Universidad del Valle de Guatemala
Redes I Sección 10
Luis Estuardo Delgado Ordoñez 17187
Josué David López Florian 17081
Julio Francisco Barahona Martinez 141206

Laboratorio 3/4
Algoritmos de Enrutamiento en Redes.

Descripción

La práctica se dividió en dos partes, la primera fue desarrollar los siguientes algoritmos::

- Flooding
  - Consiste en enviar un mensaje a todos los vecinos de cada nodo. La idea de esto es que eventualmente un nodo conectará con el destino y recibirá el mensaje. Si no se llega a recibir y no ha ocurrido otro error implica que el nodo no existe.
- Distance vector routing
  - Conocido como el ruteo Bellman Ford. Cada nodo genera una tabla de distancias que contiene la distancia entre él y los nodos de destino. Las distancias se calculan utilizando información de los vectores de distancia de los nodos.
- Link state routing
  - También se le dice Dijkstra. sus pasos son recursivos y son así:
    - Marcar nodos como no visitados y hacer un listado de todos.
    - Asignar y leer los pesos de las conexiones de cada nodo
    - Para el nodo actual, considere todos sus vecinos no visitados y calcule sus distancias tentativas a través del nodo actual. Compare la distancia tentativa recién calculada con el valor asignado actual y asigne el más pequeño.
    - Cuando se haya terminado de calcular todos los vecinos no visitados del nodo actual, marcar el nodo actual como visitado y eliminar del conjunto no visitado. Un nodo visitado nunca se vuelve a comprobar.
    - Si el nodo de destino se ha marcado como visitado o si la distancia tentativa más pequeña entre los nodos del conjunto no visitado es infinita el algoritmo llega a su fin.

La segunda parte consistió en usar un listado de pesos para cada nodo. Con este grafo encontré la mejor ruta con cada algoritmo implementado. La constancia de el proceso se verá en los logs generado por cada nodo fuente que despliegue el recorrido al nodo destino de la siguiente manera:

- Nodo fuente [ID / dirección]
- Nodo destino [ID / dirección]
- Saltos recorridos [numérico]
- Distancia [numérico]
- Listado de nodos usados [texto]
- Mensaje [texto]

Resultados

El reto que se presentó fue el no usar protocolos de enrutamiento. Este era el propósito del laboratorio. Debido a que no estaban estas librerías, el paquete de datos que se armó fue pensado en que tenía que manejar cada posible resultado de cada algoritmo. La mayor variación fue con flooding ya que este no tenía camino o ruta, más bien tenía que mandar a todos las conexiones de todo nodo que recibiera el mensaje y saber detenerse hasta que el nodo objetivo recibiera el mensaje.

Para Bellman Ford el reto fue hacer reconocer al algoritmo que ruta tomar ya que se presentó que existía un nodo que tenía el mismo predecesor pero no era el que conducía a el nodo objetivo. Por esta razón se elaboró una lista que se iba construyendo de final a principio y cada vez que encontraba error elimina de la lista de conexiones esa ruta para así regresar y volver a buscar un predecesor que tuviera la ruta nueva.

En la elaboración y construcción de los algoritmos de flooding, Distance Vector Routing y link state routing se elaboraron distintas pruebas para ver su desempeño y si daban con las rutas establecidas por los algoritmos y logren llegar a su destino objetivo. El primero en probar los resultados de las pruebas fue flooding quien nos dio los distintos resultados que podemos ver en las siguientes screenshots:

```
C:\Users\Blinkout\Desktop\Redes\lab34\Git>python controller.py
['h', 'f', 'd', 'c', 'a']
Ingrese la dirección para conectarse: 3.22.99.8
Ingrese el numero de puerto: 55555
choose a nickname: Cont
Elige el numero de uno de los siguientes algoritmos:
1. Flooding
2. Distance vector routing
3. Link state routing
1
Escriba el mensaje que quiera enviar
Hola flood
elija el nodo desde donde se quiere enviar dict_keys(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i'])
a
elija el nodo destino del mensaje dict_keys(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i'])
e
Ingrese la cantidad máxima de saltos.4
```

figura 1: Controlador haciendo flooding desde “a” a “e” con 4 saltos

```
C:\Users\Blinkout\Desktop\Redes\lab34\Git>python clientH.py
Ingrese la dirección para conectarse: 3.22.99.8
Ingrese el numero de puerto: 55555
```

figura 2: No llega a H porque no le alcanzan los saltos

```
C:\Users\Blinkout\Desktop\Redes\lab34\Git>python clientA.py
Ingrese la dirección para conectarse: 3.22.99.8
Ingrese el numero de puerto: 55555
Nodo origen: a
Nodo destino: e
Mensaje: Hola flood
Nodos pasados:
['a']
Saltos: 0
Distancia recorrida: 0
```

figura 3: Observamos cómo no regresa a A

```
C:\Users\Blinkout\Desktop\Redes\lab34\Git>python clientC.py
Ingrese la dirección para conectarse: 3.22.99.8
Ingrese el numero de puerto: 55555
Nodo origen: a
Nodo destino: e
Mensaje: Hola flood
Nodos pasados:
['a', 'c']
Saltos: 1
Distancia recorrida: 1
Nodo origen: a
Nodo destino: e
Mensaje: Hola flood
Nodos pasados:
['a', 'b', 'f', 'd', 'c']
Saltos: 4
Distancia recorrida: 20
```

figura 4: Llega hasta C 2 veces por cantidad de saltos

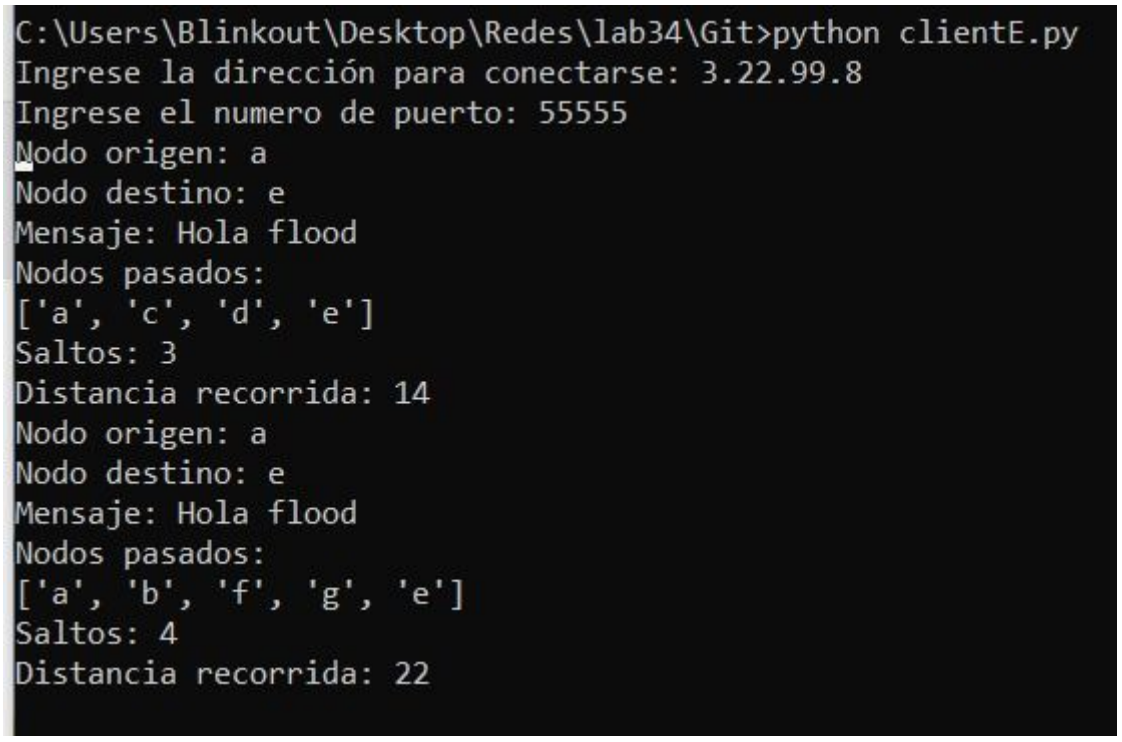

figura 5: Llega hasta E por 2 caminos diferentes.

Estos fueron los resultados visto al hacer uso del algoritmo de Flooding. Se hizo para todos los nodos, pero estos son solo variaciones de lo mismo obtenido en los nodos A y C. Luego procedimos a ver los resultados obtenidos por el siguiente algoritmo llamado Distance vector routing, el cual dio los siguiente resultados:

figura 6: DVR iniciando desde controlador con B hasta E.

```
C:\Users\Blinkout\Desktop\Redes\lab34\Git>python clientB.py
Ingrese la dirección para conectarse: 3.22.99.8
Ingrese el numero de puerto: 55555
Nodo origen: b
Nodo destino: e
Mensaje: Hola desde DVR
Nodos pasados:
['b']
Saltos: 0
Distancia recorrida: 0
```

figura 7: Pantalla de B siendo origen

```
C:\Users\Blinkout\Desktop\Redes\lab34\Git>python clientF.py
Ingrese la dirección para conectarse: 3.22.99.8
Ingrese el numero de puerto: 55555
Nodo origen: b
Nodo destino: e
Mensaje: Hola desde DVR
Nodos pasados:
['b', 'f']
Saltos: 1
Distancia recorrida: 8
```

figura 8: Pantalla de F siendo el segundo nodo a recorrer el mensaje

```
C:\Users\Blinkout\Desktop\Redes\lab34\Git>python clientG.py
Ingrese la dirección para conectarse: 3.22.99.8
Ingrese el numero de puerto: 55555
Nodo origen: b
Nodo destino: e
Mensaje: Hola desde DVR
Nodos pasados:
['b', 'f', 'g']
Saltos: 2
Distancia recorrida: 12
```

figura 9: Pantalla de G siendo el tercer nodo a recorrer el mensaje

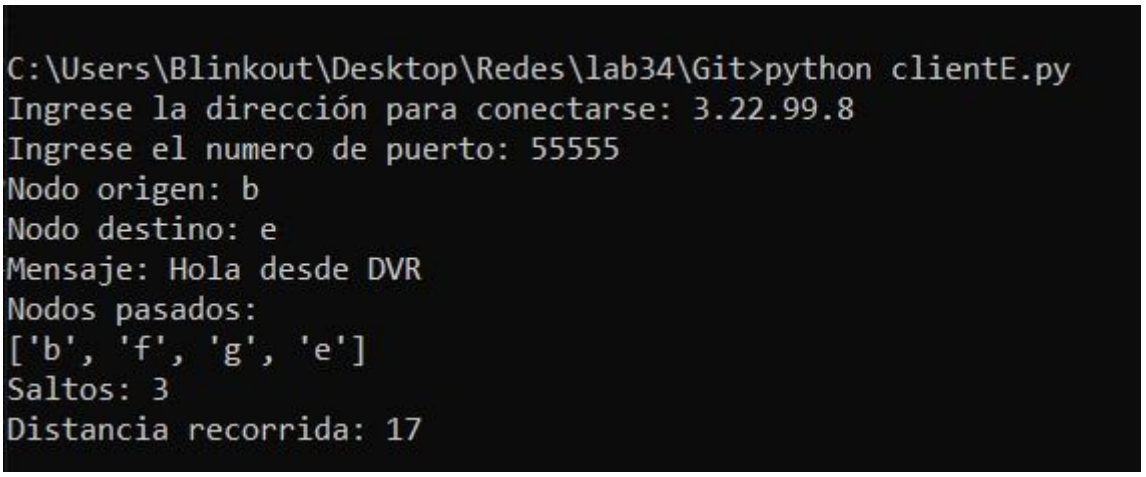

figura 10: Pantalla de E siendo el nodo destino a donde llega el mensaje.

Esto fue lo obtenido al final con este algoritmo con las ruta que el mismo algoritmo trazo para el envío del mensaje. Ahora toca ver lo obtenido por el último nodo de Link state routing. Esto fue lo obtenido con el último algoritmo:

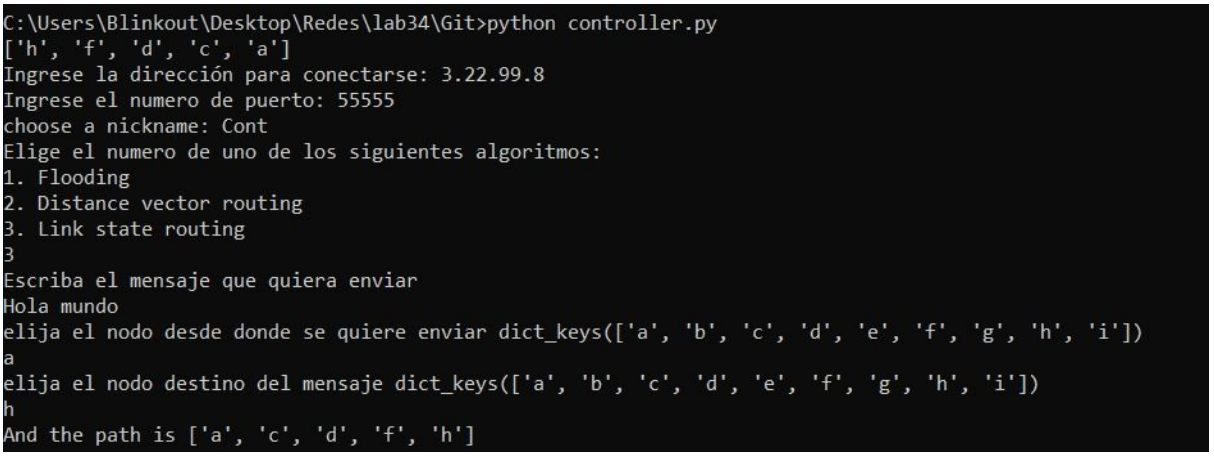

figura 11: Controlador con Dijkstra queriendo enviar mensaje de A a H

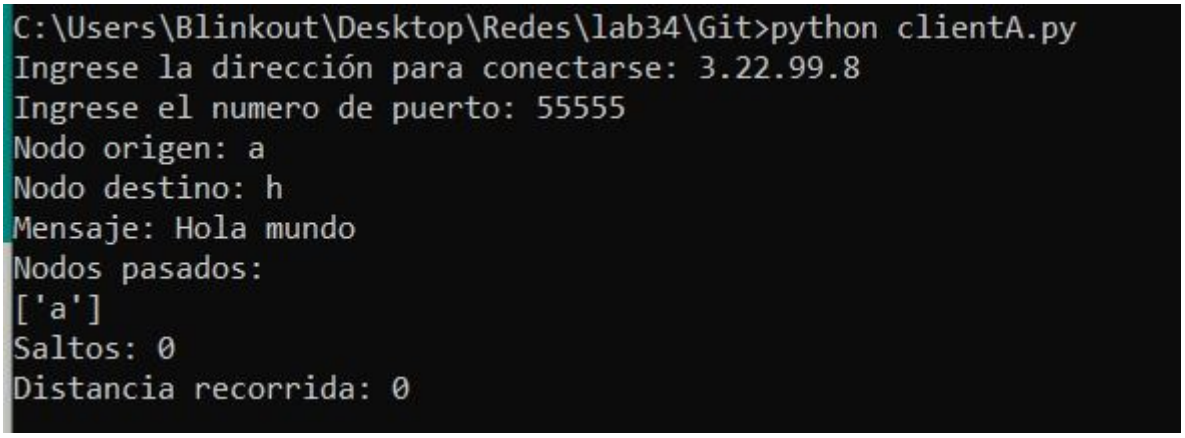

figura 11: Cliente A siendo el origen

```
C:\Users\Blinkout\Desktop\Redes\lab34\Git>python clientC.py
Ingrese la dirección para conectarse: 3.22.99.8
Ingrese el numero de puerto: 55555
Nodo origen: a
Nodo destino: h
Mensaje: Hola mundo
Nodos pasados:
['a', 'c']
Saltos: 1
Distancia recorrida: 1
```

figura 11: Client C siendo el segundo

```
C:\Users\Blinkout\Desktop\Redes\lab34\Git>python clientD.py
Ingrese la dirección para conectarse: 3.22.99.8
Ingrese el numero de puerto: 55555
Nodo origen: a
Nodo destino: h
Mensaje: Hola mundo
Nodos pasados:
['a', 'c', 'd']
Saltos: 2
Distancia recorrida: 5
```

figura 12: Client D siendo el tercero

```
C:\Users\Blinkout\Desktop\Redes\lab34\Git>python clientF.py
Ingrese la dirección para conectarse: 3.22.99.8
Ingrese el numero de puerto: 55555
Nodo origen: a
Nodo destino: h
Mensaje: Hola mundo
Nodos pasados:
['a', 'c', 'd', 'f']
Saltos: 3
Distancia recorrida: 8
```

figura 13: Client F siendo el tercero

```
C:\Users\Blinkout\Desktop\Redes\lab34\Git>python clientH.py
Ingrese la dirección para conectarse: 3.22.99.8
Ingrese el numero de puerto: 55555
Nodo origen: a
Nodo destino: h
Mensaje: Hola mundo
Nodos pasados:
['a', 'c', 'd', 'f', 'h']
Saltos: 4
Distancia recorrida: 11
```

figura 13: Client H siendo el destino

Discusión

El algoritmo utilizado para realizar el flooding realizó la transmisión de cliente a cliente. Cada cliente realizaba un for loop el cual enviaba a cada uno de sus vecinos una copia nueva del mensaje. Un error que se mostró durante las pruebas y ejecución de este algoritmo fue que sí el nodo enviaba datos a más de 2 vecinos el último vecino en la lista no recibía el mensaje. Posibles razones para esto son, una mala implementación del algoritmo, el servidor no recibía todos los mensajes debido a un throttle del algoritmo, el servidor no podía manejar más de 2 mensajes por cliente a la vez. Pese a las limitaciones el algoritmo se vió efectivo a una menor escala comparada con la presentación.

Originalmente el método de DVR (distance-vector routing) era el usado para el manejo de ruteo. [1] Por cuestiones de escalabilidad y el hecho que tenía que calcular hasta el final de cada posible punto creció de manera descomunal. Cuando pasó el tiempo se implementaron nuevas técnicas que buscan las áreas más conectadas y pre establecen rutas para el tráfico potencial. Esto deja a DVR con menor potencia y accesibilidad en la mayoría de casos que presentan tráfico masivo.

EL algoritmo LSR (Link-State Routing) fue el algoritmo con la implementación más simple. Fue el algoritmo con la mayor cantidad de documentación alrededor de sí mismo. En cuanto a funcionalidad el algoritmo está implementado correctamente, sin embargo en el testeo y la implementación en la red, se vió que al utilizarlo dentro del controlador para demostraciones solo se podía utilizar una vez antes de tener que reiniciar el sistema. Esto puede ser que la implementación utilizaba variables que en alguna parte del código se modificarán sin nuestro conocimiento. Esto entorpece los esfuerzos de testing y demuestra más el algoritmo sí funcionaba por la ejecución inicial.

Comentarios Personales

- En lo personal siento que como el grafo era pequeño no se tenía mayor problema. Si hacemos un graph con conexiones de más de 10000 nodos creo que flooding sería el primero en fallar por la cantidad de información que enviará.
- La cantidad de información al ser pequeña daba bastante libertad en cuanto al manejo de los datos. Definitivamente dvr fue el más difícil de implementar para nosotros debido a como teníamos pensado la estructura del mensaje y la red.
- Creemos que el mejor algoritmo en cualquier caso de ser pequeño o grande el grafo, resulta muy versátil para el envío de información. Era el que más documentación tenía y el que menos errores dio al implementarse.

Conclusiones

- Flooding es un algoritmo útil cuando se quiere asegurar que la información llegue o que todos los nodos reciban dicha información.
- Flooding debe tener una implementación de infraestructura que se prepare para las situaciones que puede levantar al tener tantas copias de un paquete en su red.
- DVR es factible con dijkstra en escalas pequeñas ya que deja una lista de destino hechos y calculados de una vez para referencia rápida.

- Link State Routing funciona como uno de los algoritmos más óptimos y fáciles de implementar para este tipo de trabajos, además de ser más escalable que los otros.

Referencias

- Boldrini, C., Conti, M., Jacopini, J., & Passarella, A. (2007, June). Hibop: a history based routing protocol for opportunistic networks. In *2007 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks* (pp. 1-12). IEEE.
- Arturo Verbel De Leon. (2018). Algoritmos de Dijkstra y Floyd-Warshall en Python 3.5. 03/09/2020, de MI CAMINO MASTER Sitio web: http://micaminomaster.com.co/grafo-algoritmo/algoritmos-dijkstra-floyd-warshall-python-3-5/