Universidad del Valle de Guatemala
Facultad de ingeniería

Redes
Laboratorio 3.1 - Informe
Algoritmos de enrutamiento

Marco Orozco
Jun Woo Lee
Andrés de la Roca

Guatemala, 2023

Descripción de la práctica

Para esta práctica se busca entender y utilizar en la práctica los diferentes algoritmos de enrutamiento, se quiere en específico conocer las implementaciones más utilizadas actualmente en las redes, como Dijkstra, Flooding, etc. Se quiere comprender como funciona en detalle las tablas de enrutamiento y como estas son implementadas dentro de los propios algoritmos y como cada algoritmo utiliza las tablas de una forma diferente.
En concreto durante la práctica lo que se realizará es utilizar los algoritmos de enrutamiento que tienen nodos interconectados para enviar información, por lo cual a partir del funcionamiento general de estos algoritmos se simulara de forma manual el envío y recepción de los mensajes que los nodos realizan entre sí para conformar sus tablas de enrutamiento.

Dijkstra
El algoritmo de Dijkstra es un método de enrutamiento. Busca encontrar el camino más corto entre un nodo de origen y un nodo de destino en una red, en lugar de simplemente retransmitir mensajes a todos los nodos. Utiliza información de distancia acumulada desde el nodo de origen hacia otros nodos para determinar la mejor ruta. A medida que se propaga la información de distancia, se establece un camino óptimo hacia el nodo de destino.

Flooding
Este es un método simple de retransmisión, considerado ampliamente como ineficiente, en este algoritmo, cuando un nodo en la red quiere enviar un mensaje a otro nodo, en lugar de tomar decisiones de enrutamiento inteligentes, simplemente elige retransmitir el mensaje a todos los nodos dentro de la red, primero empezando a enviar el mensaje por medio de sus nodos vecinos y esperar que el mensaje llegue a su destino.
Cada nodo vecino que recibe el mensaje retransmite a todos sus nodos vecinos, lo que crea una propagación en cadena del mensaje por toda la red, eventualmente, el mensaje podrá llegar al destino B, pero no sin antes haber recorrido toda la red. En algunos casos se pueden implementar mecanismo dentro del algoritmo para limitar la congestión de la red que causa el estar reenviando mensajes a todos los nodos.
Aunque el algoritmo Flooding puede resultar útil en algunos casos en donde la red es altamente dinámica y no se pueden mantener tablas de enrutamiento propias de otros algoritmos de enrutamiento.

Distance Vector Routing

El algoritmo de enrutamiento por vector de distancia es uno de los métodos clásicos utilizados en redes para determinar el mejor camino para enviar paquetes de datos entre nodos. Se basa en la idea de que cada router (o nodo) mantiene una tabla que indica la mejor distancia conocida a cada destino y a través de qué vecino se debe enviar para lograr esa distancia.

Resultados

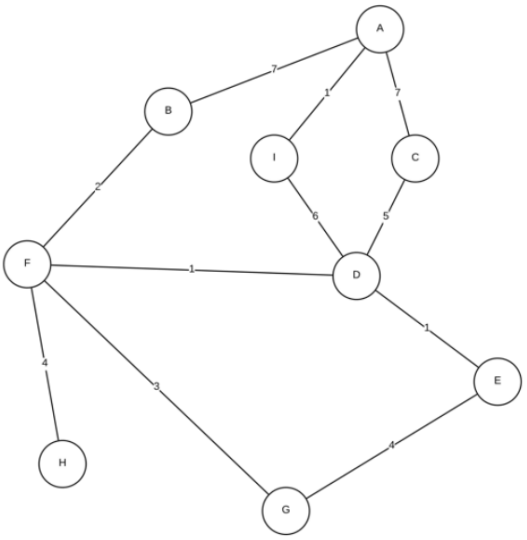


Fig. 1. Mapa de conexiones propuesto

Flooding

De acuerdo al mapa de conexiones propuesto en la figura 1, se obtuvieron los siguientes resultados al momento de realizar las pruebas.

```
--- Algoritmo Flooding ---
Ingrese el identificador de este nodo: A
Ingrese los identificadores de los nodos vecinos separados por coma: B,I,C
Nodo creado:  node_flooding { id: 'A', neighbors: [ 'B', 'I', 'C' ] }
--- Menu del nodo A ---
1. Enviar mensaje
2. Recibir mensaje
3. Salir
Ingrese la opción: 1
Ingrese el mensaje a enviar: Hello World
Mensaje enviado desde A hacia B: Hello World
Mensaje enviado desde A hacia I: Hello World
Mensaje enviado desde A hacia C: Hello World
```

Nodo A

```
--- Algoritmo Flooding ---
Ingrese el identificador de este nodo: B
Ingrese los identificadores de los nodos vecinos separados por coma: A,F
Nodo creado:  node_flooding { id: 'B', neighbors: [ 'A', 'F' ] }
--- Menu del nodo B ---
1. Enviar mensaje
2. Recibir mensaje
3. Salir
Ingrese la opción: 2
Ingrese el identificador del nodo que envió el mensaje: A
Ingrese el mensaje recibido: Hello World
Mensaje recibido en nodo B desde nodo A: Hello World
Mensaje enviado desde B hacia F: Hello World
```

Nodo B

```
--- Algoritmo Flooding ---
Ingrese el identificador de este nodo: C
Ingrese los identificadores de los nodos vecinos separados por coma: A,D
Nodo creado:  node_flooding { id: 'C', neighbors: [ 'A', 'D' ] }
--- Menu del nodo C ---
1. Enviar mensaje
2. Recibir mensaje
3. Salir
Ingrese la opción: 2
Ingrese el identificador del nodo que envió el mensaje: A
Ingrese el mensaje recibido: Hello World
Mensaje recibido en nodo C desde nodo A: Hello World
Mensaje enviado desde C hacia D: Hello World
```

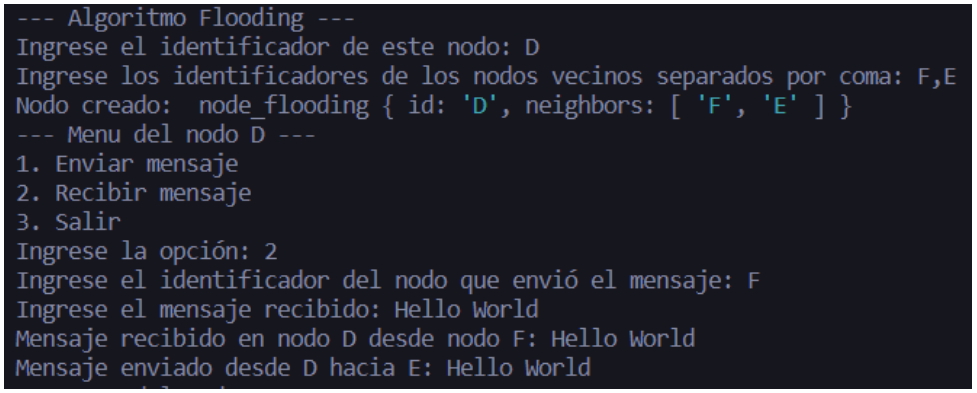
Nodo C

```
--- Algoritmo Flooding ---
Ingrese el identificador de este nodo: I
Ingrese los identificadores de los nodos vecinos separados por coma: A,D
Nodo creado:  node_flooding { id: 'I', neighbors: [ 'A', 'D' ] }
--- Menu del nodo I ---
1. Enviar mensaje
2. Recibir mensaje
3. Salir
Ingrese la opción: 2
Ingrese el identificador del nodo que envió el mensaje: A
Ingrese el mensaje recibido: Hello World
Mensaje recibido en nodo I desde nodo A: Hello World
Mensaje enviado desde I hacia D: Hello World
```

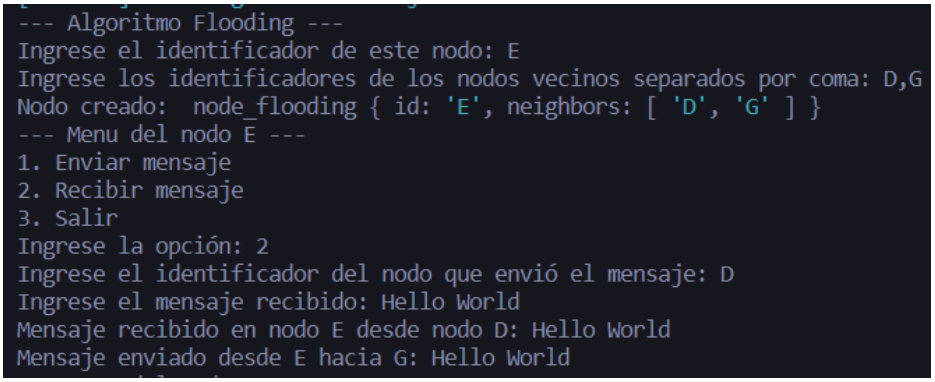
Nodo I

```
--- Algoritmo Flooding ---
Ingrese el identificador de este nodo: F
Ingrese los identificadores de los nodos vecinos separados por coma: B,D,G,H
Nodo creado:  node_flooding { id: 'F', neighbors: [ 'B', 'D', 'G', 'H' ] }
--- Menu del nodo F ---
1. Enviar mensaje
2. Recibir mensaje
3. Salir
Ingrese la opción: 2
Ingrese el identificador del nodo que envió el mensaje: B
Ingrese el mensaje recibido: Hello World
Mensaje recibido en nodo F desde nodo B: Hello World
Mensaje enviado desde F hacia D: Hello World
Mensaje enviado desde F hacia G: Hello World
Mensaje enviado desde F hacia H: Hello World
```

Nodo F



Nodo D



Nodo E

Como se puede apreciar, el algoritmo recorre toda la red que de nodos que se ha propuesto, para ciertas interacciones entre los nodos se omite el poner ciertos vecinos ya que estos envían el mismo mensaje al mismo nodo, por lo que se evita que se haga un bucle infinito al momento de enviar mensajes. Esta es una de las formas en las que se puede controlar el flujo de este algoritmo, por lo que se podría ver como un algoritmo de flooding “controlado”.

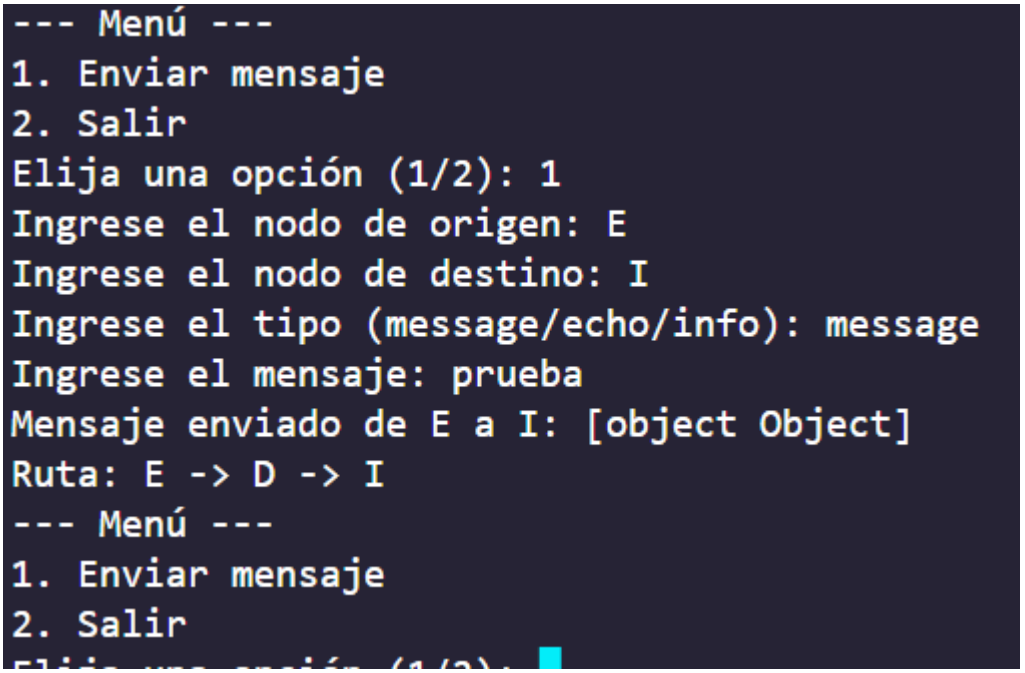
Dijsktra

```
graph.addNode("A");
graph.addNode("B");
graph.addNode("C");
graph.addNode("D");
graph.addNode("E");
graph.addNode("F");
graph.addNode("G");
graph.addNode("H");
graph.addNode("I");

graph.addEdge("A", "B", 7);
graph.addEdge("A", "C", 7);
graph.addEdge("A", "I", 1);
graph.addEdge("B", "F", 2);
graph.addEdge("F", "D", 1);
graph.addEdge("I", "D", 6);
graph.addEdge("C", "D", 5);
graph.addEdge("F", "H", 4);
graph.addEdge("F", "G", 3);
graph.addEdge("D", "E", 1);
```

```
--- Menú ---
1. Enviar mensaje
2. Salir
Elija una opción (1/2): 1
Ingrese el nodo de origen: A
Ingrese el nodo de destino: G
Ingrese el tipo (message/echo/info): info
Ingrese el mensaje: hola
Mensaje enviado de A a G: [object Object]
Ruta: A -> I -> D -> F -> G
--- Menú ---
1. Enviar mensaje
2. Salir
Elija una opción (1/2):
```

```
--- Menú ---
1. Enviar mensaje
2. Salir
Elija una opción (1/2): 1
Ingrese el nodo de origen: B
Ingrese el nodo de destino: D
Ingrese el tipo (message/echo/info): echo
Ingrese el mensaje: hey
Mensaje enviado de B a D: [object Object]
Ruta: B -> F -> D
--- Menú ---
1. Enviar mensaje
2. Salir
Elija una opción (1/2):
```

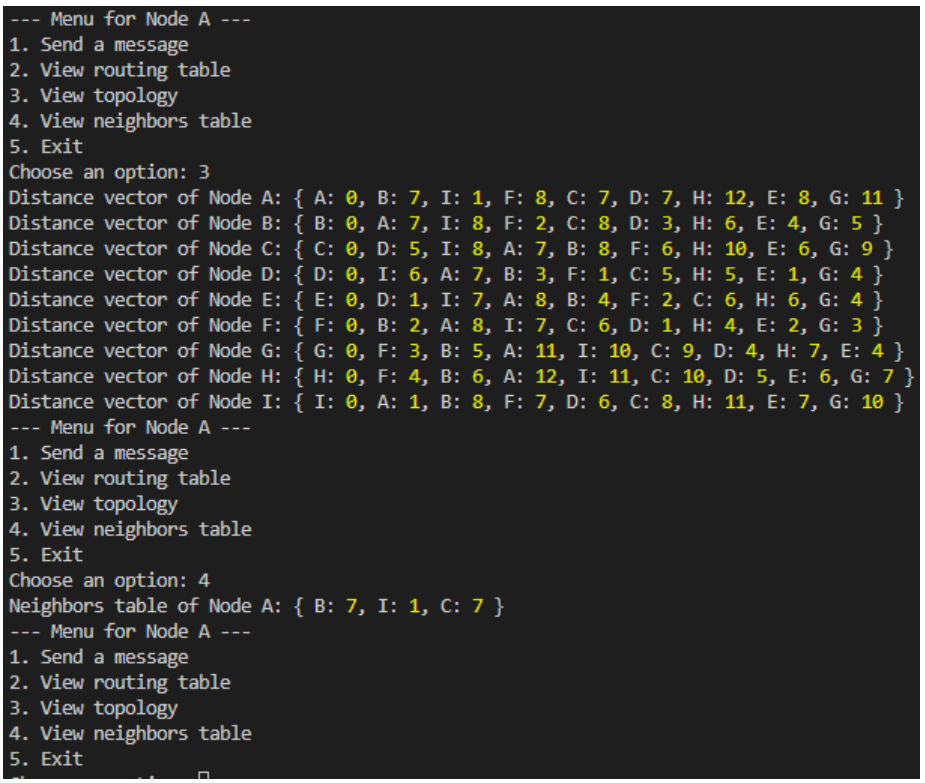


Se muestra el path de forma correcta y los mensajes se envían y reciben correctamente.

Distance Vector

```
const nodes = [
    new NodeDistanceVector('A', { 'B': 7, 'I': 1, 'C': 7 }),
    new NodeDistanceVector('B', { 'A': 7, 'F': 2 }),
    new NodeDistanceVector('C', { 'A': 7, 'D': 5 }),
    new NodeDistanceVector('D', { 'C': 5, 'I': 6, 'F': 1, 'E': 1 }),
    new NodeDistanceVector('E', { 'D': 1, 'G': 4 }),
    new NodeDistanceVector('F', { 'B': 2, 'D': 1, 'H': 4, 'G': 3 }),
    new NodeDistanceVector('G', { 'E': 4, 'F': 3 }),
    new NodeDistanceVector('H', { 'F': 4 }),
    new NodeDistanceVector('I', { 'A': 1, 'D': 6 })
];
```

```
Which node would you like to interact with (A-I)? A
--- Menu for Node A ---
1. Send a message
2. View routing table
3. View topology
4. View neighbors table
5. Exit
Choose an option: 2
Routing table of Node A: { B: 'B', I: 'I', F: 'I', C: 'C', D: 'I', H: 'I', E: 'I', G: 'I' }
--- Menu for Node A ---
1. Send a message
2. View routing table
3. View topology
4. View neighbors table
5. Exit
Choose an option: 1
Enter destination node ID: F
Enter the message: hello
Message sent from Node A to Node I: "hello"
Message sent from Node I to Node D: "hello"
Message sent from Node D to Node F: "hello"
Message sent via route: A -> I -> D -> F
--- Menu for Node A ---
1. Send a message
2. View routing table
3. View topology
4. View neighbors table
5. Exit
```



Calcula la distancia más corta del nodo al nodo destinatario y lo manda a través de esa ruta. Como se ve en la routing table del nodo A, para ir al nodo F, el siguiente nodo que tiene que tomar es el nodo I. Lo que se puede ver cuando se mandó el mensaje que el mensaje paso desde A -> I -> D -> F

Discusión

El algoritmo Flooding, aunque simple en concepto e implementación en comparación con otros algoritmos, presenta tanto beneficios como desventajas. Una de las principales ventajas es su idoneidad para redes con topologías cambiantes dinámicamente y una gestión de rutas de baja complejidad, ya que no requiere ningún mantenimiento de tablas de enrutamiento. Sin embargo, esta simplicidad conlleva la transmisión indiscriminada de mensajes a todos los nodos, lo que puede generar congestión y, en algunos casos, la duplicación de datos.

Dijsktra pertenece a la categoría de algoritmos avanzados, son mucho más eficientes en la búsqueda de rutas óptimas para el envío de mensajes, evitando el desperdicio de recursos. Sin embargo, estos algoritmos requieren cálculos más complejos y el mantenimiento de tablas de enrutamiento actualizadas, lo que los hace menos adecuados para topologías de red altamente dinámicas.

El algoritmo Flooding, aunque simple en concepto y en implementación en comparación a los demás algoritmos, presenta tanto beneficios como desventajas. Una de las grandes ventajas que tiene es el ser adecuado para redes con topologías cambiantes dinámicamente y con una escasa complejidad en la gestión de rutas, ya que no requiere de ningún mantenimiento de tablas de enrutamiento. Sin embargo, esta simplicidad resulta en la transmisión indiscriminada de los mensajes a todos los nodos, lo que llega a generar congestión y en

algunos casos incluso duplicación de datos. Por otro lado, los otros algoritmos vistos en esta práctica son mucho más eficientes en la búsqueda de una ruta que haga el envío de mensajes estratégicamente para no malgastar recursos innecesarios, sin embargo debido a los cálculos que estos requieren con las tablas de enrutamiento y demás entradas de datos necesarias para su funcionamiento correcto estos necesitan de un mayor tiempo de mantenimiento, algo que haría que no fuera factible implementar esta de ser una topología altamente dinámica.

En el caso del enrutamiento por distance vector, combina aspectos tanto de Flooding como de Dijkstra. Al igual que Flooding, opera en base a información local, ya que los routers solo conocen la distancia a sus vecinos directos y no tienen una visión completa de la red. Sin embargo, a diferencia de Flooding, utiliza tablas de enrutamiento y se esfuerza por encontrar una ruta óptima, similar a Dijkstra.

La ventaja principal del enrutamiento por distance vector es que es relativamente simple y se autoajusta: a medida que las condiciones de la red cambian, los routers intercambian información y actualizan sus tablas de forma autónoma. Esto lo hace adecuado para redes de tamaño medio con topologías que no cambian demasiado rápidamente.

Comentario grupal

Tras analizar y discutir los diferentes algoritmos de enrutamiento, podemos concluir que no existe una solución única que sea la mejor en todas las situaciones. Cada algoritmo presenta sus propias ventajas y desafíos, y es crucial entender las características y necesidades de nuestra red antes de seleccionar un método de enrutamiento.

El algoritmo Flooding, con su simplicidad, ofrece una solución efectiva para redes en constante cambio, pero a expensas de la eficiencia y el posible desperdicio de recursos. Por otro lado, Dijkstra, siendo uno de los algoritmos más robustos, garantiza rutas óptimas pero exige una mayor complejidad en su gestión y no es ideal para topologías de red cambiantes.

El enrutamiento por vector de distancia, siendo un punto intermedio, nos enseña que a veces los compromisos son necesarios. Puede no ser perfecto, pero su capacidad de autoajuste y relativa simplicidad lo hacen adecuado para muchas redes de tamaño medio.

Conclusiones

- El algoritmo de Flooding es efectivo cuando se tiene una topología altamente dinámica
- Dijkstra ofrece un enrutamiento más eficiente al calcular rutas óptimas y evitar el desperdicio de recursos.
- Similarmente como el Dijkstra, el distance vector ofrece una forma de calcular rutas óptimas, pero también aporta una forma donde se pueda ir actualizando y autoajustando.

Referencias

Sahil, J. (2023) Fixed and Flooding routing algorithms. Extraído de Geeks For Geeks
Yale University (2014) Flooding. Extraido de cs.yale.edu