i.e. $\gamma \geq \sum_i \mu H_i$), es fácil ver que $\lambda_i < \mu H_i$ se mantiene para todas las i porque un encaminamiento óptimo debe mantener cualquier enlace proveniente de un nodo saturado, antes que cualquier otro enlace [1].

En este esquema, en vez de que cada nodo en la red mantenga una tabla con una entrada para cada posible destino y para cada posible vecino; la entrada en la tabla para la localización (i, j) es el tiempo estimado por un mensaje enviado vía el i-ésimo vecino para alcanzar el j-ésimo destino, incluyendo el tiempo perdido en el nodo que envía la espera de transmisión. La tabla se mantiene actualizada por los vecinos, que periódicamente intercambian sus estimaciones de tiempo mínimo para llegar a cada destino y usan esta información para actualizar la tabla de ruteo de cada nodo. Cuando sólo hay un destino, la tabla de ruteo en el nodo que envía, es un vector cuyo elemento i-ésimo es T_i', más una función lineal da una estimación del tiempo de espera en el *buffer* $T_i = (1 + L_i)/\mu H_i$, donde hay L_i mensajes en el *buffer* (incluyendo el que está en servicio) [1].

La tabla se usa de la siguiente manera: a su llegada al nodo receptor, el mensaje es reconocido y el nodo consulta la dirección de destino del mensaje. Si es la j-ésima dirección, el nodo consulta la columna j de la tabla de ruteo, elige la entrada mínima en la columna y coloca el mensaje en la cola de la línea de transmisión asociada. Por último, el nodo actualiza la tabla de ruteo para reflejar el nuevo estado de la cola. Podemos pensar en el nodo como asignar el mensaje a la ruta i para la que el valor actual de $T_i + T_i'$ es un mínimo [1].

Haciendo referencia a las hipótesis formuladas en la búsqueda de la ruta óptima, es decir, las llegadas con la función de densidad con distribución de Poisson y los mensajes de longitud exponencial,

[5] son condiciones necesarias y suficientes para que la solución de una programación no lineal sea óptima. Es una generalización del método de los Multiplicadores de Lagrange

se tiene que el tiempo de transmisión a través de varias rutas deben satisfacer las siguientes condiciones promedio:

$$\frac{1}{(\mu H_i - \lambda_i) + T_i} = \frac{1}{(\mu H_j - \lambda_j) + T'_j} \text{ si } \lambda_i > 0 \text{ y } \lambda_j > 0 \quad (2.19)$$

$$\frac{1}{(\mu H_i - \lambda_i) + T_i} < \frac{1}{(\mu H_j - \lambda_j) + T'_j} \text{ si } \lambda_i = 0 \text{ y } \lambda_j = 0 \quad (2.20)$$

Esto es que, el tiempo esperado por el mensaje para viajar desde la fuente hasta el nodo destino es el mismo sobre cualquier ruta es el mismo a través de cualquier fuente para la cual el tráfico es asignado y es menor que el tiempo de viaje por una ruta que no tiene tráfico asignado a ella [1].

Algo interesante de esta propiedad de equilibrio del ruteo adaptativo es que no corresponde a las condiciones necesarias satisfechas por el encaminamiento óptimo. Éste es un resultado sorprendente, ya que parecería que un algoritmo en que las rutas de cada mensaje han sido minimizadas con el fin de reducir el retraso de transmisión que, además, para minimizar el óptimo global es que para cada minimización individual tomando en cuenta la tabla de ruteo, tendrían efecto sobre las opciones de futuro. Colocando un mensaje en la cola de cualquier canal de salida agrega un retardo adicional en los subsecuentes mensajes, los cuales llegan antes de que la cola se disipe. Como resultado, los mensajes subsecuentes deben unirse a la cola y esperar una retraso adicional en este nodo para transmitir o seleccionar una ruta alternativa la cual agrega un retardo extra en algún otro sitio de la red [1].

Este esquema de actualizar la tabla de ruteo se llama encaminamiento cuadrático, esto por la forma funcional:

$$S \triangleq (1/\mu H_i)(1 + L_i)(1 + \frac{L_i}{2}) \quad (2.21)$$

La cuál es una función cuadrática de la cola de longitud L_i y las condiciones de equilibrio cambia a:

$$\bar{S}_i + T'_i = \bar{S}_j + T'_j \text{ si } \lambda_i > 0 \text{ y } \lambda_j > 0 \quad (2.22)$$

$$\bar{S}_i + T'_i < \bar{S}_j + T'_j \text{ si } \lambda_i > 0 \text{ y } \lambda_j = 0 \quad (2.23)$$

y corresponde a las condiciones necesarias para nuestra asignación óptima. Entonces el encaminamiento equilibrado es también óptimo [1].

Algoritmos de ruteo utilizando matrices de adyacencia

Existe una investigación realizada en 2008 sobre un protocolo de ruteo basado en matrices de adyacencia en redes móviles Ad Hoc. Esta investigación propone la creación de un nodo agente que recopila la información la topología de la red y crea la matriz de adyacencia de ésta [9].

El nodo agente es un nodo diseñado para implementar el protocolo basado en matrices de adyacencia. Este nodo tiene la misma tabla de ruteo para otros nodos. El nodo agente es el que inicia el proceso de descubrimiento de la topología y puede agrupar la información y enviar a todos la información de la topología de red, en la forma de una matriz de adyacencia [9].

En una red Ad Hoc, todos los nodos periódicamente mandan un mensaje “Hello”, para revisar su conectividad con los nodos vecinos. En este esquema, cada nodo mantiene una tabla de la información

de los vecinos. El nodo agente hace un *broadcast* con una petición a todos los nodos vecinos para que estos a su vez los manden a sus vecinos, así sucesivamente. Cuando se acaban los vecinos siguientes, se manda un mensaje de regreso con la dirección del nodo y la matriz de adyacencia con sus vecinos, cuando recibe el mensaje de respuesta, el nodo agente puede obtener la matriz de adyacencia de un salto de cada nodo en toda la red. Puede suceder que un nodo reciba más de un mensaje, en este caso, el nodo checa el número de secuencia del mensaje y almacena o actualiza la matriz de adyacencia con el número de secuencia mayor [9].

2.2.5. Protocolos de ruteo

Los protocolos de ruteo determinan la ruta óptima a través de la red usando algoritmos de ruteo e información de transporte sobre estas rutas. Los protocolos de ruteo funcionan en la capa de red del modelo de referencia OSI. Ellos usan información específica de la capa de red, incluyendo direcciones de red, para mover unidades de información a través de la red. Nuestro problema se encuentra ubicado en la capa de aplicación del modelo de referencia OSI.

Vector de distancia (RIP)

La idea detrás de el algoritmo de vector de distancia es sugerido por su nombre [6]. Cada nodo construye un arreglo unidimensional (un vector) de conteniendo las distancias (costos) hacia todos los nodos y distribuye ese vector a sus vecinos inmediatos. Inicialmente se asume que cada nodo conoce el costo del enlace a cada uno de sus vecinos inmediatos, un enlace caído se considera con costo infinito [15].

El protocolo de ruteo en redes IP Routing Information Protocol (RIP) es el ejemplo canónico de un protocolo de Internet construido a partir del algoritmo de Bellman-Ford descrito anteriormente [15].

Los protocolos de ruteo entre redes difieren de el modelo del grafo que se describió con anterioridad. En una red de redes, la meta de los ruteadores es aprender como reenviar lo paquetes a varias redes, por lo tanto, en vez de considerar el costo de llegar a otros ruteadores, los ruteadores de difunden el costo alcanzar la red. Por ejemplo, en la Figura 2.9, el ruteador C debe informar al ruteador A del hecho de que se pueden alcanzar las redes 2 y 3 (que están directamente conectadas) con un costo de 0, las redes 5 y 6 tienen costo de 1; las redes 4 y 2 tienen costo de 2 [15].

RIP es la implementación más directa del ruteo por vector de distancia. Los ruteadores que utilizan RIP envían sus avisos cada 30 segundos, el ruteador también envía sus avisos cuando hay algún cambio en su tabla de ruteo. En la práctica, el protocolo busca alcanzar el objetivo en menos de 16 saltos, de hecho 16 saltos es equivalente a la distancia infinita [15].

[6] Esta basado en el algoritmo de Bellman-Ford

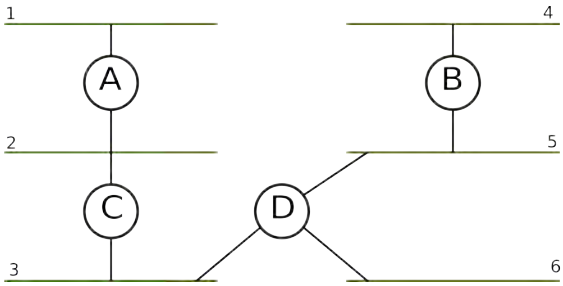

Figura 2.9: Ejemplo de una red ejecutando RIP

Estado de enlace (OSPF)

El ruteo por estado de enlace es el segundo más usado en los protocolos de ruteo. Las consideraciones iniciales son las mismas que en el de vector de distancia. Se asume que cada nodo es capaz de detectar el estado de los enlace que lo comunican con los vecinos inmediatos (conectado o desconectado) y el costo de cada enlace [15].

La idea es simple, cada nodo conoce como llegar directamente a su vecinos inmediatos y para tener el conocimiento total, lo único que hay que hacer es difundir esta información por toda la red, cuando cada nodo tenga toda la información será capaz de armar su propio mapa de la red entera. Este protocolo está constituido de 2 mecanismos: la diseminación segura de la información del estado de los enlaces y el cálculo de rutas desde la suma de todos los estados de los enlaces conocidos [15].

En la práctica, cada *switch* calcula su propia tabla de ruteo independientemente desde el paquete de estado del enlace [7] y es usado para la realización del algoritmo de Dijkstra llamado algoritmo se búsqueda hacia adelante, que mantiene 2 listas que contiene un conjunto de entradas de la forma (Destino, Costo, Siguiente Salto), este protocolo opera como se explicó en la subsección 2.2.4, [15].

The Open Shortest Path First Protocol (OSPF) es uno de los estándares más utilizados, sobre todo por ser un estándar no propietario [15].

2.2.6. Protocolos de ruteo en redes móviles

Entre la gran variedad de algoritmos de ruteo para redes móviles, no es posible aplicarlos eficientemente a nuestro problema, pues no consideran el estado del planificador de cada uno de los nodos.

AODV (*Ad hoc On Demand Distance Vector*)

Es un protocolo de ruteo diseñado para redes Ad Hoc. Este protocolo es capaz de realizar enrutamiento unicast y multicast. Es un protocolo reactivo, es decir, solo establece una ruta a un destino si esta se le solicita [20].

[7] LSP: Cada nodo crea un paquete de actualización, que contiene el ID del nodo, una lista de los vecinos directamente conectados y el costo de los enlaces, un número de secuencia y el tiempo de vida del paquete

El algoritmo AODV permite a los nodos móviles obtener rutas rápidamente para nuevos destinos, y no exige a los nodos mantener las rutas a los destinos que no están en la comunicación activa. Adicionalmente, AODV permite para la formación de grupos multicast donde el número de miembros es libre de cambiar durante la vida de la red. AODV también define las contestaciones para roturas y cambios en la topología de la red. El funcionamiento de AODV es libre de ciclos, y evita el problema Bellman-Ford "contando al infinito" ofrece la convergencia rápida cuando la topología de la red cambia (típicamente, cuando un nodo se mueve en la red) [20].

Una característica de AODV es el uso de un número de secuencia destino para cada entrada de la tabla de ruteo. El número de secuencia destino es creado por el destino o el *grouphead* del *multicast* para cualquier información de la ruta utilizable que envía a pedir los nodos. Usando el número de secuencia destino aseguran que sea libre de ciclos. Dado la opción entre dos rutas a un destino, un nodo solicitante siempre selecciona el que tenga un número de secuencia más grande [20].

Otra característica de AODV es que las roturas en las conectividades causan de inmediato notificaciones, enviado al conjunto de nodos afectados, pero sólo a ese conjunto de nodos [20].

Route Requests (RREQs), *Route Replies* (RREPs) y *Multicast Route Invalidations* (MINVs) son los tres tipos del mensaje definidos por AODV. Estos tipos mensajes son manejados por UDP e IP normal. Así, por ejemplo, el nodo solicitante espera usar su dirección IP como la IP fuente para los mensajes. El rango de alcance del *broadcast* para los RREQs puede ser indicado por el TTL en la cabeza de IP. La fragmentación es típicamente no requerida [20].

Ya que los *endpoints* de una conexión de comunicación tengan rutas válidas para cada uno, AODV no juega ningún papel. Cuando una ruta a un nuevo destino (un solo nodo o un grupo *multicast*) es necesaria, el nodo usa una transmisión *broadcast* y manda un RREQ para encontrar una ruta al destino. Una ruta puede determinarse cuando la solicitud alcanza ya sea, el destino mismo o un nodo intermedio con una ruta actual hacia el destino. La ruta es hecha disponible para mandar de forma unicasting un RREP hacia a la fuente del RREQ. Desde que cada nodo que recibe la solicitud (un RREQ) guarda una ruta hacia la fuente de la solicitud, el RREP puede ser unicast del destino a la fuente, o de cualquier nodo del intermedio que puede satisfacer la solicitud de la fuente. En el escenario *multicast*, RREQs son también usados cuando un nodo desea unirse a un grupo *multicast*. Una bandera en el RREQ permite los nodos saber que cuando ellos reciben el RREP, ellos no están poniendo simplemente indicadores de la ruta, sino realmente están insertando una rama al árbol del *multicast* [20].

OLSR (*Optimized Link State Routing*)

Es un protocolo de ruteo pro-activo, que trabaja en forma distribuida para establecer las conexiones entre los nodos en una red inalámbrica ad hoc. La diseminación directa de información por toda la red (*flooding*) es ineficiente y muy costosa en una red inalámbrica y móvil, debido a las limitaciones de ancho de banda y la escasa calidad del canal. OLSR provee un mecanismo eficiente de diseminación de información basado en el esquema de los *Multipoint Relays* (MPR) [8].

Bajo este esquema, en lugar de permitir que cada nodo retransmita cualquier mensaje que reciba (*flooding* clásico), todos los nodos de la red seleccionan entre sus vecinos un conjunto de *multipoint relays* (retransmisores), encargados de retransmitir los mensajes que envía el nodo en cuestión. Los

demás vecinos del nodo no pueden retransmitir, lo que reduce el tráfico generado por una operación de *flooding* [8].

Hay varias formas de escoger los *multipoint relays* de un nodo, pero independientemente de la forma de elección, el conjunto de MPRs de un nodo debe verificar que son capaces de alcanzar a todos los vecinos situados a una distancia de 2 saltos del nodo que los calcula (criterio de cobertura de MPR) [8].

Una red enrutada con OLSR utiliza básicamente dos tipos de mensajes de control:

- Los mensajes HELLO son enviados periódicamente por cada nodo de la red a sus nodos vecinos, pero nunca son retransmitidos más allá del primer salto (1 hop) desde su emisor (alcance local). Estos mensajes contienen la lista de vecinos conocidos por el nodo emisor así como la identidad de los *multipoint relays* seleccionados por transmisor. Su intercambio permite a cada nodo de la red conocer los nodos situados a 1 y 2 saltos de distancia (es decir, aquellos a los que se puede hacer llegar un mensaje con una transmisión directa o con una transmisión y una retransmisión) y saber si ha sido seleccionado como MPR por alguno de sus vecinos [8].
- Los mensajes TC (*Topology Control*) son enviados periódicamente y de forma asíncrona. A través de ellos, los nodos informan al conjunto de la red acerca de su topología cercana. Al contrario que los HELLO, los mensajes TC son de alcance global y deben llegar a todos los nodos de la red. El conjunto de los mensajes TC recibidos por un nodo inalámbrico le permite reconstruir su base de datos topológica, computar el árbol de caminos mínimos (mediante el algoritmo de Dijkstra) y calcular así la tabla de enrutamiento hacia todas las posibles destinaciones. La diseminación de mensajes TC se hace de acuerdo con el mecanismo de *flooding* basado en MPR [8].

TORA (*Temporally-Ordered Routing Algorithm*)

TORA es un protocolo de ruteo distribuido para redes móviles e inalámbricas; este protocolo apuesta por un alto grado de escalabilidad usando un "flat", es decir, un algoritmo de ruteo no jerárquico. En la operación del algoritmo, se intenta suprimir la propagación de mensajes de control a los más lejanos niveles de la red. TORA construye y mantiene un grafo acíclico dirigido con raíz en el destino [12].

TORA es un protocolo complicado (utiliza un reloj lógico para establecer un orden temporal del cambio topológico de la red), su principal característica es que cuando un enlace falla los mensajes de control solo se propagan alrededor del punto de falla [12].

El protocolo TORA, es un protocolo reactivo, se caracteriza por proporcionar al nodo remitente, no sólo uno, sino múltiples trayectos para hacerle llegar al destino. El procedimiento es el siguiente: cada nodo realiza una copia de TORA para cada destino y el protocolo crea, mantiene y cancela los trayectos de enrutamiento. El TORA asocia un peso a cada nodo de la red respecto a un destino, y los mensajes se desplazan desde un nodo con mayor peso hacia uno con peso menor; mientras los caminos descubiertos con paquetes de tipo QRY (*Query*) vienen actualizados con aquellos de tipo UPD (*Update*) [12].

Si un nodo necesita conocer un trayecto hacia un destino manda en difusión (*broadcast*) un paquete QRY que se propaga, hasta que no alcanza el nodo destinatario o a un nodo que posea un trayecto

válido hacia el destino. El nodo que responda se servirá a su vez de un paquete UPD que agregará también su peso. Los paquetes UPD se enviarán en difusión de modo que permitan a todos los nodos intermedios modificar su peso convenientemente. Se deriva, por tanto, que los nodos que quieran alcanzar destinos lejanos o directamente inalcanzables, aumentan su peso local hasta el máximo valor consentido, mientras que el nodo que encuentre un nodo cercano con un peso que tienda a infinito, cambiará el trayecto [12].

El paquete de tipo CLR (Clear) interviene en algunos casos para resetear todos los estados de direccionamiento de una porción de red cuando el destino sea completamente inalcanzable [12].

El protocolo TORA se apoya en el protocolo para redes MANET llamado IMEP (*Internet MANET Encapsulation Protocol*) que proporciona un servicio de expedición fiable para protocolos de enrutamiento [12].

La agregación en un único bloque de los mensajes IMEP y TORA reduce el *overhead* de la red y prueban el estado de los nodos vecinos. Para obtener tal agregación se utiliza periódicamente un intercambio de mensajes llamados BEACON y HELLO [12].

2.2.7. Métricas de desempeño

Sea un sistema con N nodos de procesamiento, numerados desde 0 a $N-1$. La demanda de tráfico es descrita como una matriz de tráfico de $N \times N$, que llamaremos TM. Cada entrada $tm_{i,j}$ en $TM, 0 \leq i \leq N-1, 0 \leq j \leq N-1$, denota la cantidad de tráfico desde el nodo i hasta el nodo j [21].

Un encaminamiento especifica como el tráfico de cada par fuente-destino (FD) es encaminada a través de la red. Consideremos un encaminamiento de ruta simple, donde solo una ruta puede usarse para cada par FD y encaminamiento múltiple cuando se utilizan múltiples rutas. En encaminamiento múltiple, cada ruta para un par FD encamina una fracción de el tráfico para el par FD [21].Un encaminamiento múltiple se caracteriza por un conjunto de rutas

$$MP_{i,j} = \{MP_{i,j}^1, MP_{i,j}^2, \ldots, MP_{i,j}^{MP_{i,j}}\} \tag{2.24}$$

para cada par $FD\ (i,j)$, y la fracción de tráfico encaminada a través de la red

$$fraccion_{i,j} = \{fraccion_{i,j}^k | k = 1, 2, \ldots, |MP_{i,j}|\}. \sum_{k=1}^{MP_{i,j}} fraccion_{i,j}^k = 1 \tag{2.25}$$

Sea un enlace $l \in MP_{i,j}^k$. La contribución de el tráfico $tm_{i,j}$ al link l a través de la ruta $MP_{i,j}^k$ es $tm_{i,j} \times fraccion_{i,j}^k$. El encaminamiento de ruta simple es un caso especial del encaminamiento múltiple donde $MP_{i,j} = 1$ y todo el tráfico desde el nodo i al nodo j es encaminada a través de $MP_{i,j}^1 (fraccion_{i,j}^1 = 1)$. Por lo tanto, un encaminamiento de ruta simple puede ser especificado por una ruta $MP_{i,j}^1$ para cada par $FD\ (i,j)$ [21].

Para una matriz de tráfico dada, el desempeño de un encaminamiento es medido por la carga máxima del enlace. Cuando los enlaces tienen la misma capacidad, es posible decir que la carga máxima del enlace es equivalente a la máxima utilización del enlace [21].

Sea $Links$ el conjunto de todos los enlaces en al red. Para encaminamiento múltiple mr, la carga máxima del enlace está dada por

$$MLOAD(mr, TM) = max_{l \in Links}\{ \sum_{i,j,k \text{ tal que } l \in MP^1_{i,j}} tm_{i,j} \times fraccion^k_{i,j}\} \tag{2.26}$$

Para encaminamiento de ruta simple sr, la fórmula se simplifica a

$$MLOAD(sr, TM) = max_{l \in Links}\{ \sum_{i,j,k \text{ tal que } l \in P^1_{i,j}} tm_{i,j}\} \tag{2.27}$$

Un encaminamiento óptimo para una matriz de tráfico TM dada es un encaminamiento que minimiza la carga máxima del enlace. Formalmente, la carga óptima para la matriz de tráfico TM está dada por

$$OPTU(TM) = min_{r \text{ es un encaminamiento}}\{MLOAD(r, TM)\} \tag{2.28}$$

El radio de desempeño de un encaminamiento r en una matriz de tráfico TM mide que tal lejos está r de ser óptima para TM. Esto es definido como la carga máxima del enlace de r dividido por el valor más pequeño posible de la carga del enlace sobre TM.

$$PERF(r, TM) = \frac{MLOAD(r, TM)}{OPTU(TM)} \tag{2.29}$$

$PERF(r, TM)$ es al menos 1. Esto es exactamente 1 si solo si el encaminamiento es óptimo para TM. La definición del radio de desempeño sigue el marco de "análisis competitivo" donde las garantías del desempeño de una solución de ciertos servicios en relación con la mejor solución posible.

El radio de desempeño de un esquema de encaminamiento puede ser mejor cuando la red está bajo carga pesada y peor cuando la carga está bajo cargas ligeras [21].

Es necesario conocer a fondo el funcionamiento de los algoritmo de ruteo en todas sus modalidades, ya que esto nos da la pauta para conocer sus deficiencias y así poder proponer mejorar estás. Entre las debilidades de éstos encontramos que no se considera en ningún momento el estado de planificador de nodo que funcionará como router, es importante tomarlo en cuenta puesto que si el planificador no asigna ni espacio ni memoria para transmitir el mensaje, el nodo no puede cumplir la función de router.

2.3. Representación de redes

Existen múltiples maneras de representar la conectividad de una red, entre éstas un a red puede ser vista como un grafo, a su vez, un grafo puede representarse como con listas de aristas, listas de adyacencia, pero esto se vuelve problemático cuando hay demasiadas aristas en el grafo. Para simplificar el cálculo, los grafos pueden representarse como matrices. Existen 2 tipos de matrices que son las más usadas para representar grafos. Una es la basada en la adyacencia de los vértices y la otra basada en la incidencia de los vértices y aristas [16].

Definición 3 ***Matriz de adyacencia***

Suponemos que $G = (V, E)$ es un grafo simple, donde $|V| = n$. Si suponemos que los vértices de G son listados arbitrariamente como $v_1, v_2, \ldots, v_n$. La matriz de adyacencia A (o A_G) de G, con respecto a la lista de vértices, es la es la $n \times n$ matriz binaria con 1 en su entrada (i, j) cuando v_i y v_j son adyacentes y 0 si no son adyacentes. En otras palabras, si su matriz de adyacencia es $A = [a_{i,j}]$, entonces:

$$a_{i,j} = \begin{cases} 1 & \textit{si } v_i, v_j \textit{ es una arista de } G \\ 0 & \textit{de otra manera} \end{cases} \tag{2.30}$$

Nótese que una matriz de adyacencia está basada en el orden de los vértices. Por lo tanto existen $n!$ diferentes matrices de adyacencia para el grafo con n vértices, dado que hay $n!$ formas posibles de ordenar n vértices [16].

La matriz de adyacencia de un grafo simple es simétrica, que significa que $a_{i,j} = a_{j,i}$, puesto que ambas entradas son iguales. Además el grafo simple no tiene ciclos, entonces cada entrada $a_{i,i}, i = 1, 2, \ldots, n = 0$. Cuando hay pocas aristas en el grafo, la matriz de adyacencia es una matriz dispersa, ya que tiene pocas entradas distintas a 0 [16].

Otra manera de representar grafos es usando matrices de incidencia [16].

Definición 4 ***Matriz de incidencia*** *Sea $G = (V, E)$ un grafo no dirigida. Si suponemos que $v_1, v_2, \ldots, v_n$ son vértices y $e_1, e_2, \ldots, e_m$ son aristas de G. Entonces la matriz de incidencia con respecto al orden de V y E es la matriz $n \times m, M = [m_{i,j}]$, donde*

$$m_{i,j} = \begin{cases} 1 & \textit{cuando la arista } e_j \textit{ es incidente con } v_i \\ 0 & \textit{de otra manera} \end{cases}$$

[16].

Todos estos conceptos son necesarios para entender que pasa con las redes móviles, los algoritmo de ruteo ya implementados, los tipos de planificadores que existen, las formas de representar las redes, todo esto para poder hacer una propuesta novedosa que aproveche los aspectos y características exitosas de éstos y sea posible con todos estos antecedentes elaborar una propuesta que solucione una deficiencia de éstos.

2.4. Algoritmos de rutas más cortas entre pares de nodos

En la literatura existen una gran variedad de algoritmos para encontrar la ruta más corta [10, 3], como ya se mencionaron los algoritmos de Bellman-Ford (sección 2.2.4) y variaciones del algoritmo de Dijkstra 2.2.4, sin embargo, esos calculan la ruta más corta desde un nodo i hasta todos los nodos del

grafo, sin embargo, para nuestro caso de estudio nos es más útil conocer las rutas más cortas de todos los nodos a todos los nodos, esto se puede resolver con el algoritmo de Floyd-Warshall.

2.4.1. El algoritmo de Floyd-Warshall

El algoritmo encuentra el camino entre todos los pares de vértices en una única ejecución. El algoritmo de Floyd-Warshall compara todos los posibles caminos a través del grafo entre cada par de vértices. El algoritmo es capaz de hacer esto con sólo V^3 comparaciones (esto es notable considerando que puede haber hasta V^2 aristas en el grafo, y que cada combinación de aristas se prueba). Lo hace mejorando paulatinamente una estimación del camino más corto entre dos vértices, hasta que se sabe que la estimación es óptima [3].

Sea un grafo G con conjunto de vértices V, tal que $V = 1, \ldots, N$. Sea además una función $caminoMin(i, j, k)$ que devuelve el camino mínimo de i a j usando únicamente los vértices de 1 a k como puntos intermedios en el camino. Dada esta función, el objetivo es encontrar el camino mínimo desde cada i a cada j usando únicamente los vértices de 1 hasta $k + 1$ [3].

Hay dos candidatos para este camino: un camino mínimo, que utiliza únicamente los vértices del conjunto $(1, \ldots, k)$, o bien existe un camino que va desde i hasta $k + 1$, después de $k + 1$ hasta j que es mejor. Sabemos que el camino óptimo de i a j que únicamente utiliza los vértices de 1 hasta k está definido por $caminoMin(i, j, k)$, y está claro que si hubiera un camino mejor de i a $k + 1$ a j, la longitud de este camino sería la concatenación del camino mínimo de i a $k + 1$ (utilizando vértices de $(1, \ldots, k)$) y el camino mínimo de $k + 1$ a j (que también utiliza los vértices en $(1, \ldots, k)$) [3].

Por lo tanto, podemos definir $caminoMin(i, j, k)$ de forma recursiva:

$$caminoMin(i, j, k) = \text{mín}(caminoMin(i, j, k - 1), caminoMin(i, k, k - 1) + caminoMin(k, j, k - 1)) \tag{2.31}$$

$$caminoMin(i, j, 0) = pesoArista(i, j) \tag{2.32}$$

Esta fórmula es la base del algoritmo Floyd-Warshall. Funciona ejecutando primero $caminoMin(i, j, 1)$ para todos los pares (i, j), usándolos para después hallar $caminoMin(i, j, 2)$ para todos los pares (i, j). Este proceso continúa hasta que $k = n$, y habremos encontrado el camino más corto para todos los pares de vértices (i, j) usando algún vértice intermedio [3].

2.5. Resumen

Conociendo las debilidades de los algoritmos de ruteo actuales, los detalles del funcionamiento de los algoritmos de planificación de los nodos, las herramientas matemáticas de análisis de redes, los algoritmos para encontrar las rutas entre pares de nodos, es posible proponer una primera aproximación de un algoritmo que nos permita considerar el estado del planificador de cada nodo en un instante t.

Aunque existen una gran cantidad de algoritmos de ruteo en la actualidad, aún hay problemas no resueltos en esta área, aunque muchos de estos algoritmos incorporan restricciones para evitar ciclos y

bloqueos, no se consideran las posibles limitantes de los nodos al transmitir, si el planificador no tiene espacio disponible.

De los conceptos revisados es este capítulo es posible concluir que en los algoritmos estudiados hace falta considerar las condiciones locales de los nodos, porque cada nodo cuenta con un planificador propio, es cual es necesario tomar en cuenta; el estado de cada uno de estos planificadores determina si un nodo es viable como ruteador o no.

Los estados del planificador de cada nodo junto con las condiciones generales de la red pueden ser representadas por un matriz de adyacencia, ya que de las representaciones revisadas, ésta es la que más de adapta a este caso de estudio.

Capítulo 3

PLANTEAMIENTO DEL PROBLEMA

Cuando estas solucionando un problema,
no te preocupes. Ahora, después de que
has resuelto el problema es el momento
de preocuparse...
Richard Phillips Feynman

En la actualidad, las computadoras personales portátiles han alcanzado bajos costos, además cuentan con un gran equipamiento de hardware, cuentan con varios Gbytes en memoria RAM y ROM, con alta resolución en pantalla, dispositivos de comunicación inalámbrica, entre otras características. Esto ha provocado haya modificado la manera de comunicarlos entre si, dado que estos dispositivos al tener la posibilidad de moverse (es decir salir del rango de alcance de una red inalámbrica) pueden simplemente desaparecer de la red de datos.

En una red, la finalidad es comunicar al nodo A con el nodo B, cuidando la transparencia del sistema distribuido, esto es que el usuario final no sepa cuales fueron los nodos intermedios que visitó antes de llegar a su destino (es decir, que ruta se siguió), cuales eran las condiciones de la red, si se perdieron datos o si hubo que retransmitir.

Sin embargo, aunque el usuario final no se entera de estos detalles, el ruteador debe escoger una ruta por la cual mandar un mensaje, esto lo hace consultando tablas de ruteo (para el caso de ruteo estático) o por medio de un algoritmo de ruteo adaptativo.

Una de las desventajas de los algoritmos de ruteo más usados actualmente, es que se necesita una imagen global de la red para poder determinar la ruta óptima. Es decir, usar un algoritmo de descubrimiento que con cada cambio en la topología de la red, lo que requiere generar el estado global del sistema nuevamente.

3.1. Problema clásico

El problema clásico de ruteo es encontrar una ruta por la cual conectar al nodo A con el nodo B. Suponemos que tenemos una red en la cual sus componentes no varían de estado en el tiempo y su topología es conocida, por lo tanto es posible construir una ruta a través de la red y mantenerla como una "ruta favorita" para esa topología especifica, como se puede ver en la Figura 3.1. En esta figura podemos observar la construcción de una ruta a través de la red

En la Figura 3.1 tenemos representados a los nodos conectados con un circulo y a los no conectados como una "X". En este caso, el estado del nodo es binario, es decir el nodo está conectado o el nodo no está conectado a la red. Podemos determinar la ruta entre 2 puntos ejecutando un algoritmo como el de Dijkstra (Sección 2.2.4) o el algoritmo de Bellman-Ford (Sección 2.2.4) y esta información guardarla en tablas de ruteo que serán actualizadas cada que haya un cambio en la red. Así cuando cuando necesitemos comunicar nuevamente al nodo A con el nodo B solo será necesario consultar la ruta en las tablas.

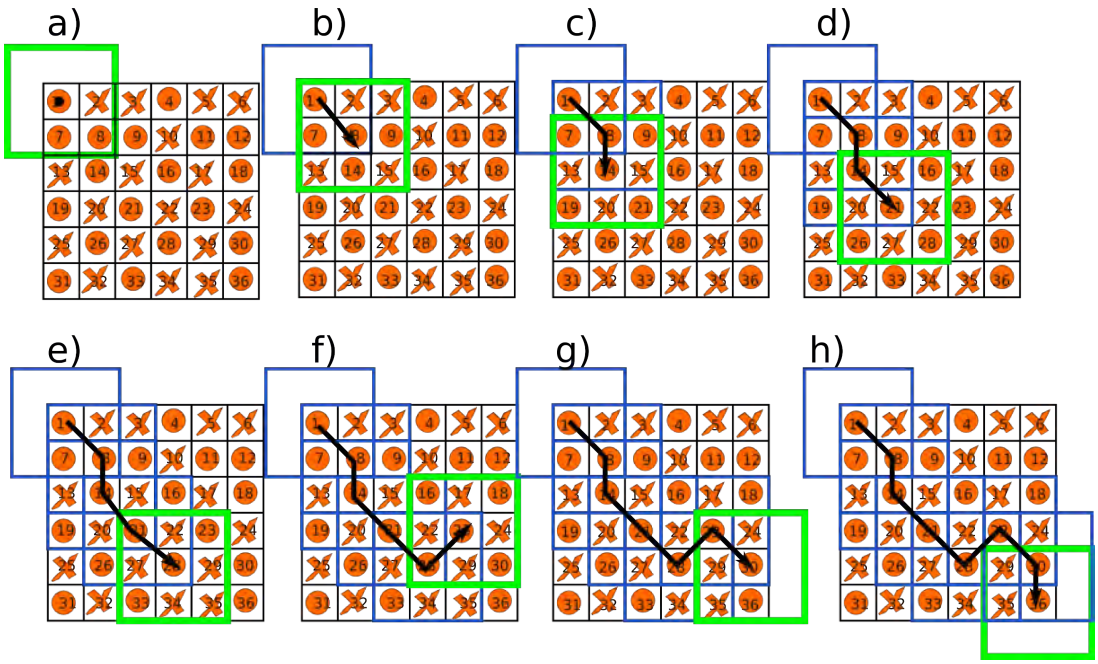

Figura 3.1: Ejemplo del problema clásico de encontrar una ruta a través de la red que no varía en el tiempo.

En la Figura 3.2 vemos la red de la Figura 3.1 representada como grafo, se consideran vecinos las 8 casillas alrededor de cada punto. Utilizando el algoritmo de Dijkstra para rutas mínimas, obtenemos que la ruta mínima del nodo 1 al nodo 36 es de 7 saltos y la ruta es $1 \rightarrow 7 \rightarrow 14 \rightarrow 21 \rightarrow 16 \rightarrow 23 \rightarrow 30 \rightarrow 36$, en nuestro ejemplo la ruta es similar, porque en la Figura 3.1 la ruta seguida fue $1 \rightarrow 8 \rightarrow 14 \rightarrow 21 \rightarrow 16 \rightarrow 23 \rightarrow 30 \rightarrow 36$, lo que no afecta al número de saltos.

Un problema común que tienen los algoritmos más usados, como el algoritmo basado en vector de distancia (sección 2.2.5), es caer en ciclos por no actualizar la información de la red y trabajar con información de estados anteriores. Otro problema con un esquema como el *wormhole* (sección 2.2) en el

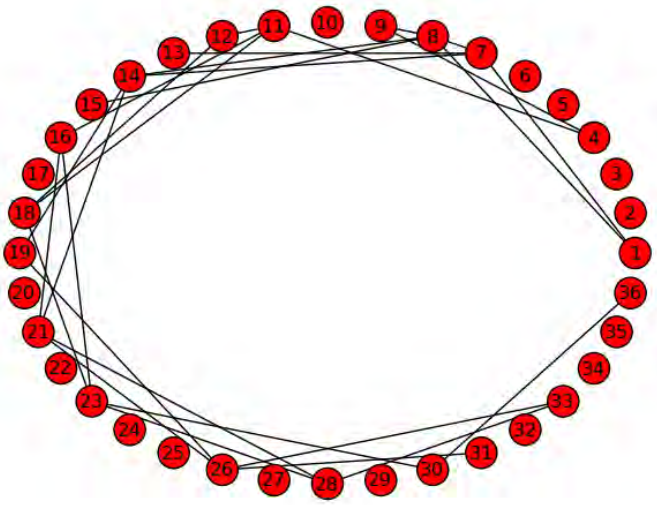

Figura 3.2: Representación del problema clásico como un grafo bidireccional

que establece una ruta y la reserva para mandar las fracciones del paquete, pero no toma en cuenta que la red es cambiante.

3.2. Objetivo general

En el contexto de nuestro problema, cada nodo actúa como router, es decir transmite información propia y retransmite información enviada por otros nodos. Los nodos se comunican directamente con sus vecinos.

El objetivo principal de este trabajo es poder establecer una ruta (no un canal de comunicación) a través de la red basándose en datos locales y que esta ruta sea óptima para las condiciones particulares de la red y de cada nodo en un instante t. Esto es que cada nodo proporcionará la información de los nodos con los que tiene una conexión activa y que cuando suceda algún cambio por desconexión o incorporación de otro nodo a la red, éste se vea reflejado en el estado global sin tener que calcular toda la imagen de la red entera.

Esto se puede lograr utilizando una función de conexión, la cual nos indica para cada nodo k si éste se encuentra disponible en la red en el tiempo t. La función de conexión es local al nodo, el conjunto de los resultados de los nodos ensambla el estado global de la red en un instante t. La función de conexión considera si el nodo, además de cumplir condiciones externas, cumple con tiempo y espacio interno para transmitir. Es decir, aunque las condiciones de la red sean óptimas esto no es suficiente para que el nodo funcione como router, es necesario que el planificador de tareas del nodo asigne tiempo para transmitir.

Este problema es complejo, puesto que es necesario determinar una ruta a través de la red y la red cambia frecuentemente, por ejemplo, si utilizamos el nodo k como nodo intermediario para llegar del nodo a al nodo b y el siguiente salto es al nodo $k+1$, para cuando el salto ocurra puede ser que el nodo k ya no esté en condiciones para transmitir, por lo tanto no tenemos una ruta fija asegurada (como en el caso del algoritmo de wormhole, que se muestra en la Figura 3.3 (subsección 2.2, [18]).

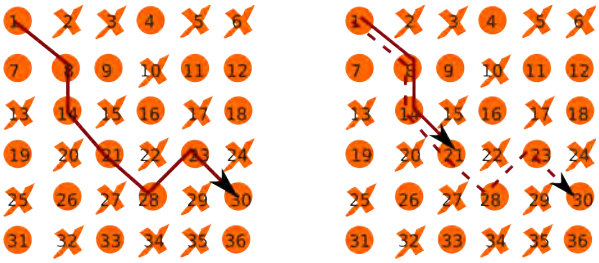

Figura 3.3: Ejemplo de una ruta usando wormhole

En este caso, en la Figura 3.3 en el lado izquierdo el algoritmo de wormhole asegura una ruta, pero en el siguiente tiempo la red cambia y la ruta asegurada ya no es valida. Por ejemplo en la Figura 3.3, en el esquema de la derecha se obtiene la ruta y se trata de asegurar para mandar todos los mensajes, sin embargo si al siguiente tiempo, la red cambia de estado. Es posible que la ruta quede cortada en cualquier punto donde la conexión ya no existe, como en el caso del esquema de la Figura 3.3, donde en el nodo 21 se pierde la conexión y es imposible seguir por esta ruta.

Por ejemplo en la Figura 3.4 se describe la ruta más corta calculada por el algoritmo de Dijsktra, ésta se guarda en la tabla de ruteo y ahora cuando se quiera mandar un mensaje entre el nodo 1 y 36 se usará la ruta $1 \rightarrow 7 \rightarrow 14 \rightarrow 21 \rightarrow 16 \rightarrow 23 \rightarrow 30 \rightarrow 36$. Al instante siguiente la red cambia, por

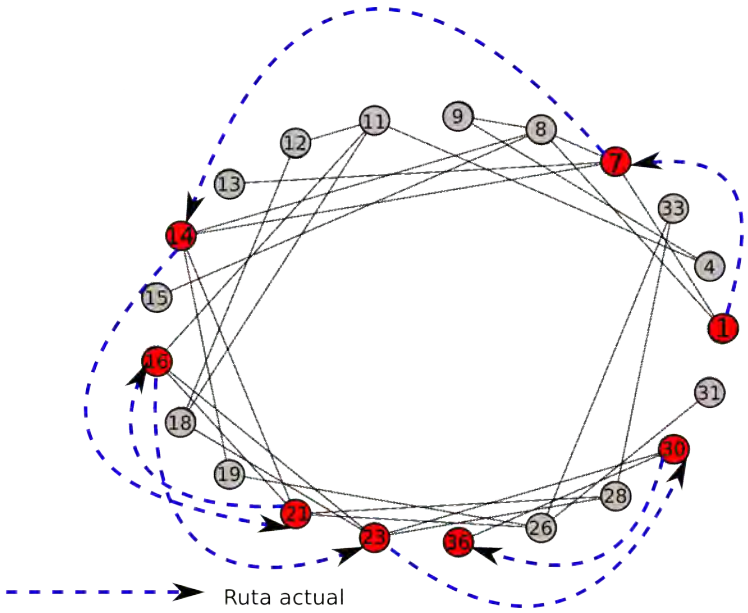

Figura 3.4: Ruta óptima para el tiempo t, con respecto a la Figura 3.3

lo que ahora se tiene un grafo como el mostrado en la Figura 3.5, en el cual cuando se llega al nodo 21,

donde no hay otro camino que tomar y la ruta no llega a su destino, sin embargo, si se actualiza la red instantáneamente, se puede ver que de acuerdo con los cambios instantáneos de la red,se puede notar que ahora es más conveniente usar la ruta $1 \rightarrow 8 \rightarrow 9 \rightarrow 16 \rightarrow 22 \rightarrow 30 \rightarrow 36$, como en la Figura 3.6.

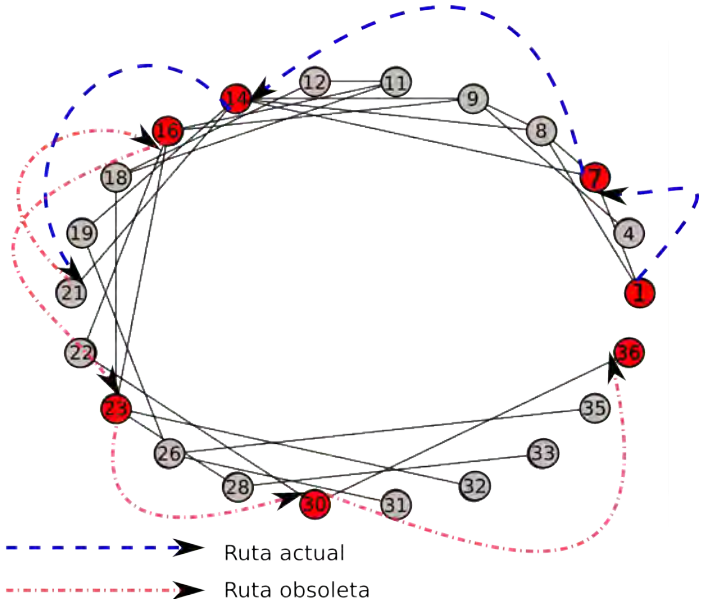

Figura 3.5: Ruta obsoleta para el tiempo $t + 1$

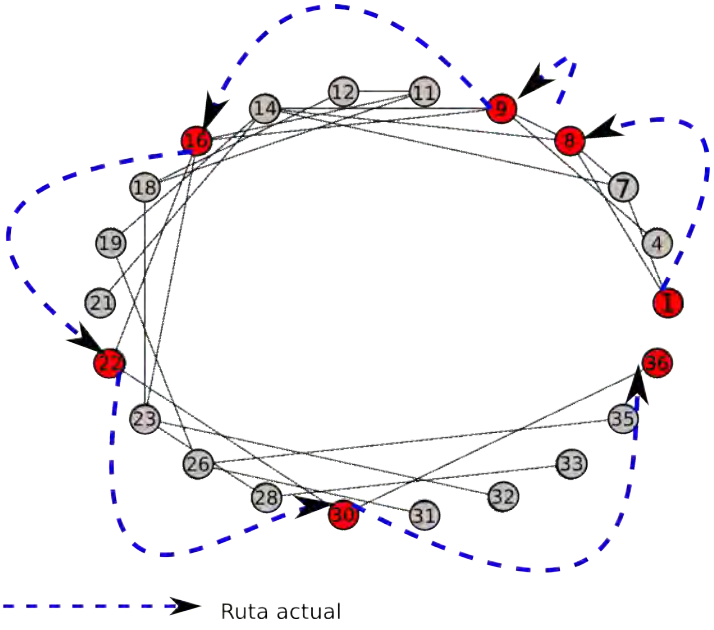

Figura 3.6: Ruta correcta para la ruta $t + 1$

En la Figura 3.7 se muestra la nomenclatura para determinar si un nodo puede o no funcionar como ruteador, si conocemos la información de cuales son los nodos que están disponibles en el instante t entonces es posible construir una ruta del punto de origen al punto destino.

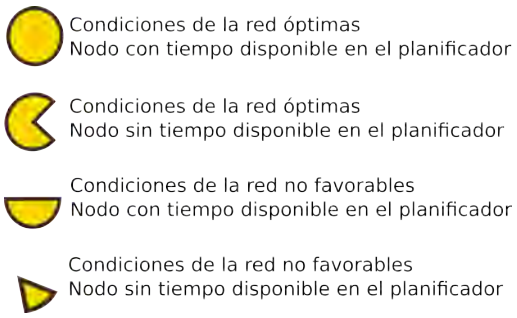

Figura 3.7: Nomenclatura utilizada para determinar si un nodo puede ser utilizado como router.

Por ejemplo, suponiendo que la Figura 3.8 es una red, el estado de los nodos de la red cambia independiente y constantemente. En tiempo t_1 comienza la construcción de la ruta, para el tiempo t_2 se identifica el nodo que se usará como intermediario, pero el estado del nodo anterior de la ruta ya cambio. El estado de la red varía frecuentemente, dependiendo de la función de conexión evaluada para cada nodo en cada instante t.

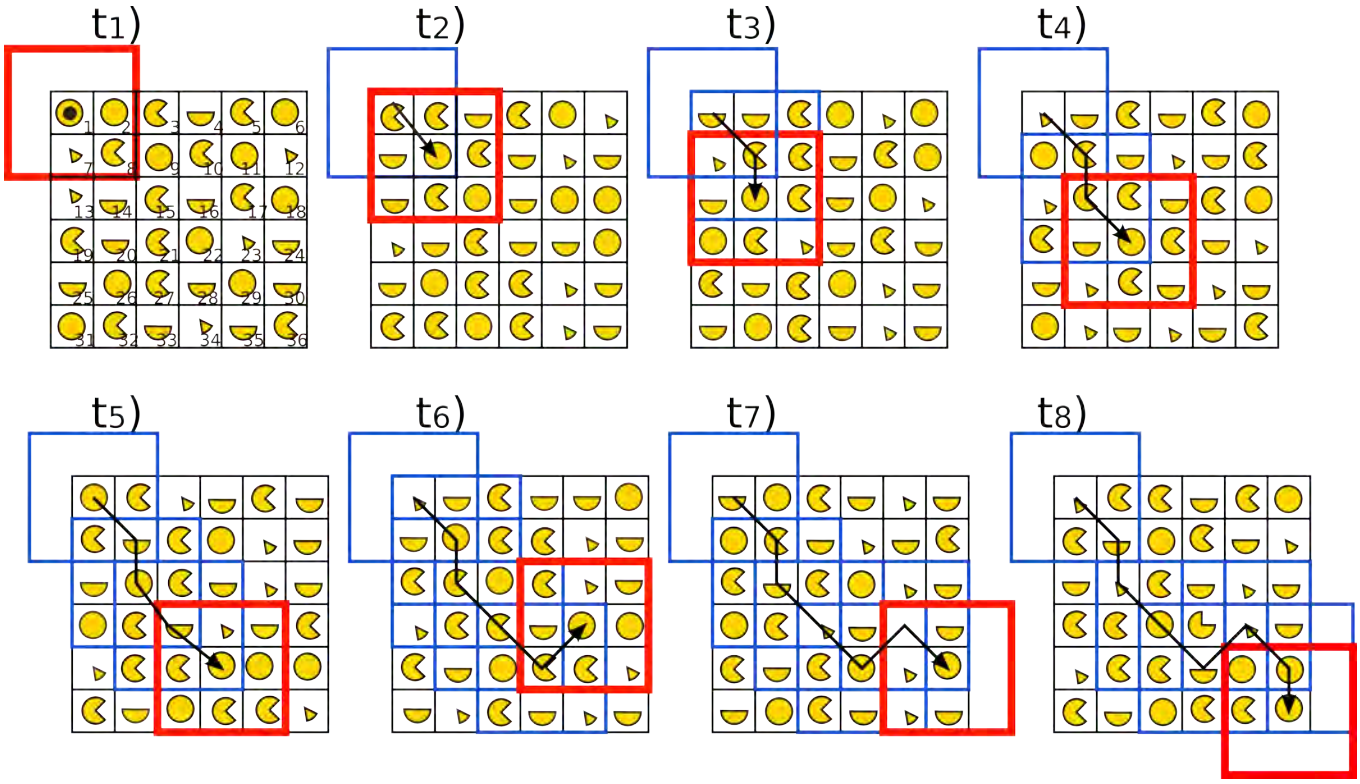

Figura 3.8: Ejemplo de encontrar una ruta a través de la red.

En este trabajo se propone la incorporación de la función de conexión que evalué las condiciones de la red y las del nodo para conocer el estado actual del nodo y por lo tanto, el estado global del sistema.

Es necesario cuantificar el costo de las rutas, ya que puede existir más de una, se propone utilizar lo saltos de nodo a nodo como medida. Esta medida puede ser acotada, es decir, podemos conocer las

rutas que existen el nodo A al nodo B de máximo g saltos, tal que $g = 0, 1, \ldots, n-1$, donde n es el número de nodos.

3.3. Resumen

Es necesario considerar las condiciones de tiempo real de cada nodo, de manera individual, como lo es el espacio disponible en el planificador el nodo. Esta evaluación debe realizarse frecuentemente para evitar tomar decisiones basadas en información obsoleta, por lo tanto valida para el tiempo $t-1$ y falsa para el tiempo t.

Al introducir la función de conexión, podemos evaluar las condiciones del nodo y las de la red en un tiempo t.

La función de conexión evalúa la disponibilidad de espacio en el planificador de cada nodo, junto con las condiciones globales de la red, esto nos permite conocer cuales son los nodos que realmente pueden funcionar como ruteadores.

El nodo destino se mantiene fijo, sin embargo el nodo origen para trazar la ruta al destino cambia a cada paso, conforme se comienza a recorrer la red.

Se busca establecer una ruta entre un nodo A y un nodo B, no abrir un canal de comunicación.

Capítulo 4

CONSTRUCCIÓN DEL ALGORITMO

La ciencia se construye a partir
de aproximaciones que gradualmente
se acercan a la verdad.
Isaac Asimov

El ruteo en una red es en esencia un problema de teoría de grafos, como se muestra en la Figura 4.1 donde se muestra un grafo representando una red. Los nodos en el grafo (etiquetados del 0 al 9), como pueden ser *hosts*, switches, ruteadores o incluso redes. Para este caso asumiremos que los nodos se comportan como ruteadores. Las aristas del grafo corresponden a los enlaces de la red. Cada arista tiene asociado un *costo*, el cual nos da algún indicio de la viabilidad de mandar tráfico por un enlace específico [15].

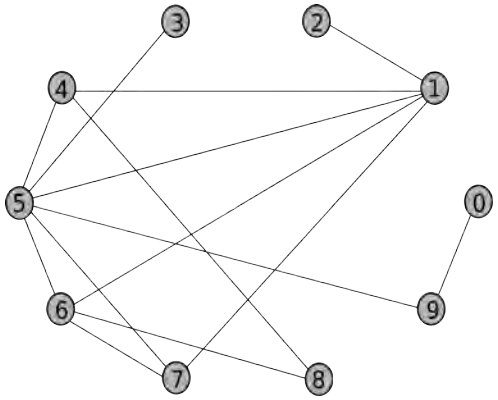

Figura 4.1: Representación de una red

Como se explicó en capítulos previos, una red puede ser representada como un grafo, por lo tanto entre las múltiples representaciones matemáticas existentes, hemos elegido representar una red por medio de una matriz de adyacencia, de esta forma los datos aportados por cada nodo se ven reflejados en 2 posiciones de la matriz y esta matriz representará el estado global del sistema.

A continuación se explica como se obtiene cada dato de la matriz de adyacencia de la función de

conexión.

4.1. Función de conexión

Para determinar si un nodo se encuentra disponible para conexión en la red se define una función que considera características del mensaje enviado y las condiciones instantáneas de la red, como es la calidad de servicio (si es vídeo, voz, ftp, etc), los saltos que lleva acumulados el paquete, la carga instantánea del canal, las cotas fijas de ocupación máxima del canal, la cota de pérdida de datos que soporta cada calidad de servicio y la disponibilidad del nodo para transmitir. Tres de estos parámetros son propios de toda la red, solo la carga del canal y la disponibilidad de transmisión son dependientes del nodo.

En el esquema de red que se presenta, cada nodo funciona como un potencial ruteador, sin embargo este nodo también realiza otras tareas además de transmitir, por lo tanto es necesario determinar si el planificador del nodo tiene espacio disponible para transmitir, esto se simula asignando un número aleatorio entre 40 (que serian los procesos básicos del nodo) y 800 (que sería el nodo sin espacio suficiente para transmitir), este número se genera cada que se llama a la función, ya que es un valor que debe ser instantáneo. Si este número aleatorio que hemos denominado *tareas* es menor a 800, el nodo puede aceptar la petición y evaluar si con las condiciones actuales de la red y del nodo es viable mostrarse como un nodo conectado y representar su valor en la matriz de adyacencia como 1; cuando el nodo está demasiado ocupado para atender la petición, ni siquiera evalúa las condiciones de la red y se muestra como no conectado, se ve representado en la matriz de adyacencia como un cero.

Después de determinar si el nodo puede transmitir, es necesario saber si las condiciones de la red lo permitirán, por ejemplo para cada calidad de servicio existe un porcentaje máximo que se puede tolerar de pérdida de datos, si en el canal de comunicación del nodo se está perdiendo más de esa cota, entonces, no es posible transmitir, también si el paquete que va a recibirse y reenviarse tiene un contador de 16 saltos, no se puede transmitir, ya que excede el tiempo de vida tolerado y el paquete debe ser desechado. También debe conocerse si el canal tiene la capacidad de transmitir la carga que se requiere, entonces se evalúa la capacidad requerida con la capacidad física del canal. Si estas 3 condiciones se cumplen entonces el nodo se mostrará como disponible y conectado en la matriz de adyacencia de la red.

Los datos para la carga máxima soportada, el porcentaje de pérdida de datos y la carga requerida se representan en el código como números aleatorios pequeños entre 0 y 1. El número de saltos se determina generando un número entero aleatorio entre 0 y 16, este parámetro puede ser modificado si se requiere que la vecindad sea de menos saltos.

Entonces definamos la función como sigue:

$$f(x) = e^{\frac{({s_d}^2+\delta {n}^2+{hops}^2+C_{i,j}^2)}{\sigma}} \tag{4.1}$$

Donde:

$$
\begin{aligned}
s &= \text{espacio necesario para transmitir} \\
s_d &= \text{espacio disponible en el nodo} \\
qos &= \text{porcentaje de pérdida de datos soportada por la QoS} \\
\delta n &= \text{pérdida de datos en el canal} \\
hops &= \text{número de nodos por los que ha pasado el paquete} \\
C_{i,j} &= \text{Carga del canal entre el nodo } i \text{ y nodo } j \\
C_T &= \text{Carga máxima soportada por el canal}
\end{aligned}
$$

Los valores de todos estos parámetros oscilan entre 0 y 1, es decir son una probabilidad. El conjunto de estas probabilidades determinan si el nodo puede transmitir o no.

Sea A una matriz cuadrada de $n \times n$ elementos. En este esquema si un nodo i se conecta con el nodo j, entonces las posiciones de la matriz $A_{i,j} = A_{j,i} = 1$, cuando la conexión no es posible entonces $A_{i,j} = A_{j,i} = 0$. Estos valores son independientes para $\frac{(n \times n) - n}{2}$ de los valores de A.

Cada fila y columna de la matriz A representa un procesador con sus conexiones, este procesador también funciona como ruteador. Debe ser evaluado si el procesador tiene tiempo de procesamiento suficiente para funcionar como ruteador, siendo este servicio de baja prioridad para el procesador, por lo tanto, el procesador primero atenderá sus propios servicios y después atenderá las peticiones externas. En la Figura 4.2 se muestra como es la matriz de adyacencia generada por el programa.

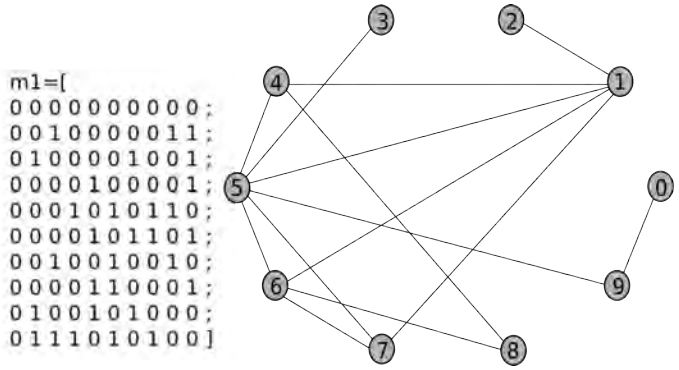

Figura 4.2: Ejemplo de una matriz de adyacencia con la representación gráfica de la red.

El peor escenario que es posible tener es que la topología de la red cambie rápidamente. Esto se ve reflejado en que para un intervalo de tiempo de n unidades de tiempo, tenemos n distintas matrices de adyacencia. Para evitar un error común en el algoritmo de ruteo adaptable RIP, que es caer en un ciclo [1], , el algoritmo evalúa a cada instante las condiciones de la red, por lo que las matrices son independientes y no guardan una relación con el estado anterior de la red.

[1] Cuando la información de las rutas no se encuentra actualizada y se trabaja con información obsoleta

4.2. Matriz de adyacencia

Para poder obtener el estado global de la red por medio de datos locales, se implementa una matriz de adyacencia, de esta manera cada nodo aporta la información sobre los nodos con los que se conecta, estos datos se representan en una fila y una columna.

Sea $M_{i,j}$ el valor de la matriz de adyacencia en la fila i y la columna j, con $i, j = 0, \ldots, n$, sea M simétrica, entonces $M_{i,j} = M_{j,i}$, por lo tanto los valores de de las posiciones $M_{i,j}, M_{j,i}$ se obtienen a través de la evaluación de una función de conexión que toma en cuenta condiciones locales del nodo, estas condiciones son la carga en el canal de comunicación y el espacio disponible en el planificador del nodo, también considera las condiciones instantáneas de la red, como es la cota de perdida de datos que permiten las distintas calidades de servicio, entre otras.

Por ejemplo si tenemos una red como la que se muestra en la figura 4.1, en esta representación de la red como un grafo de 10 nodos, la construcción de su matriz de adyacencia se realiza de la siguiente manera:

1. Para el nodo 0, este nodo solo tiene conexión con el nodo 9, entonces aporta datos a la matriz M en las posiciones $M_{0,9}$ y $M_{9,0}$. Cada valor de la matriz es la evaluación de la función f para cada nodo con el enlace a todos los demás nodos de la red, por lo tanto cada casilla es el valor de f, , como se ilustra en la Tabla 4.1.

×	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	f=0	f=0	f=0	f=0	f=0	f=0	f=0	f=0	f=0	f=1
1	f=0	×	×	×	×	×	×	×	×	×
2	f=0	×	×	×	×	×	×	×	×	×
3	f=0	×	×	×	×	×	×	×	×	×
4	f=0	×	×	×	×	×	×	×	×	×
5	f=0	×	×	×	×	×	×	×	×	×
6	f=0	×	×	×	×	×	×	×	×	×
7	f=0	×	×	×	×	×	×	×	×	×
8	f=0	×	×	×	×	×	×	×	×	×
9	f=1	×	×	×	×	×	×	×	×	×

Tabla 4.1: Nodo 0 conectado con nodo 9

2. El nodo 1 tiene conexión con los nodos 2, 4, 5, 6 y 7, por lo tanto este nodo aportará la información de la fila y columna 1, en las posiciones $M_{1,2}, M_{1,4}, M_{1,5}, M_{1,6}, M_{1,7}$, $M_{2,1}, M_{4,1}, M_{5,1}, M_{6,1}$ y $M_{7,1}$, como se ilustra en la Tabla 4.2.

3. El nodo 2 se conecta con el nodo 1, por lo tanto aporta información a las posiciones $M_{1,2}$ y $M_{2,1}$, como se ilustra en la Tabla 4.3.

4. Así con cada nodo, al final tendremos la matriz de adyacencia de toda la red, como se ve en la Tabla 4.4

×	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	×	f=0	×	×	×	×	×	×	×	×
1	f=0	f=0	f=1	f=0	f=1	f=1	f=1	f=1	f=0	f=0
2	×	f=1	×	×	×	×	×	×	×	×
3	×	f=0	×	×	×	×	×	×	×	×
4	×	f=1	×	×	×	×	×	×	×	×
5	×	f=1	×	×	×	×	×	×	×	×
6	×	f=1	×	×	×	×	×	×	×	×
7	×	f=1	×	×	×	×	×	×	×	×
8	×	f=0	×	×	×	×	×	×	×	×
9	×	f=0	×	×	×	×	×	×	×	×

Tabla 4.2: Nodo 1 conectado con nodos 2, 4, 5, 6 y 7

×	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	×	×	f=0	×	×	×	×	×	×	×
1	×	×	f=1	×	×	×	×	×	×	×
2	f=0	f=1	f=0	f=0	f=0	f=0	f=0	f=0	f=0	f=0
3	×	×	f=0	×	×	×	×	×	×	×
4	×	×	f=0	×	×	×	×	×	×	×
5	×	×	f=0	×	×	×	×	×	×	×
6	×	×	f=0	×	×	×	×	×	×	×
7	×	×	f=0	×	×	×	×	×	×	×
8	×	×	f=0	×	×	×	×	×	×	×
9	×	×	f=0	×	×	×	×	×	×	×

Tabla 4.3: Nodo 2 conectado con nodo 1

×	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	f=0	f=0	f=0	f=0	f=0	f=0	f=0	f=0	f=0	f=1
1	f=0	f=0	f=1	f=0	f=1	f=1	f=1	f=1	f=0	f=0
2	f=0	f=1	f=0	f=0	f=0	f=0	f=0	f=0	f=0	f=0
3	f=0	f=0	f=0	f=0	f=0	f=1	f=0	f=0	f=0	f=0
4	f=0	f=1	f=0	f=0	f=0	f=1	f=0	f=0	f=1	f=0
5	f=0	f=1	f=0	f=1	f=1	f=0	f=1	f=1	f=0	f=1
6	f=0	f=1	f=0	f=0	f=0	f=1	f=0	f=1	f=1	f=0
7	f=0	f=1	f=0	f=0	f=0	f=1	f=1	f=0	f=0	f=0
8	f=0	f=0	f=0	f=0	f=1	f=0	f=1	f=0	f=0	f=0
9	f=1	f=0	f=0	f=0	f=0	f=1	f=0	f=0	f=0	f=0

Tabla 4.4: Matriz de adyacencia de toda la red

En la Tabla 4.4 vemos la matriz de adyacencia de la red en el tiempo t, sin embargo supongamos que ocurren cambios en la red y algunos nodos dejan de tener carga y en la siguiente evaluación de la función de conexión resultan estar disponibles y otros nodos comienzan a tener más trabajo interno y

dejan de estar disponibles, entonces la matriz de adyacencia para el tiempo $t + 1$ seria distinta, como en la Tabla 4.5.

×	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	f=0	f=0	f=0	f=0	f=1	f=0	f=1	f=0	f=1	f=1
1	f=0	f=0	f=1	f=0	f=0	f=1	f=1	f=0	f=0	f=0
2	f=0	f=1	f=0	f=1	f=1	f=1	f=1	f=0	f=0	f=0
3	f=0	f=0	f=1	f=0	f=0	f=1	f=0	f=0	f=0	f=1
4	f=1	f=0	f=1	f=0	f=0	f=0	f=1	f=0	f=1	f=0
5	f=0	f=1	f=1	f=1	f=0	f=0	f=1	f=1	f=0	f=1
6	f=1	f=1	f=1	f=0	f=1	f=1	f=0	f=1	f=0	f=1
7	f=0	f=0	f=0	f=0	f=0	f=1	f=1	f=0	f=1	f=0
8	f=1	f=0	f=0	f=0	f=1	f=0	f=0	f=1	f=0	f=1
9	f=1	f=0	f=0	f=1	f=0	f=1	f=0	f=0	f=1	f=0

Tabla 4.5: Otra matriz de adyacencia de toda la red

4.2.1. Análisis de la matriz de adyacencia

Se han analizado las matrices generadas por la función antes descrita. Las matrices resultantes presentan las siguientes características:

- Son matrices cuadradas y simétricas.
- Presentan la diagonal en ceros (porque no hay ciclos)
- Son matrices triangulares superiores por encima de la diagonal en cero.
- Son matrices triangulares inferiores por debajo de la diagonal en cero.
- Dependiendo del número de nodos conectados los eigenvectores positivos disminuyen.

Dado que la matriz de adyacencia es una representación matemática de la red, es posible calcular el costo de las rutas entre todos los nodos, agregando una restricción que es el número de saltos. Es decir calcular el costo de las rutas desde cada nodo a los otros $n - 1$ nodos, utilizando el algoritmo de Floyd-Warshall.

4.3. Manejo de autovalores

Para cada matriz de adyacencia generada en el tiempo se calculan sus autovalores, éstos sólo son indicadores de que tan conectada está la red, puesto que cuando aumentan las conexiones los autovalores positivos disminuyen, cuando tenemos una matriz dispersa el número de autovalores positivos y negativos se encuentra balanceado.

4.4. Construcción de la ruta

La ruta se construye pasó a paso, el algoritmo de Floyd-Warshall nos devuelve un archivo de texto en el formato mostrado en la Tabla 4.6.

En el primer bloque de datos, se indica cuantos saltos son necesarios para llegar de un punto a un destino. Por ejemplo la fila

... 0: {0: 0, 1: 1, 2: 1, 3: 2, 4: 2, 5: 2, 6: 1, **7: 2**, 8: 1, 9: 1}

muestra los saltos necesarios desde el nodo 0 a todos los demás. Supongamos que queremos ir del nodo 0 al nodo 7, la fila nos indica que para llegar de 0 a 7 son necesarios 2 saltos. Después de saber a que distancia se encuentran los puntos, es necesario consultar el otro bloque de datos y consultar cuales son los nodos intermedios a visitar antes de llegar al destino.

Con este mismo ejemplo, vayamos al bloque de datos de la ruta y revisemos el bloque de datos correspondiente al nodo 0.

...0: {0: None, 1: 0, 2: 0, 3: 6, 4: 2, 5: 1, 6: 0, **7: 8**, **8: 0**, 9: 0}

El cuál nos indica que para llegar del nodo 0 al 7 debemos pasar por el nodo 8, consultamos como llegar del nodo 0 al nodo 8 y obtenemos la ruta, que es $0 \rightarrow 8 \rightarrow 7$.

En la Tabla 4.6 se muestran resaltados los datos útiles para determinar el número de saltos y la ruta a seguir para el tiempo t. En cada paso hacia adelante se visita un nuevo nodo, puede ser que el nodo visitado con anterioridad ya no sea alcanzable. También puede suceder que la ruta se haga más corta al llegar al siguiente nodo o se prolongue significativamente.

4.5. Vista general del algoritmo

En resumen el algoritmo consiste de 2 pasos:

1. La construcción de las matrices de adyacencia utilizando la función de conexión.
2. La construcción de la rutas en el tiempo, hasta alcanzar el nodo destino.

```
0: {0: 0, 1: 1, 2: 1, 3: 2, 4: 2, 5: 2, 6: 1, 7: 2, 8: 1, 9: 1},
1: {0: 1, 1: 0, 2: 2, 3: 2, 4: 2, 5: 1, 6: 1, 7: 2, 8: 1, 9: 1},
2: {0: 1, 1: 2, 2: 0, 3: 2, 4: 1, 5: 2, 6: 2, 7: 2, 8: 1, 9: 1},
3: {0: 2, 1: 2, 2: 2, 3: 0, 4: 1, 5: 2, 6: 1, 7: 1, 8: 2, 9: 1},
4: {0: 2, 1: 2, 2: 1, 3: 1, 4: 0, 5: 1, 6: 2, 7: 1, 8: 2, 9: 2},
5: {0: 2, 1: 1, 2: 2, 3: 2, 4: 1, 5: 0, 6: 1, 7: 1, 8: 1, 9: 2},
6: {0: 1, 1: 1, 2: 2, 3: 1, 4: 2, 5: 1, 6: 0, 7: 2, 8: 2, 9: 2},
7: {0: 2, 1: 2, 2: 2, 3: 1, 4: 1, 5: 1, 6: 2, 7: 0, 8: 1, 9: 2},
8: {0: 1, 1: 1, 2: 1, 3: 2, 4: 2, 5: 1, 6: 2, 7: 1, 8: 0, 9: 2},
9: {0: 1, 1: 1, 2: 1, 3: 1, 4: 2, 5: 2, 6: 2, 7: 2, 8: 2, 9: 0}},

{0:
{
 0: None, 1: 0, 2: 0, 3: 6, 4: 2, 5: 1, 6: 0, 7: 8, 8: 0, 9: 0},
 1: {0: 1, 1: None, 2: 0, 3: 6, 4: 5, 5: 1, 6: 1, 7: 5, 8: 1, 9: 1},
 2: {0: 2, 1: 0, 2: None, 3: 4, 4: 2, 5: 4, 6: 0, 7: 4, 8: 2, 9: 2},
 3: {0: 6, 1: 6, 2: 4, 3: None, 4: 3, 5: 4, 6: 3, 7: 3, 8: 7, 9: 3},
 4: {0: 2, 1: 5, 2: 4, 3: 4, 4: None, 5: 4, 6: 3, 7: 4, 8: 2, 9: 2},
 5: {0: 1, 1: 5, 2: 4, 3: 4, 4: 5, 5: None, 6: 5, 7: 5, 8: 5, 9: 1},
 6: {0: 6, 1: 6, 2: 0, 3: 6, 4: 3, 5: 6, 6: None, 7: 3, 8: 0, 9: 0},
 7: {0: 8, 1: 5, 2: 4, 3: 7, 4: 7, 5: 7, 6: 3, 7: None, 8: 7, 9: 3},
 8: {0: 8, 1: 8, 2: 8, 3: 7, 4: 2, 5: 8, 6: 0, 7: 8, 8: None, 9: 0},
 9: {0: 9, 1: 9, 2: 9, 3: 9, 4: 2, 5: 1, 6: 0, 7: 3, 8: 0, 9: None}})
```

Tabla 4.6: Un ejemplo de archivo de rutas

4.6. Resumen

El algoritmo propuesto incorpora algoritmos conocidos y probados como el de Floyd-Warshall, para el cálculo de rutas, por la forma en que se calculan las rutas y lo variable que es la red, el algoritmo puede tardar en converger al destino, a veces puede presentarse el caso que cuando cambie la red la ruta converja más rápido que lo calculado al inicio, pero también puede suceder que visitemos varios nodos varias veces (sin necesariamente estar en un ciclo) antes de llegar al destino, sin embargo éstas son cuestiones que aun deben ser trabajadas en el futuro.

La función de conexión es el punto clave de este trabajo, puesto que no solo toma en cuenta el estado de la red, esta función incorpora restricciones de tiempo real, como los son las condiciones globales de la red (pérdida de datos, número de saltos permitidos para el paquete, etc) y las condiciones locales del nodo, como son la carga en el canal y el espacio disponible en el planificador del nodo. Las condiciones del nodo son las que más influyen en el estado general del sistema.

Capítulo 5

RESULTADOS

Son vanas y están plagadas de errores
las ciencias que no han nacido del
experimento, madre de toda certidumbre.
Leonardo Da Vinci

5.1. Tiempo real

Cabe señalar que en este algoritmo hace uso del tiempo real porque al ser un sistema distribuido móvil, los componentes varían en el tiempo. Cada vez que se conecta o se desconecta un dispositivo de la red, la red cambia y también es posible que varios dispositivos cambien su estado en un mismo tiempo, esto afecta el estado global de la red, cada matriz de adyacencia representa el estado de la red en un tiempo t_i, $t_i \in T$, con $i = 1, \ldots, k$, $K \in \aleph$. En la figura 5.1 se muestra como varía la red en distintos t_i.

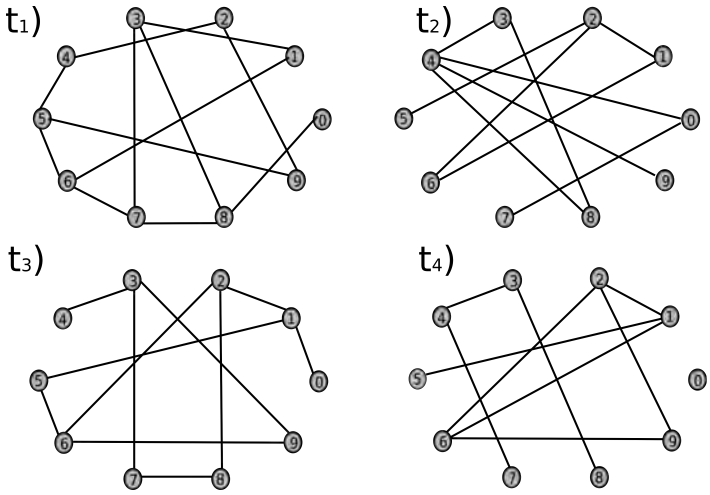

Figura 5.1: Ejemplo de el cambio del estado de la red para 4 instantes de tiempo distintos

Otro elemento de tiempo real es el planificador del nodo. Éste se encarga de asignar tiempo de procesador a las tareas del nodo, si este planificador no tiene espacio suficiente para transmitir el mensaje a través de la red, entonces el nodo debe mostrarse como no disponible.

5.2. El tiempo que se tarda en construir una ruta

El algoritmo antes de calcular la ruta utilizando el algoritmo de Floyd-Warshall, lleva a cabo la construcción de la matriz de adyacencia, evaluando las condiciones de cada nodo.

Se realizaron varias corridas para calcular el tiempo mínimo, máximo y promedio de las ejecuciones, dando como resultado los datos mostrados en la Tabla 5.1. La gráfica de los tiempos de todas la corridas de muestra en la Figura 5.2.

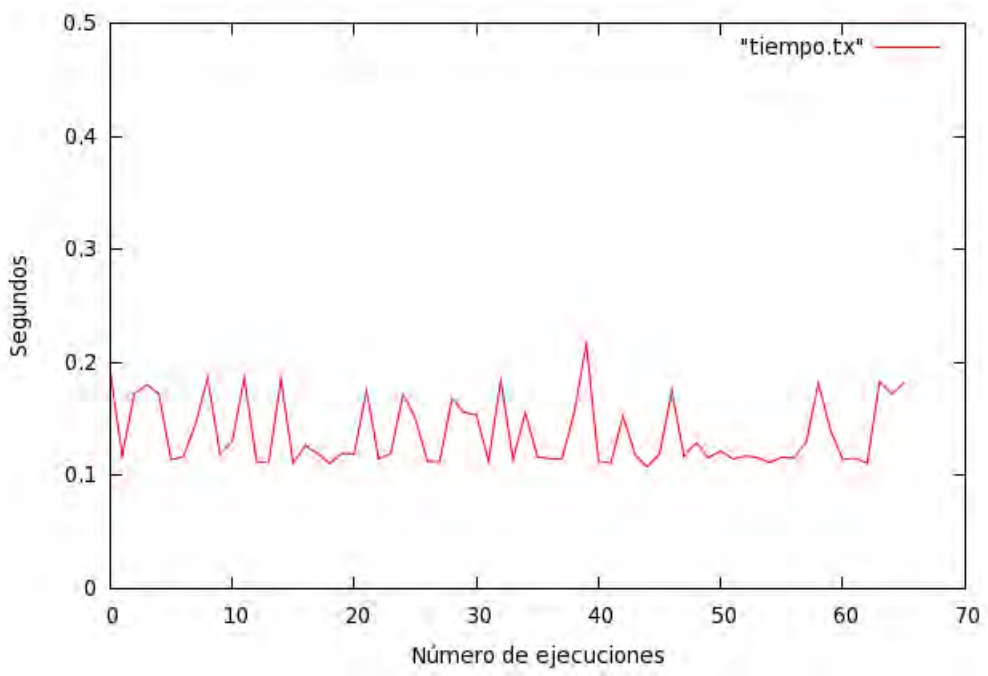

Figura 5.2: Gráfica del tiempo de ejecución de 65 corridas del algoritmo

Tiempo mínimo	0.1076991558
Tiempo promedio	0.1366388798
Tiempo máximo	0.2163119316
σ	0.561268

Tabla 5.1: Tiempos de ejecución del algoritmo

El tiempo no presenta un patrón, el tiempo oscila entre 0.1076991558 ms y 0.2163119316 ms.

5.3. Vigencia de la ruta

Durante la construcción de la ruta pueden darse 4 casos distintos:

1. Que la ruta se conserve igual a la calculada por el algoritmo Floyd-Warshall, esto significa que a pesar de los cambios en la red la ruta calculada en el tiempo t_1 sea la misma para t_n, es decir, que después de n cambios la ruta es la misma que se había calculado en un principio. Se muestra un ejemplo de este caso en A.1.
2. Que la ruta en el tiempo t_n sea más larga que la calculada en el tiempo t_1, es decir, que los constantes cambios de topología ocasionaron que la ruta tuviera más saltos. Se presenta un ejemplo de este caso en A.2.
3. Que la ruta en el tiempo t_n sea más corta que la que se calculó en el tiempo t_1. Se muestra un ejemplo de este caso en A.3.
4. Que la ruta sea infinita o dicho de otra manera el destino es inalcanzable, esto es decir que el nodo destino esta aislado o que el nodo al que se movió en el paso anterior esta aislado, en este caso se espera un tiempo a que el nodo se reconecte o se desecha el paquete. Se presenta un ejemplo de este caso en A.4.

Por ejemplo, en la Figura 5.3 deseamos ir del nodo 3 al 8. En este punto consultamos cuantos saltos son necesarios para llegar de $1 \rightarrow 8$ y son necesarios 3 saltos y la ruta sería $3 \rightarrow 1 \rightarrow 4 \rightarrow 8$, para el siguiente tiempo no sabemos si la ruta seguirá siendo la misma, nos movemos al siguiente nodo en la ruta, es decir, al nodo 1.

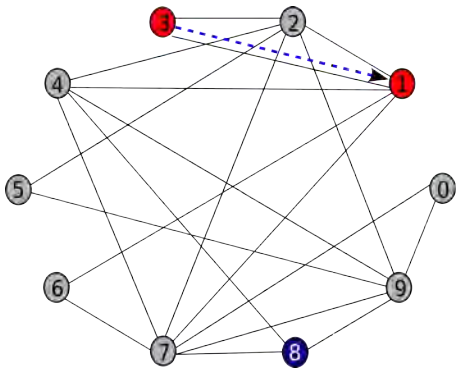

Figura 5.3: Caso 1: Iteración 1

El siguiente estado de la red (que se muestra en la Figura 5.4) volvemos a calcular la ruta ahora desde el nodo 1 al nodo 8, para esta configuración existe un enlace directo de 1 a 8, por lo tanto la ruta final fue más corta que la calculada con la configuración inicial, la ruta queda $3 \rightarrow 1 \rightarrow 8$.

Otro caso posible es que la ruta no se encuentre de manera tan rápida y conveniente. Es posible que la ruta sea más larga y costosa. Por ejemplo, si suponemos que deseamos comunicar el nodo 4 con el nodo 8 en el instante t_1 tenemos la configuración que se muestra en la Figura 5.5 En este paso si consultamos el diccionario de datos correspondiente a este estado de la red y obtendremos que son

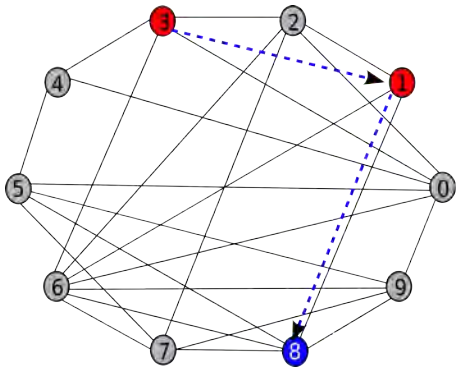

Figura 5.4: Caso 1: Iteración 2

necesarios 3 saltos y la ruta sería $4 \rightarrow 0 \rightarrow 5 \rightarrow 8$. Como no conocemos el siguiente estado de la red solo nos movemos al nodo 0.

Situándonos en el nodo 0, ahora se tiene la configuración mostrada en la Figura 5.5 para el tiempo t_2, notemos que el enlace para llegar de $4 \rightarrow 0$ ya no existe en esta configuración. Estando en el nodo 0 obtenemos la ruta nuevamente pero ahora desde $0 \rightarrow 8$, en este caso son necesarios 2 saltos y la ruta es $0 \rightarrow 3 \rightarrow 8$. Nos movemos al nodo 3 para el siguiente paso.

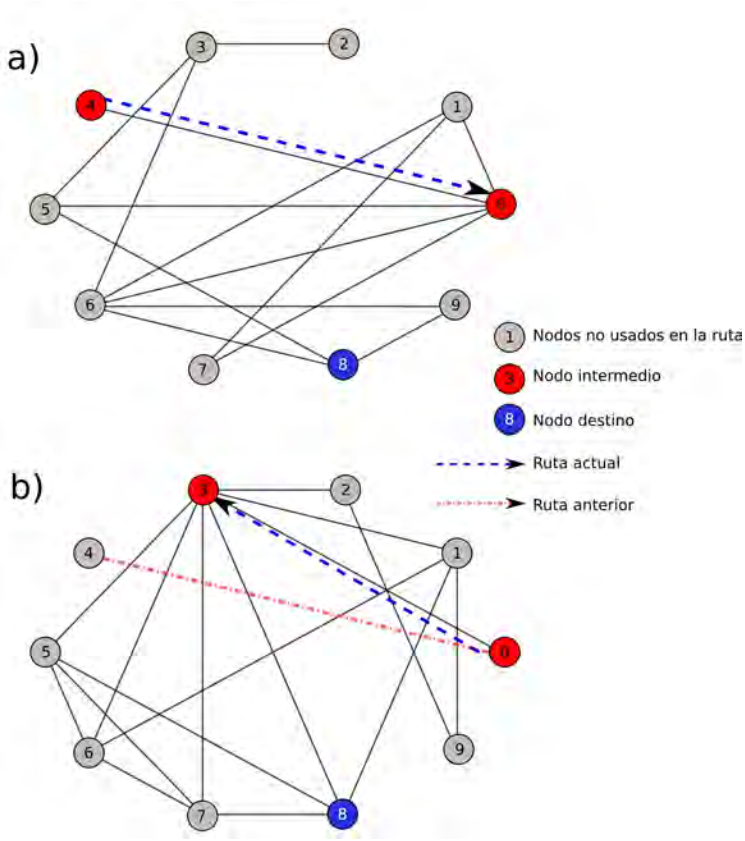

Figura 5.5: a) Caso 2: Iteración 1, tiempo t_1. b) Caso 2: Iteración 2, tiempo t_2

En la configuración mostrada en la Figura 5.6 en el tiempo t_3 notamos que la conexión $3 \rightarrow 8$ ya no existe, por lo tanto es necesario consultar la ruta más corta ahora de $3 \rightarrow 8$. En este instante t_3 son necesarios 2 saltos y la ruta es $3 \rightarrow 9 \rightarrow 8$. Nos movemos al nodo 9.

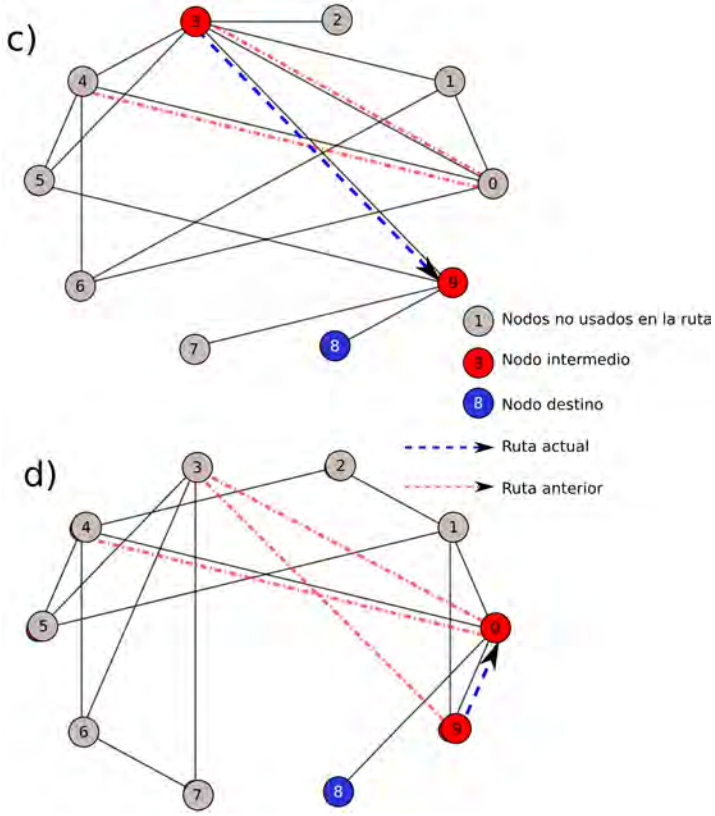

Figura 5.6: c) Caso 2: Iteración 3, tiempo t_3. d) Caso 2: Iteración 4, tiempo t_4

En el tiempo t_4 , como se muestra en la figura 5.6 nos encontramos de nuevo con el caso en que el enlace de $9 \rightarrow 8$ ya no existe. Por lo que consultamos de nuevo la ruta, serán necesarios 2 saltos y la ruta será $9 \rightarrow 0 \rightarrow 8$. Nos movemos al nodo 0, como ya hemos pasado por el nodo 0 en el tiempo t_2, sin embargo, como son configuraciones independientes y en esta el enlace de $0 \rightarrow 3$ ya no existe, no estamos ante un caso de ciclo.

En el tiempo t_5 tenemos la configuración mostrada en la Figura 5.7, estando situados en el nodo 0 es necesario consultar la ruta de nuevo de $0 \rightarrow 8$, son necesarios 2 saltos y la ruta es $0 \rightarrow 2 \rightarrow 8$, nos movemos al nodo 2.

En el tiempo t_6 con la configuración mostrada en la Figura 5.7 situados en el nodo 2, consultamos al ruta y tenemos una conexión directa entre el nodo 2 y el nodo 8, por lo tanto, ya hemos llegado al destino.

En este caso es muy tardado llegar del nodo origen al destino, porque en el camino van desapareciendo conexiones y de tener al principio un ruta de 3 saltos, al final tenemos una ruta con 6 saltos.

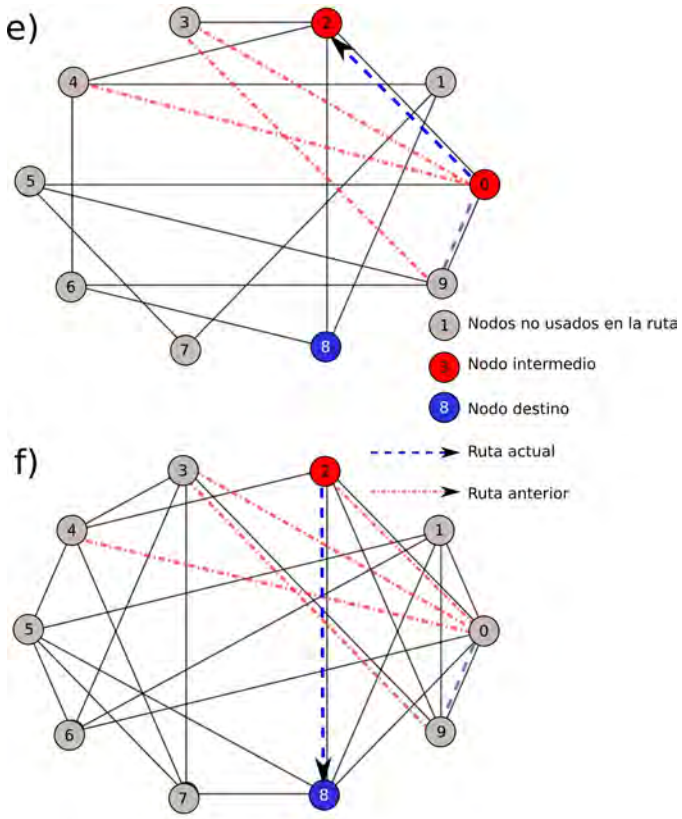

Figura 5.7: e) Caso 2: Iteración 5, tiempo t_5. f) Caso 2: Iteración 6, tiempo t_6

5.4. Variaciones

La ruta varía en cada cambio de la matriz de adyacencia de la función de conexión, además en cada paso cambia el origen, pero el destino se mantiene fijo.

Como se aprecia en las Figuras 5.5, 5.6 y 5.7, el destino, que en este caso es el nodo 8, no cambia, lo que cambia es el origen de la nueva ruta, el nodo anterior no influye en la construcción de la ruta nueva, ya que las matrices son independientes.

Capítulo 6

CONCLUSIONES

Sólo podemos ver poco del futuro,
pero lo suficiente para darnos cuenta
de que hay mucho que hacer.

Alan Mathison Turing

Durante la realización de este trabajo se llegó a las siguientes conclusiones.

La ruta construida por el algoritmo es óptima para cada estado transitorio de la red. Es probable quedar en un estado en el que sea imposible calcular una nueva ruta al destino, esto puede ser a causa de que el planificador del nodo se encuentre muy ocupado para transmitir, en este caso es necesario agregar al algoritmo una restricción para que a la llegada del paquete siempre se le asigne espacio en el planificador para transmitir, aunque ya no para recibir paquetes nuevos, esta restricción no ha sido agregada al código, se deja para trabajo futuro.

La rutas pueden visitar el mismo nodo más de una vez, sin que esto signifique que se esta en presencia de un ciclo. Además, las rutas que se usan para llegar del nodo A al nodo B no son necesariamente reversibles para conectar B con A, porque al instante siguiente de que el paquete llega a su destino, es posible que la ruta por la que se llego ya no exista, sin embargo se garantiza la llegada del mensaje al construir una nueva ruta.

El análisis de los eigenvalores de la matriz de adyacencia de la función de conexión no aportó mucha información sobre la conectividad de la red, sin embargo para trabajo futuro se ha encontrado en [14] que el segundo eigenvalor más pequeño de la matriz Laplaciana del grafo es determinante para el rango de convergencia de algoritmos de consenso distribuido, la investigación sobre el impacto de este valor en nuestro problema de estudio se deja como trabajo futuro.

En los resultados de tiempo de ejecución se observó que el tiempo es oscilante y no converge a ningún valor en particular. Como reto futuro se tiene el optimizar el tiempo de ejecución de manera que el tiempo converja.

Bibliografía

[1] C. E. Agnew. On quadratic adaptive routing algorithms. *Commun. ACM*, 19(1):18–22, 1976.

[2] S. G. A. Arezou Mohammadi. Scheduling algorithms for real-time systems. Technical Report 2005-499, School of Computing,Queens University, Kingston, Ontario,Canada K7L 3N6, July 2005.

[3] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.

[4] J. Duato, S. Yalamanchili, and L. Ni. *Interconnection Networks: An Engineering Approach*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1997.

[5] A. Girard. *Routing and Dimensioning in Circuit-Switched Networks*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[6] C. J. Glass, C. J. Glass, L. M. Ni, and L. M. Ni. The turn model for adaptive routing. In *In Proceedings of the International Symposium on Computer Architecture*, pages 278–287, 1992.

[7] J. Hu and R. Marculescu. Dyad - smart routing for networks-on-chip. In *In ACM/IEEE Design Automation Conference*, pages 260–263, 2004.

[8] P. Jacquet, P. Mühlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot. Optimized link state routing protocol for ad hoc networks. pages 62–68, 2001.

[9] Z. Liu, J. Kim, B. Lee, and C. Kim. A routing protocol based on adjacency matrix in ad hoc mobile networks. *Advanced Language Processing and Web Information Technology, International Conference on*, 0:430–436, 2008.

[10] D. Medhi and K. Ramasamy. *Network Routing: Algorithms, Protocols, and Architectures*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[11] P. Mohapatra. Wormhole routing techniques for directly connected multicomputer systems. *ACM Comput. Surv.*, 30(3):374–410, 1998.

[12] V. D. Park and M. S. Corson. A highly adaptive distributed routing algorithm for mobile wireless networks, 1997.

[13] A. Patooghy and H. Sarbazi-Azad. Performance comparison of partially adaptive routing algorithms. *Advanced Information Networking and Applications, International Conference on*, 2:763–767, 2006.

[14] B. . E. A. A. . Patterson, S. ; Bamieh. Convergence rates of distributed average consensus with stochastic link failures. 2010.

[15] L. L. Peterson and B. S. Davie. *Computer Networks: A Systems Approach, Fourth Edition (The Morgan Kaufmann Series in Networking)*. Morgan Kaufmann, 4 edition, March 2007.

[16] K. H. Rosen. *Handbook of discrete and combinatorial mathematics*. AT&T Laboratories, 2000.

[17] C. L. Seitz. The cosmic cube. *Commun. ACM*, 28(1):22–33, 1985.

[18] A. Y. Seydim. Wormhole routing in parallel computers, 1998.

[19] G. P. G. G. Siberschartz Abraham. *Sistemas Operativos*. Limusa Wiley, 2002.

[20] C. P. Sun. Ad-hoc on-demand distance vector routing. In *In Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, 1997.

[21] X. Yuan, W. Nienaber, Z. Duan, and R. Melhem. Oblivious routing for fat-tree based system area networks with uncertain traffic demands. In *SIGMETRICS '07: Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 337–348, New York, NY, USA, 2007. ACM.

APÉNDICES

Apéndice A

PRUEBAS DE VARIACIONES DE LA RUTA

Como se explicó la sección 5.3, pueden presentarse 4 casos en la construcción de la red, a continuación se ejemplifica cada una de ellas.

A.1. La ruta se conserva igual respecto al algoritmo Floyd-Warshall

En este caso (Figura A.1), suponemos que nuestro nodo origen es el nodo 5 y el nodo destino es el nodo 1. En el tiempo t_1 el algoritmo Floyd-Warshall calcula que son necesarios 2 saltos y la ruta más corta a seguir es $5 \rightarrow 4 \rightarrow 1$ (Figura A.2). En este paso nos movemos al nodo 4, que será nuestro nodo origen para la siguiente iteración.

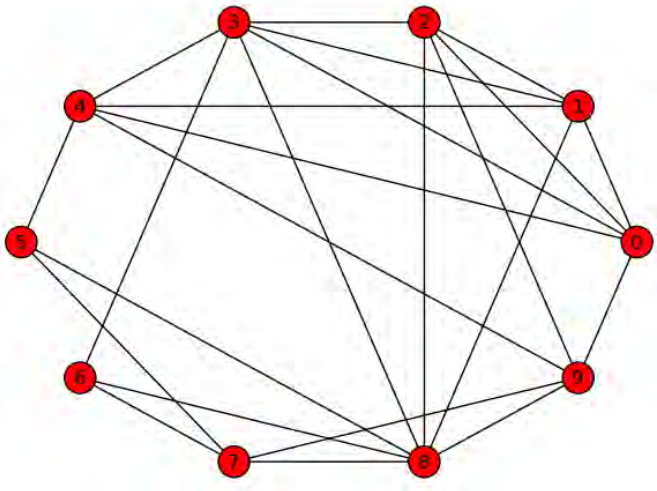

Figura A.1: Caso 1: Iteración 1, tiempo t_1

({
0: {0: 0, 1: 1, 2: 1, 3: 1, 4: 1, 5: 2, 6: 2, 7: 2, 8: 2, 9: 1},
1: {0: 1, 1: 0, 2: 1, 3: 1, 4: 1, 5: 2, 6: 2, 7: 2, 8: 1, 9: 2},
2: {0: 1, 1: 1, 2: 0, 3: 1, 4: 2, 5: 2, 6: 2, 7: 2, 8: 1, 9: 1},
3: {0: 1, 1: 1, 2: 1, 3: 0, 4: 1, 5: 2, 6: 1, 7: 2, 8: 1, 9: 2},
4: {0: 1, 1: 1, 2: 2, 3: 1, 4: 0, 5: 1, 6: 2, 7: 2, 8: 2, 9: 1},
5: {0: 2, 1: 2, 2: 2, 3: 2, 4: 1, 5: 0, 6: 2, 7: 1, 8: 1, 9: 2},
6: {0: 2, 1: 2, 2: 2, 3: 1, 4: 2, 5: 2, 6: 0, 7: 1, 8: 1, 9: 2},
7: {0: 2, 1: 2, 2: 2, 3: 2, 4: 2, 5: 1, 6: 1, 7: 0, 8: 1, 9: 1},
8: {0: 2, 1: 1, 2: 1, 3: 1, 4: 2, 5: 1, 6: 1, 7: 1, 8: 0, 9: 1},
9: {0: 1, 1: 2, 2: 1, 3: 2, 4: 1, 5: 2, 6: 2, 7: 1, 8: 1, 9: 0}},

{0:
{
0: None, 1: 0, 2: 0, 3: 0, 4: 0, 5: 4, 6: 3, 7: 9, 8: 1, 9: 0},
1: {0: 1, 1: None, 2: 1, 3: 1, 4: 1, 5: 4, 6: 3, 7: 8, 8: 1, 9: 0},
2: {0: 2, 1: 2, 2: None, 3: 2, 4: 0, 5: 8, 6: 3, 7: 8, 8: 2, 9: 2},
3: {0: 3, 1: 3, 2: 3, 3: None, 4: 3, 5: 4, 6: 3, 7: 6, 8: 3, 9: 0},
4: {0: 4, 1: 4, 2: 0, 3: 4, 4: None, 5: 4, 6: 3, 7: 5, 8: 1, 9: 4},
5: {0: 4, 1: 4, 2: 8, 3: 4, 4: 5, 5: None, 6: 7, 7: 5, 8: 5, 9: 4},
6: {0: 3, 1: 3, 2: 3, 3: 6, 4: 3, 5: 7, 6: None, 7: 6, 8: 6, 9: 7}, 7:
{0: 9, 1: 8, 2: 8, 3: 6, 4: 5, 5: 7, 6: 7, 7: None, 8: 7, 9: 7}, 8:
{0: 1, 1: 8, 2: 8, 3: 8, 4: 1, 5: 8, 6: 8, 7: 8, 8: None, 9: 8}, 9:
{0: 9, 1: 0, 2: 9, 3: 0, 4: 9, 5: 4, 6: 7, 7: 9, 8: 9, 9: None}})

Figura A.2: Archivo de la ruta de A.1

En la siguiente iteración (Figura A.3), tomando como origen el nodo 4, se calcula la ruta más corta ahora entre el nodo 4 y el nodo 1, el archivo de ruta A.4 indica que existe un camino directo de $4 \rightarrow 1$ en el tiempo t_2. Esto significa que la ruta calculada en el paso anterior, $5 \rightarrow 4 \rightarrow 1$ sigue estando vigente.

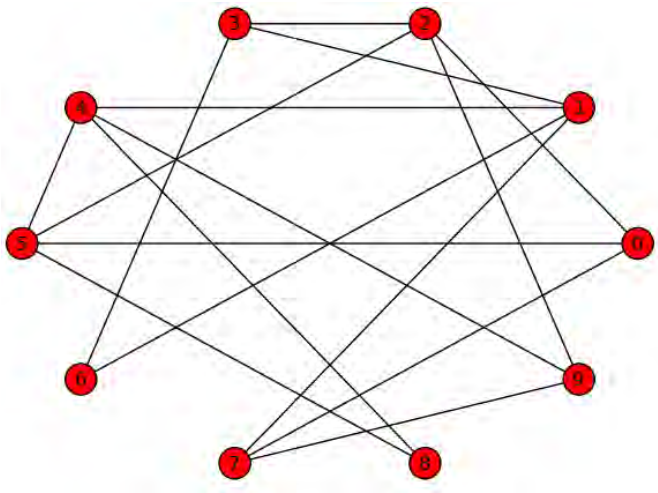

Figura A.3: Caso 1: Iteración 2, tiempo t_2

A.2. La ruta es más larga

En este caso (Figura A.5), suponemos que nuestro nodo origen es el nodo 3 y el nodo destino es el nodo 2. En el tiempo t_1 el algoritmo Floyd-Warshall calcula que son necesarios 2 saltos y la ruta más

({
0: {0: 0, 1: 2, 2: 1, 3: 2, 4: 2, 5: 1, 6: 3, 7: 1, 8: 2, 9: 2},
1: {0: 2, 1: 0, 2: 2, 3: 1, 4: 1, 5: 2, 6: 1, 7: 1, 8: 2, 9: 2},
2: {0: 1, 1: 2, 2: 0, 3: 1, 4: 2, 5: 1, 6: 2, 7: 2, 8: 2, 9: 1},
3: {0: 2, 1: 1, 2: 1, 3: 0, 4: 2, 5: 2, 6: 1, 7: 2, 8: 3, 9: 2},
4: {0: 2, 1: 1, 2: 2, 3: 2, 4: 0, 5: 1, 6: 2, 7: 2, 8: 1, 9: 1},
5: {0: 1, 1: 2, 2: 1, 3: 2, 4: 1, 5: 0, 6: 3, 7: 2, 8: 1, 9: 2},
6: {0: 3, 1: 1, 2: 2, 3: 1, 4: 2, 5: 3, 6: 0, 7: 2, 8: 3, 9: 3},
7: {0: 1, 1: 1, 2: 2, 3: 2, 4: 2, 5: 2, 6: 2, 7: 0, 8: 3, 9: 1},
8: {0: 2, 1: 2, 2: 2, 3: 3, 4: 1, 5: 1, 6: 3, 7: 3, 8: 0, 9: 2},
9: {0: 2, 1: 2, 2: 1, 3: 2, 4: 1, 5: 2, 6: 3, 7: 1, 8: 2, 9: 0}},

{0:
{
0: None, 1: 7, 2: 0, 3: 2, 4: 5, 5: 0, 6: 3, 7: 0, 8: 5, 9: 2},
1: {0: 7, 1: None, 2: 3, 3: 1, 4: 1, 5: 4, 6: 1, 7: 1, 8: 4, 9: 4},
2: {0: 2, 1: 3, 2: None, 3: 2, 4: 5, 5: 2, 6: 3, 7: 0, 8: 5, 9: 2},
3: {0: 2, 1: 3, 2: 3, 3: None, 4: 1, 5: 2, 6: 3, 7: 1, 8: 4, 9: 2},
4: {0: 5, 1: 4, 2: 5, 3: 1, 4: None, 5: 4, 6: 1, 7: 1, 8: 4, 9: 4},
5: {0: 5, 1: 4, 2: 5, 3: 2, 4: 5, 5: None, 6: 3, 7: 0, 8: 5, 9: 2},
6: {0: 2, 1: 6, 2: 3, 3: 6, 4: 1, 5: 2, 6: None, 7: 1, 8: 4, 9: 2},
7: {0: 7, 1: 7, 2: 0, 3: 1, 4: 1, 5: 0, 6: 1, 7: None, 8: 4, 9: 7},
8: {0: 5, 1: 4, 2: 5, 3: 1, 4: 8, 5: 8, 6: 1, 7: 1, 8: None, 9: 4},
9: {0: 2, 1: 4, 2: 9, 3: 2, 4: 9, 5: 2, 6: 3, 7: 9, 8: 4, 9: None}})

Figura A.4: Archivo de la ruta de A.3

corta a seguir es $3 \rightarrow 1 \rightarrow 2$ (Figura A.6). En este paso nos movemos al nodo 1, que será nuestro nodo origen para la siguiente iteración.

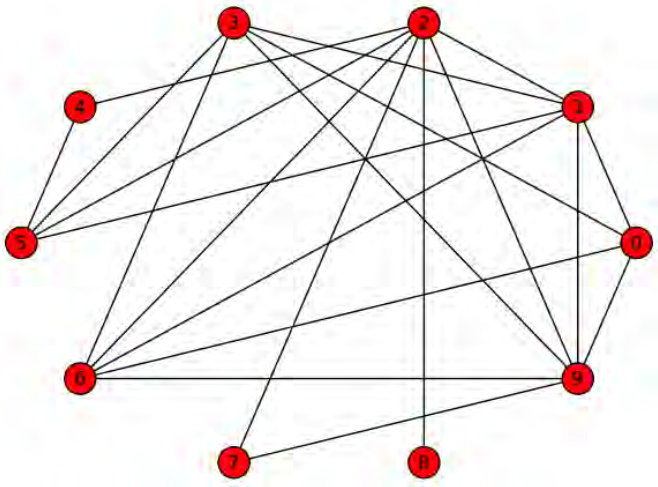

Figura A.5: Caso 2: Iteración 1, tiempo t_1

En la siguiente iteración (Figura A.7), tomando como origen el nodo 1, se calcula la ruta más corta ahora entre el nodo 1 y el nodo 2, el archivo de ruta (Figura A.8) indica que la ruta más corta entre estos nodos es $1 \rightarrow 0 \rightarrow 2$, en el tiempo t_2. Esto significa que la ruta calculada en el paso anterior, $3 \rightarrow 1 \rightarrow 2$, ya no es valida. En este paso nos movemos al nodo 0 que se convertirá en el nodo origen para la siguiente iteración.

En el tiempo t_3 (Figura A.9), tomando como origen el nodo 0, ahora se calcula la ruta más corta

({
0: {0: 0, 1: 1, 2: 2, 3: 1, 4: 3, 5: 2, 6: 1, 7: 2, 8: 3, 9: 1},
1: {0: 1, 1: 0, 2: 1, 3: 1, 4: 2, 5: 1, 6: 1, 7: 2, 8: 2, 9: 1},
2: {0: 2, 1: 1, 2: 0, 3: 2, 4: 1, 5: 1, 6: 1, 7: 1, 8: 1, 9: 1},
3: {0: 1, 1: 1, 2: 2, 3: 0, 4: 2, 5: 1, 6: 1, 7: 2, 8: 3, 9: 1},
4: {0: 3, 1: 2, 2: 1, 3: 2, 4: 0, 5: 1, 6: 2, 7: 2, 8: 2, 9: 2},
5: {0: 2, 1: 1, 2: 1, 3: 1, 4: 1, 5: 0, 6: 2, 7: 2, 8: 2, 9: 2},
6: {0: 1, 1: 1, 2: 1, 3: 1, 4: 2, 5: 2, 6: 0, 7: 2, 8: 2, 9: 1},
7: {0: 2, 1: 2, 2: 1, 3: 2, 4: 2, 5: 2, 6: 2, 7: 0, 8: 2, 9: 1},
8: {0: 3, 1: 2, 2: 1, 3: 3, 4: 2, 5: 2, 6: 2, 7: 2, 8: 0, 9: 2},
9: {0: 1, 1: 1, 2: 1, 3: 1, 4: 2, 5: 2, 6: 1, 7: 1, 8: 2, 9: 0}},

{
0: {0: None, 1: 0, 2: 1, 3: 0, 4: 2, 5: 1, 6: 0, 7: 9, 8: 2, 9: 0},
1: {0: 1, 1: None, 2: 1, 3: 1, 4: 2, 5: 1, 6: 1, 7: 2, 8: 2, 9: 1},
2: {0: 1, 1: 2, 2: None, 3: 1, 4: 2, 5: 2, 6: 2, 7: 2, 8: 2, 9: 2},
3: {0: 3, 1: 3, 2: 1, 3: None, 4: 5, 5: 3, 6: 3, 7: 9, 8: 2, 9: 3},
4: {0: 1, 1: 2, 2: 4, 3: 5, 4: None, 5: 4, 6: 2, 7: 2, 8: 2, 9: 2},
5: {0: 1, 1: 5, 2: 5, 3: 5, 4: 5, 5: None, 6: 1, 7: 2, 8: 2, 9: 1},
6: {0: 6, 1: 6, 2: 6, 3: 6, 4: 2, 5: 1, 6: None, 7: 2, 8: 2, 9: 6},
7: {0: 9, 1: 2, 2: 7, 3: 9, 4: 2, 5: 2, 6: 2, 7: None, 8: 2, 9: 7},
8: {0: 1, 1: 2, 2: 8, 3: 1, 4: 2, 5: 2, 6: 2, 7: 2, 8: None, 9: 2},
9: {0: 9, 1: 9, 2: 9, 3: 9, 4: 2, 5: 1, 6: 9, 7: 9, 8: 2, 9: None}})

Figura A.6: Archivo de la ruta de A.15

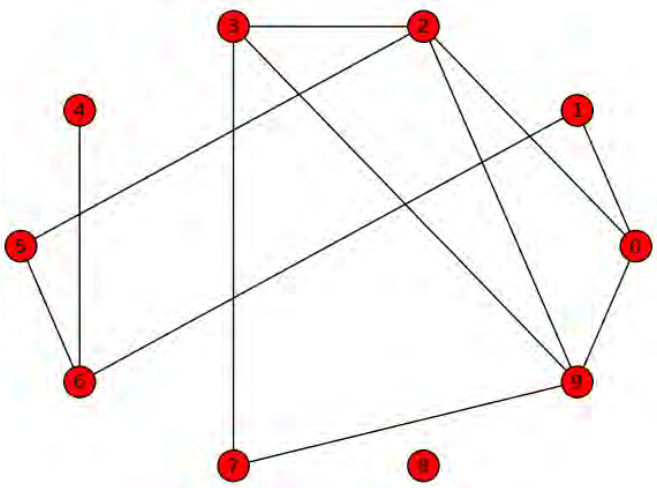

Figura A.7: Caso 2: Iteración 2, tiempo t_2

entre el nodo 0 y el nodo 2. El archivo de ruta (Figura A.10) muestra que el camino más corto es $0 \rightarrow 1 \rightarrow 2$. En este paso nos movemos al nodo 1 y lo tomaremos como origen en la siguiente iteración.

Para el tiempo t_4 (Figura A.11), tomando como origen el nodo 1, se calcula la ruta más corta entre este nodo y el nodo destino. En el archivo de rutas (Figura A.12) nos indica que hay un enlace directo entre estos nodos. Por lo tanto en el tiempo t_4 se ha alcanzado al nodo destino.

En este caso la ruta final fue $3 \rightarrow 1 \rightarrow 0 \rightarrow 1 \rightarrow 2$, la cual es más larga que la calculada en el tiempo t_1.

({
0:{0: 0, 1: 1, 2: 1, 3: 2, 4: 3, 5: 2, 6: 2, 7: 2, 8: inf, 9: 1},
1: {0: 1, 1: 0, 2: 2, 3: 3, 4: 2, 5: 2, 6: 1, 7: 3, 8: inf, 9: 2},
2: {0: 1, 1: 2, 2: 0, 3: 1, 4: 3, 5: 1, 6: 2, 7: 2, 8: inf, 9: 1},
3: {0: 2, 1: 3, 2: 1, 3: 0, 4: 4, 5: 2, 6: 3, 7: 1, 8: inf, 9: 1},
4: {0: 3, 1: 2, 2: 3, 3: 4, 4: 0, 5: 2, 6: 1, 7: 5, 8: inf, 9: 4},
5: {0: 2, 1: 2, 2: 1, 3: 2, 4: 2, 5: 0, 6: 1, 7: 3, 8: inf, 9: 2},
6: {0: 2, 1: 1, 2: 2, 3: 3, 4: 1, 5: 1, 6: 0, 7: 4, 8: inf, 9: 3},
7: {0: 2, 1: 3, 2: 2, 3: 1, 4: 5, 5: 3, 6: 4, 7: 0, 8: inf, 9: 1},
8: {0: inf, 1: inf, 2: inf, 3: inf, 4: inf, 5: inf, 6: inf, 7: inf, 8: 0, 9: inf},
9: {0: 1, 1: 2, 2: 1, 3: 1, 4: 4, 5: 2, 6: 3, 7: 1, 8: inf, 9: 0}},

{
0: {0: None, 1: 0, 2: 0, 3: 2, 4: 6, 5: 2, 6: 1, 7: 9, 8: None, 9: 0},
1: {0: 1, 1: None, 2: 0, 3: 2, 4: 6, 5: 6, 6: 1, 7: 9, 8: None, 9: 0},
2: {0: 2, 1: 0, 2: None, 3: 2, 4: 6, 5: 2, 6: 5, 7: 3, 8: None, 9: 2},
3: {0: 2, 1: 0, 2: 3, 3: None, 4: 6, 5: 2, 6: 5, 7: 3, 8: None, 9: 3},
4: {0: 1, 1: 6, 2: 5, 3: 2, 4: None, 5: 6, 6: 4, 7: 3, 8: None, 9: 0},
5: {0: 2, 1: 6, 2: 5, 3: 2, 4: 6, 5: None, 6: 5, 7: 3, 8: None, 9: 2},
6: {0: 1, 1: 6, 2: 5, 3: 2, 4: 6, 5: 6, 6: None, 7: 3, 8: None, 9: 0},
7: {0: 9, 1: 0, 2: 3, 3: 7, 4: 6, 5: 2, 6: 5, 7: None, 8: None, 9: 7},
8: {0: None, 1: None, 2: None, 3: None, 4: None, 5: None, 6: None, 7: None, 8: None, 9: None},
9: {0: 9, 1: 0, 2: 9, 3: 9, 4: 6, 5: 2, 6: 1, 7: 9, 8: None, 9: None}})

Figura A.8: Archivo de la ruta de A.7

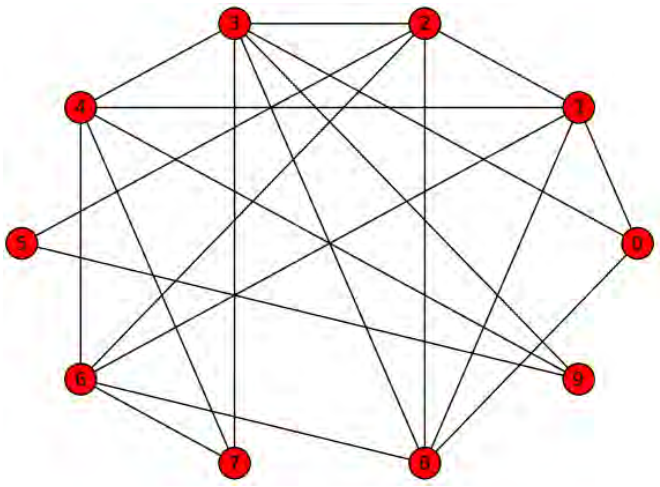

Figura A.9: Caso 2: Iteración 3, tiempo t_3

(
{
0: {0: 0, 1: 1, 2: 2, 3: 1, 4: 2, 5: 3, 6: 2, 7: 2, 8: 1, 9: 2},
1: {0: 1, 1: 0, 2: 1, 3: 2, 4: 1, 5: 2, 6: 1, 7: 2, 8: 1, 9: 2},
2: {0: 2, 1: 1, 2: 0, 3: 1, 4: 2, 5: 1, 6: 1, 7: 2, 8: 1, 9: 2},
3: {0: 1, 1: 2, 2: 1, 3: 0, 4: 1, 5: 2, 6: 2, 7: 1, 8: 1, 9: 1},
4: {0: 2, 1: 1, 2: 2, 3: 1, 4: 0, 5: 2, 6: 1, 7: 1, 8: 2, 9: 1},
5: {0: 3, 1: 2, 2: 1, 3: 2, 4: 2, 5: 0, 6: 2, 7: 3, 8: 2, 9: 1},
6: {0: 2, 1: 1, 2: 1, 3: 2, 4: 1, 5: 2, 6: 0, 7: 1, 8: 1, 9: 2},
7: {0: 2, 1: 2, 2: 2, 3: 1, 4: 1, 5: 3, 6: 1, 7: 0, 8: 2, 9: 2},
8: {0: 1, 1: 1, 2: 1, 3: 1, 4: 2, 5: 2, 6: 1, 7: 2, 8: 0, 9: 2},
9: {0: 2, 1: 2, 2: 2, 3: 1, 4: 1, 5: 1, 6: 2, 7: 2, 8: 2, 9: 0}},

{
0: {0: None, 1: 0, 2: 1, 3: 0, 4: 1, 5: 2, 6: 1, 7: 3, 8: 0, 9: 3},
1: {0: 1, 1: None, 2: 1, 3: 0, 4: 1, 5: 2, 6: 1, 7: 4, 8: 1, 9: 4},
2: {0: 1, 1: 2, 2: None, 3: 2, 4: 1, 5: 2, 6: 2, 7: 3, 8: 2, 9: 3},
3: {0: 3, 1: 0, 2: 3, 3: None, 4: 3, 5: 2, 6: 2, 7: 3, 8: 3, 9: 3},
4: {0: 1, 1: 4, 2: 1, 3: 4, 4: None, 5: 9, 6: 4, 7: 4, 8: 1, 9: 4},
5: {0: 1, 1: 2, 2: 5, 3: 2, 4: 9, 5: None, 6: 2, 7: 3, 8: 2, 9: 5},
6: {0: 1, 1: 6, 2: 6, 3: 2, 4: 6, 5: 2, 6: None, 7: 6, 8: 6, 9: 4},
7: {0: 3, 1: 4, 2: 3, 3: 7, 4: 7, 5: 2, 6: 7, 7: None, 8: 3, 9: 3},
8: {0: 8, 1: 8, 2: 8, 3: 8, 4: 1, 5: 2, 6: 8, 7: 3, 8: None, 9: 3},
9: {0: 3, 1: 4, 2: 3, 3: 9, 4: 9, 5: 9, 6: 4, 7: 3, 8: 3, 9: None}})

Figura A.10: Archivo de la ruta de A.9

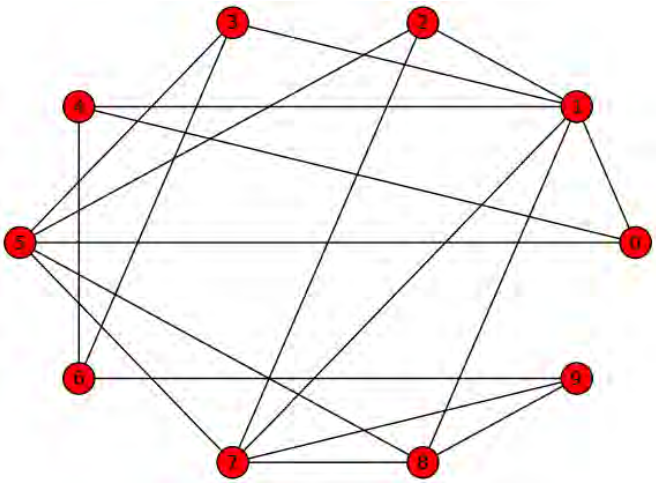

Figura A.11: Caso 2: Iteración 4, tiempo t_4

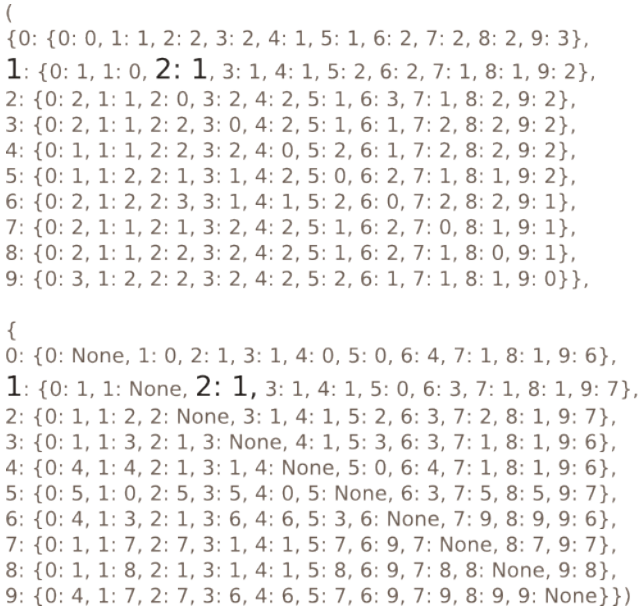

Figura A.12: Archivo de la ruta de A.11

A.3. La ruta es más corta

En este caso (Figura A.13), suponemos que nuestro nodo origen es el nodo 8 y el nodo destino es el nodo 7. En el tiempo t_1 el algoritmo Floyd-Warshall calcula que son necesarios 3 saltos y la ruta más corta a seguir es $8 \rightarrow 0 \rightarrow 3 \rightarrow 7$ (Figura A.14). En este paso nos movemos al nodo 0, que será nuestro nodo origen para la siguiente iteración.

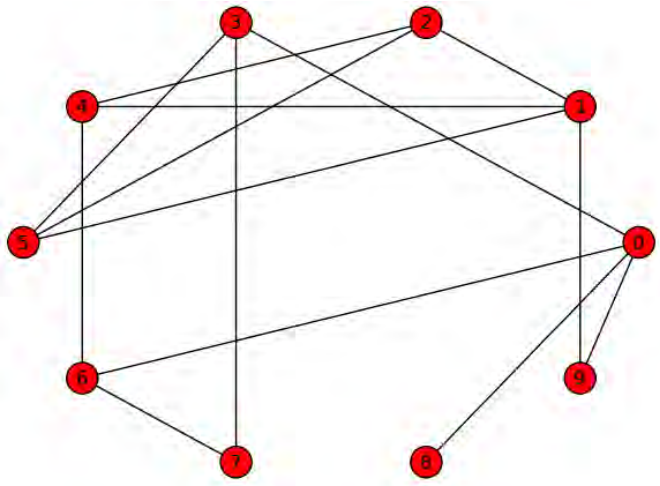

Figura A.13: Caso 3: Iteración 1, tiempo t_1

(({
0: {0: 0, 1: 2, 2: 3, 3: 1, 4: 2, 5: 2, 6: 1, 7: 2, 8: 1, 9: 1},
1: {0: 2, 1: 0, 2: 1, 3: 2, 4: 1, 5: 1, 6: 2, 7: 3, 8: 3, 9: 1},
2: {0: 3, 1: 1, 2: 0, 3: 2, 4: 1, 5: 1, 6: 2, 7: 3, 8: 4, 9: 2},
3: {0: 1, 1: 2, 2: 2, 3: 0, 4: 3, 5: 1, 6: 2, 7: 1, 8: 2, 9: 2},
4: {0: 2, 1: 1, 2: 1, 3: 3, 4: 0, 5: 2, 6: 1, 7: 2, 8: 3, 9: 2},
5: {0: 2, 1: 1, 2: 1, 3: 1, 4: 2, 5: 0, 6: 3, 7: 2, 8: 3, 9: 2},
6: {0: 1, 1: 2, 2: 2, 3: 2, 4: 1, 5: 3, 6: 0, 7: 1, 8: 2, 9: 2},
7: {0: 2, 1: 3, 2: 3, 3: 1, 4: 2, 5: 2, 6: 1, 7: 0, 8: 3, 9: 3},
8: {0: 1, 1: 3, 2: 4, 3: 2, 4: 3, 5: 3, 6: 2, 7: 3, 8: 0, 9: 2},
9: {0: 1, 1: 1, 2: 2, 3: 2, 4: 2, 5: 2, 6: 2, 7: 3, 8: 2, 9: 0}},

{0:
{
0: None, 1: 9, 2: 5, 3: 0, 4: 6, 5: 3, 6: 0, 7: 3, 8: 0, 9: 0},
1: {0: 9, 1: None, 2: 1, 3: 5, 4: 1, 5: 1, 6: 4, 7: 3, 8: 0, 9: 1},
2: {0: 3, 1: 2, 2: None, 3: 5, 4: 2, 5: 2, 6: 4, 7: 3, 8: 0, 9: 1},
3: {0: 3, 1: 5, 2: 5, 3: None, 4: 1, 5: 3, 6: 0, 7: 3, 8: 0, 9: 0},
4: {0: 6, 1: 4, 2: 4, 3: 5, 4: None, 5: 1, 6: 4, 7: 6, 8: 0, 9: 1},
5: {0: 3, 1: 5, 2: 5, 3: 5, 4: 1, 5: None, 6: 0, 7: 3, 8: 0, 9: 1},
6: {0: 6, 1: 4, 2: 4, 3: 0, 4: 6, 5: 3, 6: None, 7: 6, 8: 0, 9: 0},
7: {0: 3, 1: 5, 2: 5, 3: 7, 4: 6, 5: 3, 6: 7, 7: None, 8: 0, 9: 0},
8: {0: 8, 1: 9, 2: 5, 3: 0, 4: 6, 5: 3, 6: 0, 7: 3, 8: None, 9: 0},
9: {0: 9, 1: 9, 2: 1, 3: 0, 4: 1, 5: 1, 6: 0, 7: 3, 8: 0, 9: None}})

Figura A.14: Archivo de la ruta de A.13

En la siguiente iteración (Figura A.15), tomando como origen el nodo 0, se vuelve a calcular la ruta más corta ahora entre el nodo 0 y el nodo 7, el archivo de ruta (Figura A.16) indica que existe

un camino directo de $0 \rightarrow 7$ en el tiempo t_2. Esto significa que la ruta calculada en el paso anterior, $8 \rightarrow 0 \rightarrow 3 \rightarrow 7$, ya no es valida y que en lugar de necesitar 3 saltos para llegar al destino, solo se necesitaron 2 con la ruta $8 \rightarrow 0 \rightarrow 7$

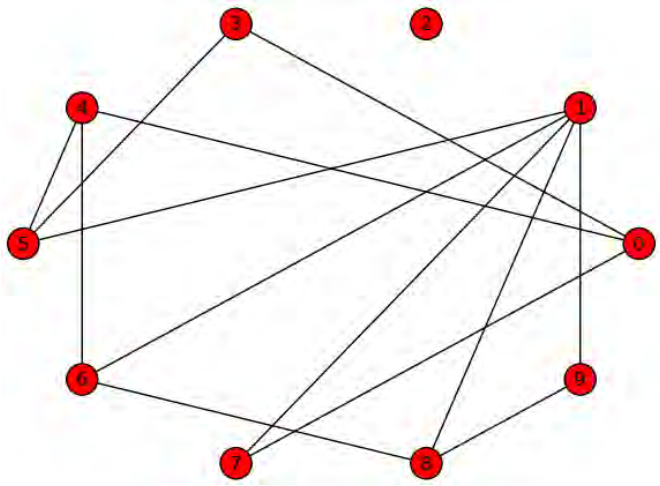

Figura A.15: Caso 3: Iteración 2, tiempo t_2

({
0: {0: 0, 1: 2, 2: inf, 3: 1, 4: 1, 5: 2, 6: 2, 7: 1, 8: 3, 9: 3},
1: {0: 2, 1: 0, 2: inf, 3: 2, 4: 2, 5: 1, 6: 1, 7: 1, 8: 1, 9: 1},
2: {0: inf, 1: inf, 2: 0, 3: inf, 4: inf, 5: inf, 6: inf, 7: inf, 8: inf, 9: inf},
3: {0: 1, 1: 2, 2: inf, 3: 0, 4: 2, 5: 1, 6: 3, 7: 2, 8: 3, 9: 3},
4: {0: 1, 1: 2, 2: inf, 3: 2, 4: 0, 5: 1, 6: 1, 7: 2, 8: 2, 9: 3},
5: {0: 2, 1: 1, 2: inf, 3: 1, 4: 1, 5: 0, 6: 2, 7: 2, 8: 2, 9: 2},
6: {0: 2, 1: 1, 2: inf, 3: 3, 4: 1, 5: 2, 6: 0, 7: 2, 8: 1, 9: 2},
7: {0: 1, 1: 1, 2: inf, 3: 2, 4: 2, 5: 2, 6: 2, 7: 0, 8: 2, 9: 2},
8: {0: 3, 1: 1, 2: inf, 3: 3, 4: 2, 5: 2, 6: 1, 7: 2, 8: 0, 9: 1},
9: {0: 3, 1: 1, 2: inf, 3: 3, 4: 3, 5: 2, 6: 2, 7: 2, 8: 1, 9: 0}},

{
0: {0: None, 1: 7, 2: None, 3: 0, 4: 0, 5: 3, 6: 4, 7: 0, 8: 6, 9: 1},
1: {0: 7, 1: None, 2: None, 3: 5, 4: 5, 5: 1, 6: 1, 7: 1, 8: 1, 9: 1},
2: {0: None, 1: None, 2: None, 3: None, 4: None, 5: None, 6: None, 7: None, 8: None, 9: None},
3: {0: 3, 1: 5, 2: None, 3: None, 4: 0, 5: 3, 6: 4, 7: 0, 8: 1, 9: 1},
4: {0: 4, 1: 5, 2: None, 3: 0, 4: None, 5: 4, 6: 4, 7: 0, 8: 6, 9: 1},
5: {0: 3, 1: 5, 2: None, 3: 5, 4: 5, 5: None, 6: 1, 7: 1, 8: 1, 9: 1},
6: {0: 4, 1: 6, 2: None, 3: 0, 4: 6, 5: 1, 6: None, 7: 1, 8: 6, 9: 1},
7: {0: 7, 1: 7, 2: None, 3: 0, 4: 0, 5: 1, 6: 1, 7: None, 8: 1, 9: 1},
8: {0: 4, 1: 8, 2: None, 3: 5, 4: 6, 5: 1, 6: 8, 7: 1, 8: None, 9: 8},
9: {0: 7, 1: 9, 2: None, 3: 5, 4: 5, 5: 1, 6: 1, 7: 1, 8: 9, 9: None}})

Figura A.16: Archivo de la ruta de A.15

A.4. La ruta es infinita

En este caso (Figura A.17), suponemos que nuestro nodo origen es el nodo 0 y el nodo destino es el nodo 8. En el tiempo t_1 el algoritmo Floyd-Warshall calcula que son necesarios 3 saltos y la ruta más corta a seguir es $0 \rightarrow 1 \rightarrow 2 \rightarrow 8$ (Figura A.18). En este paso nos movemos al nodo 1, que será nuestro nodo origen para la siguiente iteración.

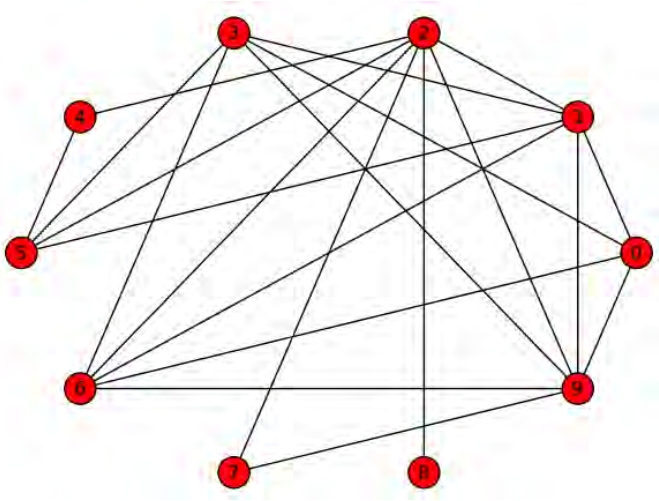

Figura A.17: Caso 4: Iteración 1, tiempo t_1

```
(({
0: {0: 0, 1: 1, 2: 2, 3: 1, 4: 3, 5: 2, 6: 1, 7: 2, 8: 3, 9: 1},
1: {0: 1, 1: 0, 2: 1, 3: 1, 4: 2, 5: 1, 6: 1, 7: 2, 8: 2, 9: 1},
2: {0: 2, 1: 1, 2: 0, 3: 2, 4: 1, 5: 1, 6: 1, 7: 1, 8: 1, 9: 1},
3: {0: 1, 1: 1, 2: 2, 3: 0, 4: 2, 5: 1, 6: 1, 7: 2, 8: 3, 9: 1},
4: {0: 3, 1: 2, 2: 1, 3: 2, 4: 0, 5: 1, 6: 2, 7: 2, 8: 2, 9: 2},
5: {0: 2, 1: 1, 2: 1, 3: 1, 4: 1, 5: 0, 6: 2, 7: 2, 8: 2, 9: 2},
6: {0: 1, 1: 1, 2: 1, 3: 1, 4: 2, 5: 2, 6: 0, 7: 2, 8: 2, 9: 1},
7: {0: 2, 1: 2, 2: 1, 3: 2, 4: 2, 5: 2, 6: 2, 7: 0, 8: 2, 9: 1},
8: {0: 3, 1: 2, 2: 1, 3: 3, 4: 2, 5: 2, 6: 2, 7: 2, 8: 0, 9: 2},
9: {0: 1, 1: 1, 2: 1, 3: 1, 4: 2, 5: 2, 6: 1, 7: 1, 8: 2, 9: 0}},

{
0: {0: None, 1: 0, 2: 1, 3: 0, 4: 2, 5: 1, 6: 0, 7: 9, 8: 2, 9: 0},
1: {0: 1, 1: None, 2: 1, 3: 1, 4: 2, 5: 1, 6: 1, 7: 2, 8: 2, 9: 1},
2: {0: 1, 1: 2, 2: None, 3: 1, 4: 2, 5: 2, 6: 2, 7: 2, 8: 2, 9: 2},
3: {0: 3, 1: 3, 2: 1, 3: None, 4: 5, 5: 3, 6: 3, 7: 9, 8: 2, 9: 3},
4: {0: 1, 1: 2, 2: 4, 3: 5, 4: None, 5: 4, 6: 2, 7: 2, 8: 2, 9: 2},
5: {0: 1, 1: 5, 2: 5, 3: 5, 4: 5, 5: None, 6: 1, 7: 2, 8: 2, 9: 1},
6: {0: 6, 1: 6, 2: 6, 3: 6, 4: 2, 5: 1, 6: None, 7: 2, 8: 2, 9: 6},
7: {0: 9, 1: 2, 2: 7, 3: 9, 4: 2, 5: 2, 6: 2, 7: None, 8: 2, 9: 7},
8: {0: 1, 1: 2, 2: 8, 3: 1, 4: 2, 5: 2, 6: 2, 7: 2, 8: None, 9: 2},
9: {0: 9, 1: 9, 2: 9, 3: 9, 4: 2, 5: 1, 6: 9, 7: 9, 8: 2, 9: None}})
```

Figura A.18: Archivo de la ruta de A.17

En la siguiente iteración (Figura A.19), en el tiempo t_2 considerando como origen al nodo 1, se calcula la ruta más corta entre los nodos 1 y 8, sin embargo el archivo de rutas (Figura A.20) nos indica que la ruta es infinita, es decir que no existe ningún camino hacia el nodo 8 en este tiempo.

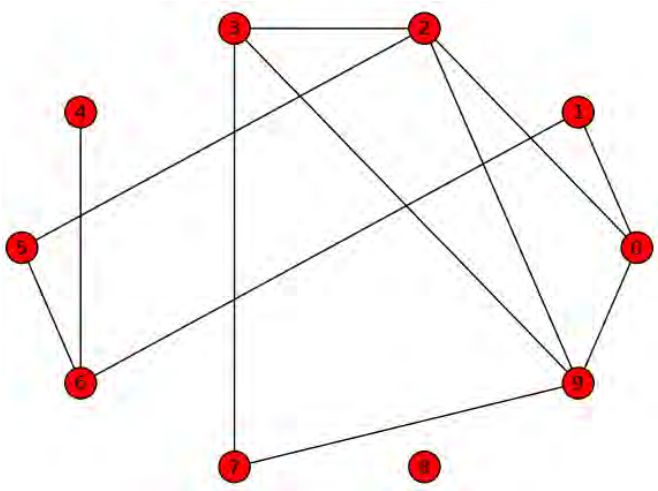

Figura A.19: Caso 4: Iteración 2, tiempo t_2

({
0:{0: 0, 1: 1, 2: 1, 3: 2, 4: 3, 5: 2, 6: 2, 7: 2, 8: inf, 9: 1},
1: {0: 1, 1: 0, 2: 2, 3: 3, 4: 2, 5: 2, 6: 1, 7: 3, 8: inf, 9: 2},
2: {0: 1, 1: 2, 2: 0, 3: 1, 4: 3, 5: 1, 6: 2, 7: 2, 8: inf, 9: 1},
3: {0: 2, 1: 3, 2: 1, 3: 0, 4: 4, 5: 2, 6: 3, 7: 1, 8: inf, 9: 1},
4: {0: 3, 1: 2, 2: 3, 3: 4, 4: 0, 5: 2, 6: 1, 7: 5, 8: inf, 9: 4},
5: {0: 2, 1: 2, 2: 1, 3: 2, 4: 2, 5: 0, 6: 1, 7: 3, 8: inf, 9: 2},
6: {0: 2, 1: 1, 2: 2, 3: 3, 4: 1, 5: 1, 6: 0, 7: 4, 8: inf, 9: 3},
7: {0: 2, 1: 3, 2: 2, 3: 1, 4: 5, 5: 3, 6: 4, 7: 0, 8: inf, 9: 1},
8: {0: inf, 1: inf, 2: inf, 3: inf, 4: inf, 5: inf, 6: inf, 7: inf, 8: 0, 9: inf},
9: {0: 1, 1: 2, 2: 1, 3: 1, 4: 4, 5: 2, 6: 3, 7: 1, 8: inf, 9: 0}},

{
0: {0: None, 1: 0, 2: 0, 3: 2, 4: 6, 5: 2, 6: 1, 7: 9, 8: None, 9: 0},
1: {0: 1, 1: None, 2: 0, 3: 2, 4: 6, 5: 6, 6: 1, 7: 9, 8: None, 9: 0},
2: {0: 2, 1: 0, 2: None, 3: 2, 4: 6, 5: 2, 6: 5, 7: 3, 8: None, 9: 2},
3: {0: 2, 1: 0, 2: 3, 3: None, 4: 6, 5: 2, 6: 5, 7: 3, 8: None, 9: 3},
4: {0: 1, 1: 6, 2: 5, 3: 2, 4: None, 5: 6, 6: 4, 7: 3, 8: None, 9: 0},
5: {0: 2, 1: 6, 2: 5, 3: 2, 4: 6, 5: None, 6: 5, 7: 3, 8: None, 9: 2},
6: {0: 1, 1: 6, 2: 5, 3: 2, 4: 6, 5: 6, 6: None, 7: 3, 8: None, 9: 0},
7: {0: 9, 1: 0, 2: 3, 3: 7, 4: 6, 5: 2, 6: 5, 7: None, 8: None, 9: 7},
8: {0: None, 1: None, 2: None, 3: None, 4: None, 5: None, 6: None, 7: None, 8: None, 9: None},
9: {0: 9, 1: 0, 2: 9, 3: 9, 4: 6, 5: 2, 6: 1, 7: 9, 8: None, 9: None}})

Figura A.20: Archivo de la ruta de la Figura A.19

Apéndice B

CÓDIGO

Listing B.1: Código completo

```
1  # -*- coding: utf-8 -*-
2
3  ##################################################
4  #Calcula dato es una funci[U+FFFD]n que determina el valor
5  #de una celda de una matriz de adyacencia
6  ##################################################
7  def calcula_dato (i,j,carga,cota,qos,hops):
8      import random
9      #simulando un procesador
10     tareas=random.randint(40,800) # 40 tareas por default maximo 800
11     x=50 #espacio para transmitir
12     #print "tareas", tareas
13     y=800-tareas
14     if (y>=x):#si queda espacio transmito
15       #print "si transmito"
16       delta_n=random.uniform(0,0.05) #p[U+FFFD]rdida en el canal
17       if (int(i)==int(j)):
18               dato=0
19       else:
20               dato= math.exp((y**2)+(delta_n**2)+(hops**2)+(carga**2))
21     else:
22       dato=0
23     return(dato)
24 ##################################################
25
26 import math
27 import random
28 import networkx as nx
29 import matplotlib.pyplot as plt
30 import sys
31 import time
32 sys.argv
33 if (len(sys.argv)!=2):
34           print "error de parametros"
```

```
35              print "uso:", sys.argv[0], "n"
36              sys.exit(-1)
37
38  n=int(sys.argv[1])
39  #carga=random.uniform(0,0.6)
40  carga=0.3
41  #cota=random.uniform(0,1)
42  cota=0.5
43  qos=0.02
44  hops=3
45  ##Tiempo
46  t1=time.time()
47  name="file_"+str(n)+".txt"
48  f = open(name,"a")
49  rutas="ruta_"+str(n)+".txt"
50  h = open(rutas,"a")
51  tiempo="tiempo.tx"
52  t = open(tiempo, "a")
53  print "carga \n", carga
54  print "cota \n", cota
55  print "cota del QoS\n", qos
56  #h=calcula_dato(1,2)
57  #print h
58  mat          =[[0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0],
59                 [0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0],
60                 [0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0],
61             [0,0,0,0,0,0,0,0,0,0]]
62  for i in range(10):
63      for row in range(10):
64          #mat[i][i]=calcula_dato(i,i)
65          mat[row][i]=mat[i][row]=calcula_dato(row,i,carga,cota,qos,hops)
66
67  f.writelines("m=[\n")
68  for i in range(10):
69    for row in mat:
70      f.writelines(str(row[i])+" ")
71      print row[i],
72    f.writelines("; \n"),
73    print
74  f.writelines("]\n \n")
75  b=[]
76  for i in range(10):
77      for row in range(10):
78        if mat[row][i]==1:
79          #print "(",i,",", row,")\n"
80          b.append((i,row))
81  a =[0,1,2,3,4,5,6,7,8,9]
82  G=nx.Graph()
83  G.add_nodes_from(a)
84  G.add_edges_from(b)
85  nx.draw_circular(G)
86  h.writelines(str(nx.floyd_warshall(G)))
87  plt.savefig("file_"+str(n)+".png") # save as png
```

```
88  ##Tiempo
89  t2=time.time()
90  t3=t2-t1
91  t.writelines(str(t3)+"\n")
92  f.close()
93  h.close()
94  t.close()
```

Índice alfabético