Universidad del Valle de Guatemala

Facultad de Ingeniería

Redes

Laboratorio #3

Alejandro José Gómez Hernández

Gabriel Alejandro Vicente Lorenzo

Oscar Oswaldo Estrada Morales

Introducción:

Las redes de comunicación son un entramado complejo que se fundamenta en la correcta transmisión de datos entre múltiples nodos interconectados. La eficiencia de estas redes radica en su capacidad para identificar las rutas más óptimas para enviar y recibir paquetes de datos. Las tablas de enrutamiento, donde se almacena esta información crucial, requieren ser actualizadas constantemente para reflejar el dinamismo inherente a la estructura de Internet. En el corazón de este proceso de actualización y adaptabilidad, se encuentran los algoritmos de enrutamiento. Este informe se centra en 3 algoritmos. En primer lugar, el algoritmo de Flooding, un enfoque básico pero fundamental en la teoría de enrutamiento, y su implementación en el lenguaje de programación Python.

Antecedentes:

Como parte de los requerimientos del laboratorio se investigó respecto a los algoritmos de enrutamiento. Cabe destacar que desde los inicios de las comunicaciones en red, ha sido un desafío encontrar métodos que realmente sean eficientes en la transmisión de paquetes de datos de un nodo a otro. La clave para lograr esto radica en las tablas de enrutamiento, que guían los paquetes a través de la red (Tanenbaum & Wetherall, 2011). Conforme ha avanzado el tiempo, es importante que estas tablas se actualicen regularmente. Es por esto que los algoritmos de enrutamiento, desarrollados a lo largo de los años, se encargan de mantener estas tablas actualizadas y adaptadas a la estructura de la red (Kurose & Ross, 2020).

Algoritmo de Flooding:

El algoritmo de Flooding es uno de los algoritmos de enrutamiento más básicos y conocidos. Su funcionamiento es sencillo: cuando un nodo recibe un paquete, lo envía a todos sus nodos vecinos, a excepción de aquel nodo desde el cual recibió el paquete. Aunque es un enfoque directo, presenta desafíos, como el tráfico redundante generado en la red. Sin embargo, su simplicidad proporciona una comprensión sólida de cómo los paquetes pueden propagarse en una red (Tanenbaum & Wetherall, 2011).

Funcionamiento:

1. Inicialización: Cuando un nodo quiere enviar un paquete o mensaje a otro nodo en la red, y decide usar flooding, comienza por enviar este paquete a todos sus nodos vecinos.

2. Retransmisión: Al recibir un paquete, cada nodo vecino a su vez retransmite el paquete a todos sus vecinos, excepto al nodo del cual recibió el paquete originalmente.
3. Prevención de bucles infinitos: Sin algún mecanismo para detener la retransmisión, el paquete seguirá circulando indefinidamente en la red. Por ello, típicamente se emplean mecanismos, como un contador en el paquete que decrece con cada retransmisión y, cuando llega a cero, el paquete no se retransmite más. Otro mecanismo común es llevar un registro de los paquetes ya retransmitidos para no hacerlo de nuevo.

Ventajas:

- Robustez: A pesar de sus desafíos, el flooding garantiza que si hay al menos un camino viable entre el origen y el destino, el paquete llegará al destino.
- Descubrimiento de la Red: Es útil en situaciones donde los nodos necesitan descubrir la topología de la red o identificar otros nodos en la red.

Desventajas:

- Tráfico Redundante: Uno de los principales problemas del flooding es que crea una cantidad significativa de tráfico redundante en la red. Si no se gestionan adecuadamente, varios paquetes duplicados pueden seguir circulando, lo que congestiona la red y puede degradar el rendimiento.
- Uso de Recursos: Debido a que cada nodo tiene que procesar y reenviar cada paquete, esto puede llevar a un consumo innecesario de ancho de banda y recursos.
- Complejidad: Aunque el algoritmo en sí es simple, la gestión de sus desafíos, como el control de paquetes duplicados y la prevención de bucles infinitos, puede añadir complejidad.

Aplicaciones típicas:

Si bien el flooding puro rara vez se usa en redes modernas debido a su ineficiencia, ha servido como base para otros algoritmos y protocolos más sofisticados. Por ejemplo, en algunas versiones de protocolos de descubrimiento de servicios o en protocolos de enrutamiento basados en vectores de distancia, el flooding puede ser utilizado de forma controlada para la distribución de información.

Algoritmo de Distance Vector Routing:

El algoritmo de Distance Vector Routing es uno de los enfoques más antiguos y fundamentales utilizados en el enrutamiento de redes. Su funcionamiento se basa en que cada nodo en la red mantiene una tabla de enrutamiento que contiene información sobre las distancias y rutas hacia otros nodos en la red. A medida que los nodos intercambian información con sus vecinos, actualizan sus tablas de enrutamiento para reflejar las distancias más cortas conocidas hacia otros destinos. Aunque

ha sido reemplazado en muchas redes modernas por algoritmos más eficientes, como OSPF o BGP, el algoritmo de Distance Vector Routing sigue siendo un concepto fundamental en el campo de las redes (Smith, Murthy & García, 1997).

Funcionamiento:

1. Inicialización: Cada nodo en la red comienza con una tabla de enrutamiento que generalmente contiene información sobre sus vecinos directos y las distancias conocidas hacia esos nodos. Inicialmente, asume que está a una distancia infinita de todos los demás nodos, excepto a una distancia de cero hacia sí mismo.
2. Intercambio de información: Los nodos vecinos intercambian información de enrutamiento periódicamente. Cada nodo actualiza su tabla de enrutamiento utilizando la información recibida de sus vecinos.
3. Cálculo de distancias: Utilizando el algoritmo de Bellman-Ford, cada nodo calcula las distancias más cortas hacia todos los demás nodos en la red. Este cálculo implica tomar la distancia mínima conocida hacia un nodo y agregarle la distancia desde el nodo actual hasta el vecino que proporcionó esa información.
4. Actualización de rutas: Si un nodo descubre una ruta más corta hacia un destino específico, actualiza su tabla de enrutamiento para reflejar la nueva distancia y la dirección del próximo salto.

Ventajas:

- Simplicidad: El algoritmo de Distance Vector Routing es relativamente simple de entender e implementar.
- Convergencia Rápida: En redes pequeñas y bien administradas, el algoritmo puede converger rápidamente a las rutas óptimas.

Desventajas:

- Conteo a la Infinidad: Uno de los problemas más conocidos asociados con Distance Vector Routing es el "conteo a la infinitud". Si una ruta falla y esa información tarda en propagarse, los nodos pueden calcular rutas incorrectas, lo que lleva a bucles en la red.
- Consumo de Ancho de Banda: La actualización periódica de tablas de enrutamiento puede generar un consumo significativo de ancho de banda, especialmente en redes grandes.

Aplicaciones típicas:
El algoritmo de Distance Vector Routing se ha utilizado históricamente en redes pequeñas y simples, como redes locales (LANs). Aunque ha sido reemplazado por algoritmos más avanzados en la mayoría de las redes de área amplia (WANs), todavía se puede encontrar en uso en situaciones específicas. Además, su concepto de enrutamiento basado en vectores de distancia ha influido en el diseño de protocolos más modernos y eficientes, como OSPF y EIGRP (Marina & Das, 2001).

Algoritmo de Dijkstra:

O también conocido como el Algoritmo de los caminos más cortos, comienza desde un nodo de inicio a todos los demás en un grafo. Comienza con un conjunto de distancias infinitas y ajusta las distancias a través de nodos vecinos, selecciona el nodo con la distancia mínima no visitada y repite el proceso hasta visitar todos los nodos. El resultado es el camino más corto y sus distancias desde el nodo de inicio. Para esta implementación se utilizo la manera de representación en matriz, que permite conocer la cantidad de nodos y sus precios a comparación de los demás.

Funcionamiento:

1. Inicialización: Al momento de crear un nuevo nodo, la matriz se expande tomando en cuenta las referencias a otros nodos. Y al momento de enviarlo, utilizando Dijkstra obtiene el camino más corto, su recorrido y si este no es posible lo devuelve con un camino infinito.
2. Retransmisión: Al recibir un paquete un nodo y no ser el receptor, lo envia por el siguiente nodo con la distancia más corta.
3. Prevención de bucles infinitos: De ser el camino infinito, el mensaje se bloquea y no es retransmitido para ningún nodo y para al momento de encontrar al receptor, de tal forma de que un mensaje en bucle infinito no se podría producir.

Ventajas:

- Es un algoritmo simple y relativamente sencillo de replicar en dado caso se tenga una buena manera de computarización del grafo.
- Trabaja bien con grafos cortos y con pesos positivos dado que su complejidad depende de la cantidad de nodos con los que trabaje.
- Independientemente de la manera en que se esté implementando, si es correcto siempre encuentra el camino más corto o de menores pesos.

Desventajas:

- Al depender de los nodos, este no trabaja considerablemente bien a comparación de otros algoritmos con grafos de tamaño grande.
- Incluir pesos negativos dentro del grafo implicaría un problema de ejecución, siendo necesario un cambio respecto a la escala de los pesos
- Es necesario la reestructuración del camino ya que no encuentra el camino más corto en un solo recorrido.

Aplicaciones típicas:

Dijkstra abre paso a varias aplicaciones simples respecto a la teoría de grafos, algunas aplicaciones en las cuales se puede resultar útil su inclusión pueden ser manejo de circuitos cerrados pequeños con componentes que utilizan interconectividad u otra actividad que resulta bastante ingeniosa es la planificación de recursos compartidos, ya que al intentar transmitir información mediante distintos canales, Dijkstra puede proveer siempre la ruta más eficiente.

Resultados:

Para el algoritmo de Flooding, como se indicó en el PDF del laboratorio, se podía optar a realizarlo hard-coded. A continuación se presentan los inputs que se designaron. Sin embargo, cabe destacar que el programa también permite el ingreso manual de los inputs.

Topología:

```
topology = {
    "NodeA": ["NodeB", "NodeC"],
    "NodeB": ["NodeA", "NodeD"],
    "NodeC": ["NodeA", "NodeD"],
    "NodeD": ["NodeB", "NodeC"],
}
```

Envíos:

```
nodes["NodeA"].send_message("Hello, NodeB!", "NodeB")
nodes["NodeD"].receive_message("Hi there!", "NodeB", "NodeA,NodeC")
nodes["NodeC"].send_message("Message from NodeC", "NodeD")
```

Para el algoritmo de Dijsktra se utilizó una implementación más dinámica, en la que mediante un archivo de texto, y un par de json era posible implementar su lógica. En cada terminal se indicaba que se deseaba utilizar Dikstra, se preguntaba el nombre del nodo y esto creaba su espacio dentro de la matriz de adyacencia que tiene registro de todas las conectividades, al tener la cantidad de terminales según la cantidad de nodos solicitados, se podian realizar las inter conectividades entre dichos nodos, de tal manera que se quedarán registradas en la matriz. El algoritmo de Dijkstra entraba en pie al momento de intentar enviar un mensaje, ya que encontraba el camino más corto entre dos nodos, devolvió el peso y el path y de esta manera se podía obtener la secuencia de envío de mensajes replicada de tal manera que se cumplía con el reenvío del mensaje.

Topología:

matriz_adyacencia = {

0, 2, 0, 6, 0, 9
2, 0, 3, 8, 5, 0
0, 3, 0, 0, 7, 0
6, 8, 0, 0, 9, 0
0, 5, 7, 9, 0, 0
0, 0, 0, 0, 0, 0

}

Envíos:

ruta = 0,1,4,5,6

Mensaje de 0 a 1 (A a B) = “probando enviar mensaje”
Mensaje de 1 a 4 (B a D) = “probando enviar mensaje”
Mensaje de 4 a 5 (D a E) = “probando enviar mensaje”
Mensaje de 5 a 6 (E a F) (mostrar) = “probando enviar mensaje”

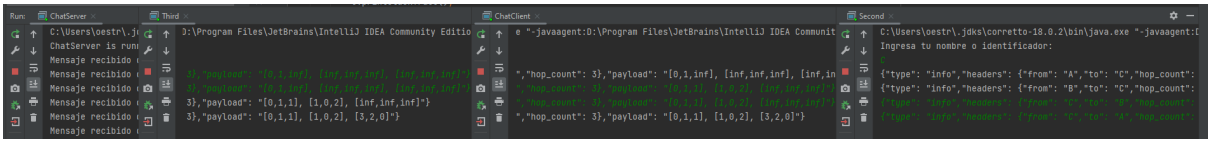
json presentado al final para F:

{

“type” : ”message”,
“headers” : [“from” : ”A”, “to” : ”F”, “hop_count” : 4, ...],
“payload” : “probando enviar mensaje”

}

En el caso del algoritmo de Distance Vector Routing se implementó una estrategia en Java donde se utilizó un sistema servidor-cliente para poder simular mejor la forma en que se trabajaba cada nodo. De forma en que inicialmente se actualizaban los nodos al enviar el mensaje simulando estas llamadas a tal forma de actualizar cada vector.



Topología:

De este modo al actualizar la tabla de cada vector obtendremos una tabla general de esta forma para un sistema de 3 nodos (A, B, C):

matriz_adyacencia = {

0, 1, 3

1, 0, 2

3, 2, 0

}

Envíos:

Al enviar un mensaje, ya bastaría con enviarlo por ejemplo desde el nodo A hacia el C. Este en el código se muestra como que hace un mensaje broadcast, pero como solo se trata de una simulación, cada salto es impreso únicamente en los nodos que fueron usados como rutas.

ruta = A, B, C

Ingrese el mensaje que A enviará: Hola C
Mensaje (A a B) = “Hola C”
Mensaje (B a C) = “Hola C”
Mensaje recibido en C: Hola C

json presentado al final para C que pasó por B:

{

“type” : ”message”,

“headers” : {“from” : ”A”, “to” : ”C”, “hop_count” : 3},

“payload” : “Hola C”

}

Discusión

Al implementar el Algoritmo de Flooding en Python, es esencial anticipar y abordar posibles problemas para asegurar una operación fluida y eficiente. Uno de los principales desafíos asociados con el Flooding es la posibilidad de que los mensajes entren en bucles infinitos. Para contrarrestar este problema, recomendamos incorporar un campo adicional en el encabezado del paquete o mensaje. Esta opción consistiría en una lista que documente cada nodo por el que el mensaje ha transitado durante su propagación.

La simplicidad inherente del algoritmo de Flooding se refleja en la estructura del código, lo que facilita su comprensión y uso. Durante la simulación, se observó que cada nodo reenvía los mensajes a todos sus vecinos, lo que garantiza la entrega del mensaje a todos los nodos en la red. Sin embargo, es importante destacar que el algoritmo de inundación puede generar un tráfico de red significativo, especialmente en redes grandes, debido a la duplicación de mensajes. Esto puede ser ineficiente en términos de ancho de banda y consumo de recursos. Por lo tanto, en escenarios de redes prácticas, se deben considerar estrategias de control de inundación y mecanismos para evitar bucles de reenvío de mensajes. Además, es fundamental evaluar su rendimiento en situaciones con mayor complejidad topológica y condiciones de red variables.

A medida que el mensaje se desplaza a través de la red, cada nodo involucrado se añade a esta lista. Por lo tanto, cuando un nodo recibe un paquete, puede revisar esta lista en el encabezado y determinar si ya ha procesado dicho paquete anteriormente. Al identificar su propia dirección en la lista, el nodo entenderá que ya ha gestionado ese paquete, evitando su retransmisión. Esta estrategia no solo previene la recurrencia de bucles, sino que también minimiza el tráfico redundante en la red. Es vital que los desarrolladores consideren esta recomendación al diseñar y codificar aplicaciones basadas en el Flooding en Python.

El funcionamiento del algoritmo de Distance Vector Routing es fundamental para comprender cómo los nodos en una red de comunicaciones colaboran para determinar las rutas óptimas hacia destinos específicos. Este enfoque se basa en la idea de que cada nodo en la red mantiene una tabla de enrutamiento que contiene información sobre las distancias y rutas hacia otros nodos. A medida que los nodos intercambian información con sus vecinos, actualizan sus tablas de enrutamiento para reflejar las distancias más cortas conocidas hacia otros destinos. Lo más intrigante de este algoritmo es su capacidad para "saltar" de un nodo a otro en busca de la mejor ruta, como si se tratara de una cadena de mensajes pasados de un vecino al siguiente, cada uno contribuyendo con su conocimiento local. Sin embargo, esta aparente simplicidad esconde desafíos, como el problema del "conteo a la infinitud", en el que los nodos pueden calcular rutas incorrectas si la información de la red tarda en

propagarse. A pesar de estas limitaciones, el Distance Vector Routing sigue siendo una piedra angular en la teoría de enrutamiento de redes y ha influido en el desarrollo de algoritmos más avanzados y eficientes en el campo de las comunicaciones.

Al momento de implementar el algoritmo de Dijkstra en Java utilizando varias terminales se puede argumentar que es un enfoque especialmente útil en situaciones en las que se requiere encontrar la ruta más corta en un grafo. El programa Java comenzaría solicitando especificar nodos de origen y destino para calcular la ruta más corta. Utilizando la lógica del algoritmo de Dijkstra, el programa encontraría el camino óptimo y mostraría los resultados según la interacción que se tenga con la matriz de adyacencia. Este enfoque interactivo y programático proporciona a los usuarios una manera precisa y flexible de encontrar rutas eficientes, lo que puede ser esencial en la toma de decisiones en logística, y enrutamiento que es el caso de este laboratorio.

Conclusiones

- El algoritmo de Flooding es uno de los enfoques más elementales en enrutamiento, basado en la idea de propagar paquetes de datos a todos los nodos vecinos.
- La gestión cuidadosa de nodos visitados es esencial para evitar bucles infinitos en la propagación de mensajes, lo que subraya la importancia de una implementación precisa y un monitoreo continuo.
- La principal fortaleza del algoritmo de Flooding radica en su capacidad para entregar mensajes a todos los nodos de la red, asegurando así una alta probabilidad de alcanzar el destino deseado.
- Distance Vector Routing, a pesar de su simplicidad y desafíos inherentes, sigue siendo una piedra angular en la teoría de enrutamiento de redes y ha influido en el desarrollo de protocolos de enrutamiento más avanzados y eficientes. Su legado perdura como un concepto fundamental en el campo de las comunicaciones.
- El algoritmo de Dijkstra es un algoritmo sencillo y poderoso que da resultados fenomenales al momento de analizar grafos relativamente reducidos.
- La representación del grafo y su manera de recorrerlo varía dentro de las implementaciones de este laboratorio pero todas son aceptables y válidas.

Referencias

Kurose, J. F., & Ross, K. W. (2020). Computer networking: A top-down approach (8th ed.). Pearson.

Marina, M. K., & Das, S. R. (2001, November). On-demand multipath distance vector routing in ad hoc networks. In Proceedings ninth international conference on network protocols. ICNP 2001 (pp. 14-23). IEEE.

Smith, B. R., Murthy, S., & Garcia, J. J. (1997). Securing distance-vector routing protocols. In Proceedings of SNDSS'97: Internet Society 1997 Symposium on Network and Distributed System Security (pp. 85-92). IEEE.

Tanenbaum, A. S., & Wetherall, D. J. (2011). Computer networks (5th ed.). Prentice Hall.

VGA. Un visualizador genérico de algoritmos. El Algoritmo de Dijkstra o de Caminos mínimos. (2023). Uned.es. http://atlas.uned.es/algoritmos/voraces/dijkstra.html#:~:text=El%20algoritmo%20de%20Dijkstra%2C%20tambi%C3%A9n,con%20pesos%20en%20cada%20arista.