

Universidad del Valle de Guatemala
Facultad de Ingeniería
Departamento de Ciencias de la Computación
Redes CC3067
Ing. Jorge Yass

Laboratorio # 3 (Parte 1)

Algoritmos de Enrutamiento



Christopher García 20541
José Rodrigo Barrera 20807
Maria Isabel Solano 20504
Semestre II
Septiembre 2023

Descripción de la práctica

Los algoritmos de enrutamiento son esenciales para la transmisión eficiente de datos en redes, como Internet. Su función principal es determinar la mejor ruta para enviar mensajes entre nodos, como routers. Para lograrlo, estos algoritmos utilizan tablas de enrutamiento que deben actualizarse en tiempo real para adaptarse a cambios en la red. En esta práctica, se simula el proceso de enrutamiento, centrándose en algoritmos "offline", donde la configuración de rutas se realiza manualmente o mediante codificación, ayudándonos a comprender cómo funcionan estos algoritmos.

Algoritmos utilizados

- *Flooding*

- En este algoritmo cuando un nodo (como un router) recibe un mensaje que necesita ser reenviado a su destino, este envía una copia del mensaje a todos sus vecinos sin hacer ninguna comprobación previa. Cada nodo que recibe el mensaje hace lo mismo, reenviándolo a todos sus vecinos, excepto al nodo del cual lo recibió. Este proceso se repite hasta que el mensaje llega a su destino. En el código trabajado, inicialmente, se define una clase llamada Flooding, que permite agregar nodos a una lista y realizar el proceso de "flooding". El proceso de "flooding" implica enviar un mensaje desde un nodo de origen a todos los nodos en la red, reenviándolo a sus vecinos. El mensaje se crea con un encabezado que contiene información como el nodo de origen y destino, así como un contador de saltos. La terminal permite el envío y la recepción de mensajes.

- *Dijkstra*

- En este algoritmo funciona calculando el camino más corto con respecto a un nodo inicial y uno final, por lo que, aplicándolo para el envío de mensajes utilizando el formato JSON no es la excepción, en donde, para ello se realizó un programa que calcula el camino más corto de un grado entre dos nodos, uno inicial y uno final, en donde, dichos nodos al momento de querer enviar el mensaje deben de ser indicados, y con ello, poder generar el cálculo del mismo camino y así generar el formato JSON que será utilizado para enviarlo al siguiente nodo vecino del nodo inicial que forma parte del camino más corto que se calculó para enviar dicho mensaje. En donde, mediante el receptor se detecta si se debe de avanzar al siguiente nodo tomando en consideración el tipo de mensaje que se está enviando o de lo contrario, evaluar si dicho mensaje ya debe de ser mostrado en el nodo destino.

Y esto con el uso del tipo "info" el cual fue utilizado para que, cuando se procese dicho mensaje, este pueda detectar cuándo es que se debe de seguir avanzando en los nodos vecinos al nodo en el que se encuentra uno del camino

más corto que se está siguiendo para enviar dicho mensaje, en donde, si se llega al nodo destino el tipo de mensaje cambia a “message” y este procede a revelar el payload que fue enviado con anterioridad.

- ***Vector Routing***

- Al inicio, solamente se crean las terminales por cada uno de los nodos a trabajar en la topología. En cada una de ellas se pregunta, primero, el nombre del nodo que representa y luego, uno por uno, se pregunta el nombre de sus vecinos y el peso que tiene la comunicación. Con ello se inicializa su tabla de enrutamiento, colocando el nombre, peso y el *hop* que debe hacer el nodo. Luego de realizar el *setup*, se ofrecen 5 opciones al usuario.
 - 1) Enviar un mensaje. Esta pide el texto que se quiere enviar y a qué nodo (de los que tiene en su tabla de enrutamiento) y genera el JSON necesario e indica a qué nodo debe enviarse.
 - 2) Enviar info a los vecinos. Genera los JSONs necesarios por cada uno de los vecinos directos del nodo, para compartir su tabla de enrutamiento.
 - 3) Recibir info/mensaje. Lee un archivo JSON en donde que pega los mensajes que se quieren enviar. Determina si este es de tipo “message” o de tipo “info”.
 - Si es “message”, determina si el mensaje es para el nodo que lo recibió. Si sí lo es, imprime el mensaje y quién lo envía.
 - Si es “info”, obtiene la tabla de enrutamiento que se le ha enviado y actualiza los valores de la suya si encuentra rutas más cortas.
 - 4) Mostrar tabla de enrutamiento. Permite observar la tabla de enrutamiento con la que se está trabajando actualmente.
 - 5) Salir. Se deja de usar el algoritmo Distance Vector Routing

Resultados y Discusión de Resultados

Flooding

```
python x python x python x
4) Salir
Número de la opción: 2
Nombre del nodo actual: D
Desea ingresar vecinos? (y/n): Y
Nombre del nodo vecino: C
Desea ingresar vecinos? (y/n): N
Entendido, continuando con el programa...

Datos ingresados
Nodo: D | Vecinos: [Nodo: C]

1) Enviar mensaje
2) Recibir mensaje
3) Salir
Ingresar opción: 2
Ingresar el mensaje recibido (Doble Enter para confirmar)
{
  "type": "message",
  "headers": {
    "from": "A",
    "to": "D",
    "hop_count": 2,
    "receivers": [
      "A",
      "B",
      "C"
    ]
  },
  "payload": "HOLA AAA"
}

(D) Mensaje entrante de: A
HOLA AAA

1) Enviar mensaje
2) Recibir mensaje
3) Salir
Ingresar opción: []

1) Enviar mensaje
2) Recibir mensaje
3) Salir
Ingresar opción: 2
Ingresar el mensaje recibido (Doble Enter para confirmar)
{
  "type": "message",
  "headers": {
    "from": "A",
    "to": "D",
    "hop_count": 1,
    "receivers": [
      "A",
      "B",
      "C"
    ]
  },
  "payload": "HOLA AAA"
}

Reenvía este paquete a: D
{
  "type": "message",
  "headers": {
    "from": "A",
    "to": "D",
    "hop_count": 1,
    "receivers": [
      "A",
      "B",
      "C"
    ]
  },
  "payload": "HOLA AAA"
}

1) Enviar mensaje
2) Recibir mensaje
3) Salir
Ingresar opción: []

3) Salir
Ingresar opción: 1
Nombre de nodo destino: D
Ingresar el mensaje que deseas enviar: HOLA AAA
Ingresar el hop_count: 4

Reenvía este paquete a: B
{
  "type": "message",
  "headers": {
    "from": "A",
    "to": "D",
    "hop_count": 3,
    "receivers": [
      "A",
      "B"
    ]
  },
  "payload": "HOLA AAA"
}

Reenvía este paquete a: C
{
  "type": "message",
  "headers": {
    "from": "A",
    "to": "D",
    "hop_count": 2,
    "receivers": [
      "A",
      "B",
      "C"
    ]
  },
  "payload": "HOLA AAA"
}

1) Enviar mensaje
2) Recibir mensaje
3) Salir
Ingresar opción: []
```

Imagen 1. Ejecución de Flooding en 3 terminales diferentes

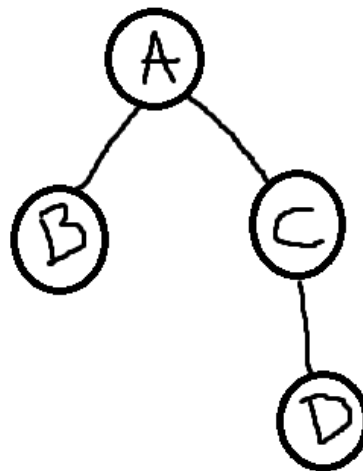
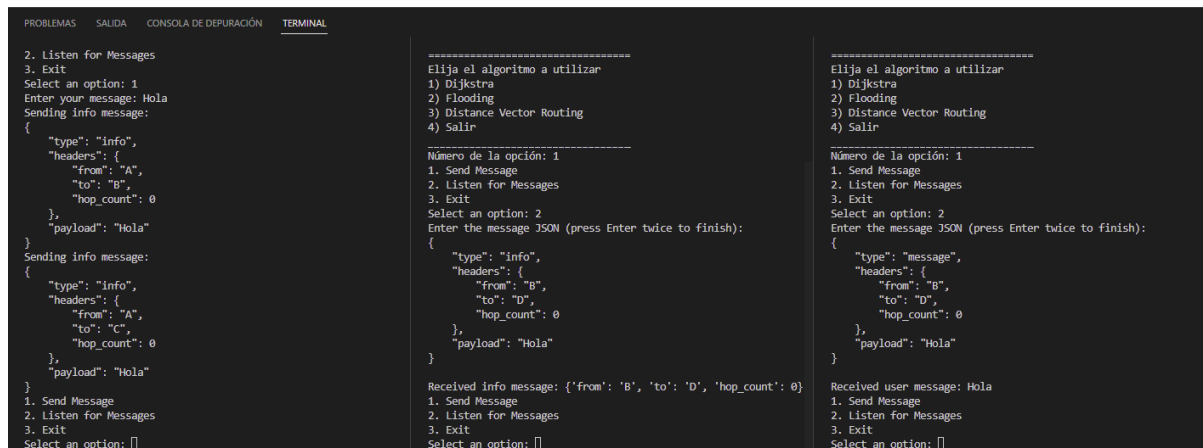


Figura 1. Nodo de referencia imagen#1-flooding

Explicación:

- En la terminal de la izquierda se encuentra el nodo A, con vecinos B y C. Por eso pide enviar a estos nodos el mensaje
- En la terminal de en medio se encuentra el nodo C, con vecinos A y D. Como ya pasó por A, bueno, de hecho viene de A, solo pide que se envíe a D el mensaje
- En la terminal de la derecha se encuentra el nodo D, con vecino C. Como este nodo es el receptor, imprime el mensaje final

Dijkstra



```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

2. Listen for Messages
3. Exit
Select an option: 1
Enter your message: Hola
Sending info message:
{
  "type": "Info",
  "headers": {
    "from": "A",
    "to": "B",
    "hop_count": 0
  },
  "payload": "Hola"
}
Sending info message:
{
  "type": "Info",
  "headers": {
    "from": "A",
    "to": "C",
    "hop_count": 0
  },
  "payload": "Hola"
}
1. Send Message
2. Listen for Messages
3. Exit
Select an option: []

=====
Elija el algoritmo a utilizar
1) Dijkstra
2) Flooding
3) Distance Vector Routing
4) Salir

Número de la opción: 1
1. Send Message
2. Listen for Messages
3. Exit
Select an option: 2
Enter the message JSON (press Enter twice to finish):
{
  "type": "info",
  "headers": {
    "from": "B",
    "to": "D",
    "hop_count": 0
  },
  "payload": "Hola"
}
Received info message: {'from': 'B', 'to': 'D', 'hop_count': 0}
1. Send Message
2. Listen for Messages
3. Exit
Select an option: []

=====
Elija el algoritmo a utilizar
1) Dijkstra
2) Flooding
3) Distance Vector Routing
4) Salir

Número de la opción: 1
1. Send Message
2. Listen for Messages
3. Exit
Select an option: 2
Enter the message JSON (press Enter twice to finish):
{
  "type": "message",
  "headers": {
    "from": "B",
    "to": "D",
    "hop_count": 0
  },
  "payload": "Hola"
}
Received user message: Hola
1. Send Message
2. Listen for Messages
3. Exit
Select an option: []
```

Imagen 2. Ejecución de Dijkstra en tres terminales diferentes

Explicación:

Básicamente el algoritmo de Dijkstra funciona de la siguiente manera, en la primera terminal dado un grafo, a esta se le indica el nodo inicial y final al que se le quiere enviar el mensaje, por lo que, en la primera terminal se tendrá al nodo A con sus vecinos B y C, entonces, esto lo que genera es un JSON el cuál deberá de moverse al siguiente nodo que se ubica con su vecino en la segunda terminal, en este caso, como es un algoritmo de Dijkstra, nos movemos en el camino más corto previamente calculado.

Entonces, colocamos el JSON en la segunda terminal la cual contiene a B y todos sus vecinos, y esto nos arroja el camino que debe de seguir, el cual sería hacia D, el cual sería el camino para llegar al nodo destino, acá es donde cambia, porque esto al detectar que ya se quiere llegar al nodo destino con el “to”, esto cambia el tipo de mensaje a “message” lo que nos permite que, al momento de llegar al último nodo el cual se encuentra en la última terminal, esto solo muestre el payload recibido del nodo inicial que se quiso enviar.

Distance Vector Routing

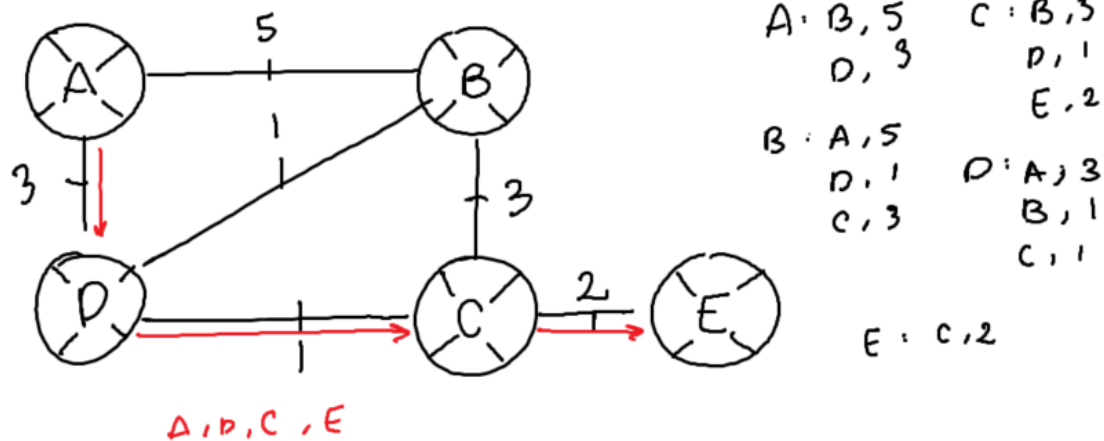


Figura 2. Ejemplo de grafo para distance vector routing

Utilizando la topología que se muestra en la figura anterior. Al lado derecho se pueden notar los valores iniciales de los vecinos directos de cada uno de los nodos que componen el grafo.

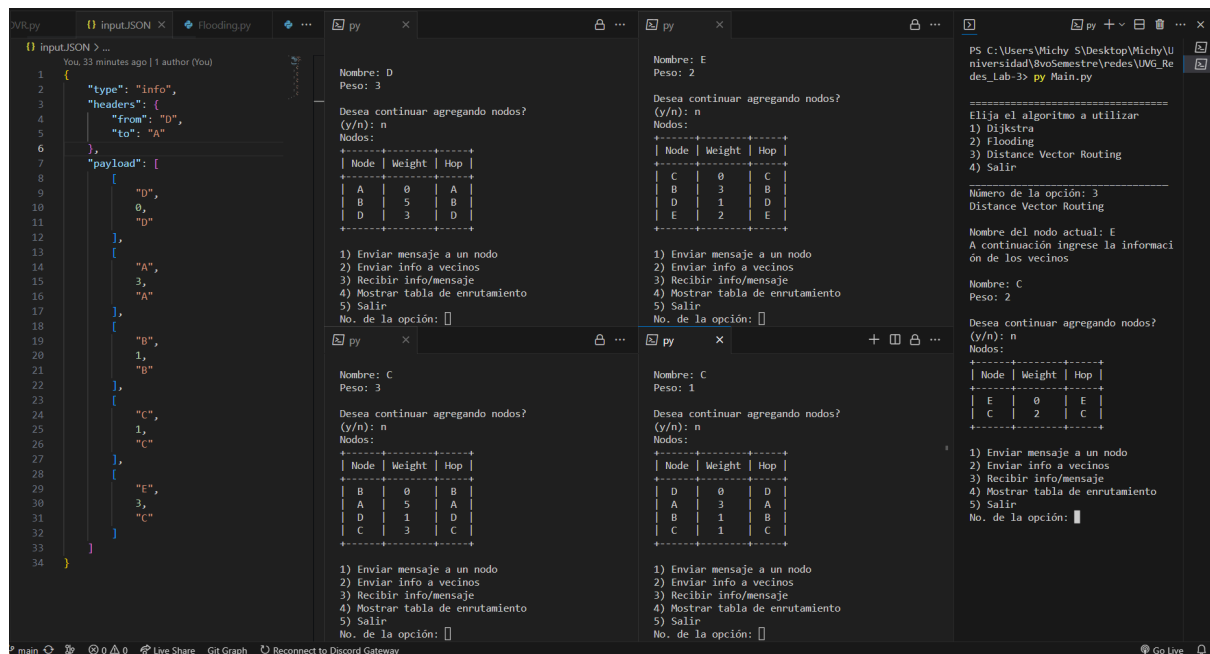


Imagen 3. Tablas de enrutamiento iniciales

Por ello se abrieron 5 consolas distintas. Cada una representando un nodo (A a E de arriba a abajo, e izquierda a derecha). En la imagen se pueden observar los valores iniciales de cada tabla de enrutamiento.

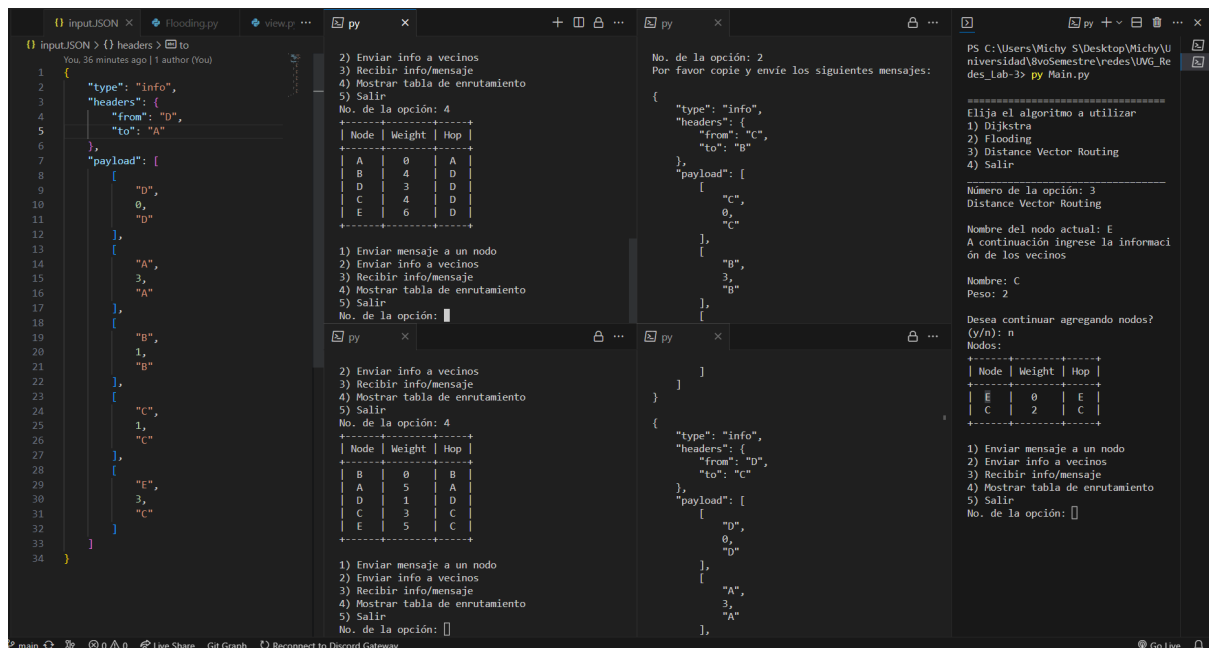


Imagen 4. Tablas de enrutamiento en camino de ser actualizadas

En la segunda imagen podemos observar tablas de enrutamiento actualizadas luego del paso de varios paquetes. En la primera consola se puede notar que ya tiene la ruta más corta hacia E, y que la ruta hacia B ya no pasa va directamente a B sino que pasa por D primero para tomar un camino menos pesado.

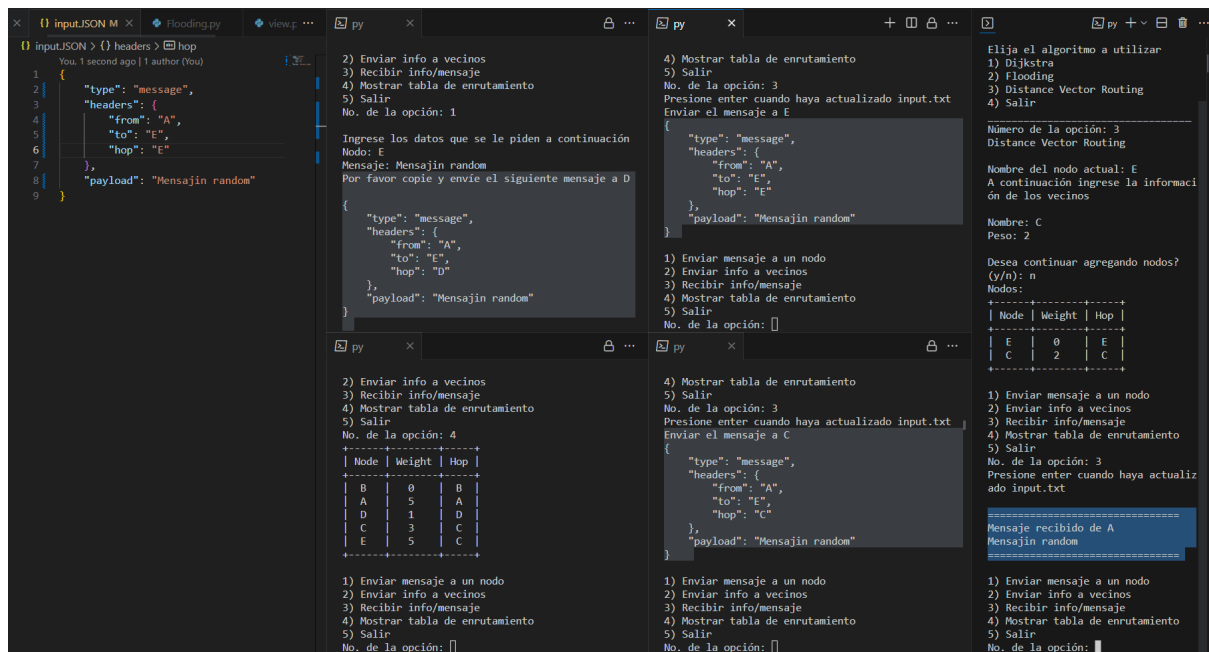


Imagen 5. Envío de paquetes de mensaje entre nodos

En la imagen anterior se puede observar el paso de mensajes de mensajes, desde A, pasando por D, C y por último E, quien reconoce que el mensaje es para él mismo, y lo imprime directamente, iniciando de quién viene originalmente y el texto que se encontraba en el payload.

A diferencia de otros algoritmos, el Distance Vector Routing no tiene un objetivo final o a un estado al que siempre llegue. Este simplemente va actualizando su información conforme el tiempo pasa y la topología cambia. Por lo que este algoritmo en teoría nunca termina de ejecutarse. Aunque en el caso en el que se estaba trabajando, es una topología estática, por lo que solo se necesita ejecutar el algoritmo hasta que converjan todas las tablas y ya no ocurran cambios. Tal como se puede observar en la siguiente imagen en donde ya están todas las tablas de enrutamiento actualizadas con las rutas más cortas.

```

1 {
2   "type": "info",
3   "headers": {
4     "from": "D",
5     "to": "B"
6   },
7   "payload": [
8     "D",
9     0,
10    "D"
11  ],
12  [
13    "A",
14    3,
15    "A"
16  ],
17  [
18    "B",
19    1,
20    "B"
21  ],
22  [
23    "C",
24    1,
25    "C"
26  ],
27  [
28    "E",
29    3,
30    "E"
31  ],
32  [
33    "C",
34    1,
35    "C"
36  ]
37 }

```

2) Enviar info a vecinos
3) Recibir info/mensaje
4) Mostrar tabla de enrutamiento
5) Salir
No. de la opción: 4

Node	Weight	Hop
A	0	A
B	4	D
D	3	D
C	4	D
E	6	D

1) Enviar mensaje a un nodo
2) Enviar info a vecinos
3) Recibir info/mensaje
4) Mostrar tabla de enrutamiento
5) Salir
No. de la opción: 4

Node	Weight	Hop
C	0	C
B	2	D
D	1	D
E	2	E
A	4	D

1) Enviar mensaje a un nodo
2) Enviar info a vecinos
3) Recibir info/mensaje
4) Mostrar tabla de enrutamiento
5) Salir
No. de la opción: 4

Node	Weight	Hop
E	0	E
C	2	C
B	5	C
D	3	C

1) Enviar mensaje a un nodo
2) Enviar info a vecinos
3) Recibir info/mensaje
4) Mostrar tabla de enrutamiento
5) Salir
No. de la opción: 3
Presione enter cuando haya actualizado input.txt

1) Enviar mensaje a un nodo
2) Enviar info a vecinos
3) Recibir info/mensaje
4) Mostrar tabla de enrutamiento
5) Salir
No. de la opción: 4

Node	Weight	Hop
E	0	E
C	2	C
B	4	C
D	3	C
A	6	C

1) Enviar mensaje a un nodo
2) Enviar info a vecinos
3) Recibir info/mensaje
4) Mostrar tabla de enrutamiento
5) Salir
No. de la opción: 4

Node	Weight	Hop
D	0	D
A	3	A
B	1	B
C	1	C
E	3	C

1) Enviar mensaje a un nodo
2) Enviar info a vecinos
3) Recibir info/mensaje
4) Mostrar tabla de enrutamiento
5) Salir
No. de la opción: 4

Node	Weight	Hop
B	0	B
A	4	D
D	1	D
C	2	D
E	4	D

1) Enviar mensaje a un nodo
2) Enviar info a vecinos
3) Recibir info/mensaje
4) Mostrar tabla de enrutamiento
5) Salir
No. de la opción: 4

Imagen 6. Tablas de enrutamiento actualizadas.

El algoritmo de Flooding es simple pero ineficiente, ya que envía paquetes de datos a todos los nodos en la red sin considerar la topología, lo que puede causar congestión y un alto consumo de ancho de banda. El algoritmo de Dijkstra, por otro lado, es más eficiente en términos de uso de ancho de banda y garantiza el camino más corto, pero requiere conocimiento global de la topología de la red y es costoso en términos de cálculos, lo que lo hace menos adecuado para redes grandes. Y el algoritmo de Distance Vector es útil en entornos simples y redes pequeñas, pero en redes más grandes y complejas, se suelen preferir algoritmos de enrutamiento más avanzados.

Conclusiones

- Los algoritmos de enrutamiento a menudo enfrentan un compromiso entre eficiencia y precisión. Los algoritmos más precisos, como Dijkstra, garantizan rutas óptimas, pero a costa de un mayor uso de recursos de cómputo y ancho de banda. Por otro lado, los algoritmos más eficientes en recursos, como Distance Vector, pueden no proporcionar la máxima precisión en redes complejas, pero sí es una herramienta poderosa para topologías cambiantes.

- La aplicación de un algoritmo u otro dependerá totalmente de la red donde se desean implementar ya que estos algoritmos necesitan de ciertos elementos y su desempeño será mejor o peor en ambientes dados.
- Con el trabajo realizado durante la práctica y la simulación desarrollada se puede tener una mejor idea de cómo se aplican los algoritmos de enrutamiento para que un router pueda encontrar a sus routers vecinos y transmitir la información con la ruta más corta.

Comentarios

- La práctica fue muy interesante de realizar. A pesar de los problemas de comprensión de las instrucciones que se tuvo junto al resto de compañeros de la clase sobre la práctica, luego de comprender bien cuál era su objetivo y cómo exactamente era requerida hacer la simulación ya esta no tuvo mayores inconvenientes ni complicaciones.
- La práctica nos pareció bastante interesante, a pesar de cada uno de los contratiempos, como también, situaciones que la propia práctica ponía, esto nos permitió aprender el funcionamiento correcto de cada uno de los algoritmos al momento de enviar un mensaje.

Link del repositorio

- https://github.com/ChristopherG19/UVG_Redес_Lab-3.git

Referencias

- Routing: Distance Vector Algorithm. (n.d.). Retrieved September 3, 2023, from https://www.cs.uni.edu/~diesburg/courses/cs3470_fa14/sessions/s22/s22.pdf
- Pedraza, L. F., López, D., & Salcedo, O. (2023). Enrutamiento basado en el algoritmo de Dijkstra para una red de radio cognitiva. *Tecnura*, 15(30), 94–100. http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S0123-921X2011000300010
- ¿Qué es el enrutamiento? - Explicación del enrutamiento de redes - AWS. (2023). Amazon Web Services, Inc. <https://aws.amazon.com/es/what-is/routing/>