Universidad del Valle De Guatemala

Facultad de Ingeniería

Redes

Laboratorio 3: Algoritmos de enrutamiento

Jose Hernández 20053
Pablo González 20362
Javier Mombiela 20067

Guatemala, 30 de mayo 2023

1. Descripción de la práctica.
   En esta parte del laboratorio, se explora el funcionamiento de los algoritmos de enrutamiento más importantes para las redes de computadora, con esta práctica se busca comprender cómo los mensajes se dirigen a través de nodos interconectados en una red y como las tablas de enrutamiento son fundamentales para dicho proceso, el objetivo principal de la práctica es implementar 3 algoritmos de enrutamiento,los algoritmos a implementar incluyen Dijkstra, flooding y por último Distance Vector Routing Se puede mencionar que la práctica se divide en dos fases principales. En la primera fase, se simula la creación y actualización de las tablas de enrutamiento. Los nodos envían sus vectores de distancia a los nodos vecinos, y este proceso se repite hasta que las tablas convergen y los nodos tienen la información completa para enrutar los mensajes de manera eficiente. En la segunda fase, se simula el envío de mensajes a través de la red utilizando las tablas de enrutamiento previamente creadas. Cada nodo consulta su tabla de enrutamiento para determinar la mejor ruta hacia el nodo de destino y reenvía el mensaje en consecuencia.
   Se establece un formato estándar de paquetes de comunicación en forma de JSON. Los paquetes contienen información sobre el tipo de mensaje, encabezados como el nodo de origen y destino, y otros detalles relevantes para la comunicación entre nodos. En conclusión el objetivo final es que podamos ganar experiencia práctica implementando los algoritmos de enrutamiento y en la simulación de la comunicación entre nodos de una red.

2. Descripción de los Algoritmos utilizados y su implementación.
   A. Dijkstra:
      El algoritmo de Dijkstra se utiliza para encontrar las rutas más cortas entre un nodo de inicio y todos los demás nodos en un grafo ponderado. Comienza inicializando las distancias de todos los nodos desde el nodo de inicio como infinito, excepto para el nodo de inicio, cuya distancia se establece en 0. Luego, itera sobre los nodos vecinos y actualiza las distancias si encuentra un camino más corto a través del nodo actual. Este proceso continúa hasta que todos los nodos se visitan y las distancias convergen. Finalmente, se construyen las rutas más cortas y sus distancias en función de las distancias actualizadas y los nodos predecesores.

   B. Flooding:
      El algoritmo de Flooding es un método simple y a menudo ineficiente para enviar un mensaje a través de una red. Consiste en transmitir copias del mensaje a todos los nodos vecinos y permitir que cada nodo retransmita el mensaje a sus propios vecinos. En este caso, se inicia con una cola que contiene el nodo de inicio y la información del camino recorrido hasta ahora. El algoritmo continúa expandiendo esta cola, retransmitiendo el mensaje a todos los vecinos no visitados. El proceso se repite hasta que todos los nodos sean visitados al menos una vez. La salida es un conjunto de rutas y costos que se acumulan a lo largo de los caminos tomados.

C. Distance Vector Routing:

El enrutamiento de vector de distancia es un algoritmo que permite a los nodos en una red calcular las rutas más cortas hacia otros nodos. En este algoritmo, cada nodo mantiene una tabla de enrutamiento que contiene información sobre la distancia y el siguiente salto para llegar a otros nodos. El algoritmo comienza inicializando las distancias a todos los nodos excepto el nodo de inicio como infinito. Luego, itera sobre todos los nodos y sus vecinos, actualizando las distancias si encuentra una ruta más corta. Este proceso se repite hasta que no haya más actualizaciones en las tablas de enrutamiento. Una vez que la convergencia se alcanza, la tabla de enrutamiento se utiliza para construir la ruta más corta desde el nodo de inicio hasta un nodo destino dado.

D. Implementaciones:

```
dijkstra(startNode) {
        let distances = {};
        let visited = new Set();
        let predecessors = {};  // Aquí guardamos el nodo predecesor de cada nodo

        for (let node of this.nodes.keys()) {
            distances[node] = Infinity;
            predecessors[node] = null;  // Inicializamos todos los predecesores como null
        }
        distances[startNode] = 0;

        while (visited.size !== this.nodes.size) {
            let nodeToVisit = null;

            for (let node in distances) {
                if (nodeToVisit === null || distances[node] < distances[nodeToVisit]) {
                    if (!visited.has(node)) {
                        nodeToVisit = node;
                    }
                }
            }

            visited.add(nodeToVisit);

            let neighbors = this.nodes.get(nodeToVisit);
            for (let neighbor of neighbors) {
                let alternatePath = distances[nodeToVisit] + neighbor.weight;
                if (alternatePath < distances[neighbor.node]) {
                    distances[neighbor.node] = alternatePath;
                    predecessors[neighbor.node] = nodeToVisit;  // Actualizamos el predecesor
                }
            }
        }

        // Ahora construimos el objeto de ruta para cada nodo
        let routes = {};
        for (let node in distances) {
            routes[node] = { distance: distances[node], path: this._buildPath(predecessors, startNode, node) };
        }

        return routes;
    }
```

```
distanceVectorRouting(startNode, targetNode) {
        // Inicializar la tabla de enrutamiento para todos los nodos
        let routingTable = {};
        for (let node of this.nodes.keys()) {
            routingTable[node] = {
                distance: node === startNode ? 0 : Infinity,
                nextHop: node === startNode ? startNode : null
            };
        }

        let hasUpdates = true;
        while (hasUpdates) {
            hasUpdates = false;

            for (let currentNode of this.nodes.keys()) {
                for (let neighbor of this.nodes.get(currentNode)) {
                    // Si la distancia a un vecino + la distancia del vecino al destino es menor
                    // que la distancia conocida, actualizamos la tabla de enrutamiento
                    if (routingTable[currentNode].distance + neighbor.weight < routingTable[neighbor.node].distance) {
                        routingTable[neighbor.node].distance = routingTable[currentNode].distance + neighbor.weight;
                        routingTable[neighbor.node].nextHop = currentNode;
                        hasUpdates = true;
                    }
                }
            }
        }

        // Construir la ruta a partir de la tabla de enrutamiento
        let node = targetNode;
        let path = [node];
        while (node !== startNode) {
            node = routingTable[node].nextHop;
            path.unshift(node);
        }

        return {
            path: path,
            distance: routingTable[targetNode].distance
        };
    }
```

```
flooding(startNode) {
        let queue = [{ node: startNode, path: [startNode], cost: 0 }];
        let paths = {};

        while (queue.length > 0) {
            let current = queue.shift();
            let currentNode = current.node;
            let currentPath = current.path;
            let currentCost = current.cost;

            let neighbors = this.nodes.get(currentNode);
            for (let neighbor of neighbors) {
                if (!currentPath.includes(neighbor.node)) {
                    queue.push({
                        node: neighbor.node,
                        path: [...currentPath, neighbor.node],
                        cost: currentCost + neighbor.weight
                    });
                }
            }

            if (!paths[currentNode]) {
                paths[currentNode] = [];
            }
            paths[currentNode].push({ path: currentPath, cost: currentCost });
        }

        return paths;
    }
```

3. Resultados

Djikstra

```
| Nodo   | Distancia | Recorrido
|--------|-----------|------------
| NodoA  | 0         | NodoA
| NodoB  | 7         | NodoA → NodoB
| NodoC  | 7         | NodoA → NodoC
| NodoD  | 7         | NodoA → NodoI → NodoD
| NodoE  | 8         | NodoA → NodoI → NodoD → NodoE
| NodoF  | 8         | NodoA → NodoI → NodoD → NodoF
| NodoG  | 11        | NodoA → NodoI → NodoD → NodoF → NodoG
| NodoH  | 12        | NodoA → NodoI → NodoD → NodoF → NodoH
| NodoI  | 1         | NodoA → NodoI

El mensaje se enviara desde NodoA hasta NodoE
La ruta optima es: NodoA,NodoI,NodoD,NodoE
PS C:\Users\rjmom\OneDrive\Documentos\GitHub\Redes-Laboratorio-3\algoritmos>
```

Flooding

```
PS C:\Users\rjmom\OneDrive\Documentos\GitHub\Redes-Laboratorio-3\algoritmos> node flooding
[
  { path: [ 'NodoA', 'NodoI', 'NodoD', 'NodoE' ], cost: 8 },
  { path: [ 'NodoA', 'NodoC', 'NodoD', 'NodoE' ], cost: 13 },
  { path: [ 'NodoA', 'NodoB', 'NodoF', 'NodoD', 'NodoE' ], cost: 11 },
  { path: [ 'NodoA', 'NodoB', 'NodoF', 'NodoG', 'NodoE' ], cost: 16 },
  {
    path: [ 'NodoA', 'NodoI', 'NodoD', 'NodoF', 'NodoG', 'NodoE' ],
    cost: 15
  },
  {
    path: [ 'NodoA', 'NodoC', 'NodoD', 'NodoF', 'NodoG', 'NodoE' ],
    cost: 20
  }
]
PS C:\Users\rjmom\OneDrive\Documentos\GitHub\Redes-Laboratorio-3\algoritmos>
```

Flooding Simulación Dinámica

```
PS C:\Users\rjmom\OneDrive\Documentos\GitHub\Redes-Laboratorio-3\algoritmos> node floodingRouter
Router A received packet: Hello World! from Router null with TTL 3
Router B received packet: Hello World! from Router A with TTL 2
Router C received packet: Hello World! from Router B with TTL 1
Packet dropped at Router A, TTL expired.
Router C received packet: Hello World! from Router A with TTL 2
Router B received packet: Hello World! from Router C with TTL 1
Packet dropped at Router A, TTL expired.
PS C:\Users\rjmom\OneDrive\Documentos\GitHub\Redes-Laboratorio-3\algoritmos>
```

Distance Vector

```
Tabla de enrutamiento para NodoA:
Destino         Siguiente       Costo
NodoA           -               0
NodoB           NodoB           7
NodoC           NodoC           7
NodoD           NodoI           7
NodoE           NodoI           8
NodoF           NodoI           8
NodoG           NodoI           11
NodoH           NodoI           12
NodoI           NodoI           1
Camino más corto: [ 'NodoA', 'NodoI', 'NodoD', 'NodoE' ]
Distancia total: 8
PS C:\Users\rjmom\OneDrive\Documentos\GitHub\Redes-Laboratorio-3\algoritmos>
```

Distance Vector Simulación Dinámica

```
PS C:\Users\rjmom\OneDrive\Documentos\GitHub\Redes-Laboratorio-3\algoritmos> node distanceVectorRouter
Routing table for Router A:
Destination: A, Cost: 0, Next Hop: A
Destination: B, Cost: 1, Next Hop: B
Destination: C, Cost: 4, Next Hop: C
Routing table for Router B:
Destination: B, Cost: 0, Next Hop: B
Destination: A, Cost: 1, Next Hop: A
Destination: C, Cost: 2, Next Hop: C
Routing table for Router C:
Destination: C, Cost: 0, Next Hop: C
Destination: A, Cost: 4, Next Hop: B
Destination: B, Cost: 2, Next Hop: B
Updated routing table of Router A
Updated routing table of Router C
Routing table for Router A:
Destination: A, Cost: 0, Next Hop: A
Destination: B, Cost: 1, Next Hop: B
Destination: C, Cost: 3, Next Hop: B
Routing table for Router B:
Destination: B, Cost: 0, Next Hop: B
Destination: A, Cost: 1, Next Hop: A
Destination: C, Cost: 2, Next Hop: C
Routing table for Router C:
Destination: C, Cost: 0, Next Hop: C
Destination: A, Cost: 3, Next Hop: A
Destination: B, Cost: 2, Next Hop: B

Sending message from Router A to Router C
Forwarding message from Router B to Router C

Sending message from Router B to Router C
Received message at Router C: Hello, Router C!
PS C:\Users\rjmom\OneDrive\Documentos\GitHub\Redes-Laboratorio-3\algoritmos>
```

4. Discusión

Los tres algoritmos de enrutamiento implementados en esta práctica, Dijkstra, Flooding y Distance Vector Routing, representan enfoques diferentes para resolver el problema de cómo dirigir mensajes a través de una red de computadoras. Cada uno de estos algoritmos tiene sus propias características y aplicaciones adecuadas en función de las necesidades específicas de la red y los requisitos de enrutamiento.

Dijkstra se destaca por su capacidad para encontrar las rutas más cortas en redes ponderadas, lo que lo hace valioso cuando se necesita minimizar la latencia o el costo de transporte. Sin embargo, su desventaja radica en su demanda de recursos computacionales y ancho de banda debido a la necesidad de mantener información detallada sobre todas las rutas posibles. Además, no es la elección adecuada para redes con topologías cambiantes, ya que requiere actualizaciones constantes de las tablas de enrutamiento.

Por otro lado, el algoritmo de Flooding es extremadamente simple y garantiza la entrega de mensajes al transmitir copias a todos los nodos, sin requerir información previa sobre la topología de la red. Esto lo hace robusto ante cambios en la red y adecuado cuando la confiabilidad es prioritaria. Sin embargo, su principal limitación

es su ineficiencia, ya que puede generar un alto tráfico de red y un uso ineficiente de los recursos, lo que lo hace menos adecuado para redes grandes o congestionadas.

Distance Vector Routing, se caracteriza por ser fácil de implementar y adecuado para redes más pequeñas con cambios de topología menos frecuentes. Es robusto y puede adaptarse a cambios en la red sin generar un gran tráfico de actualización. Sin embargo, su principal limitación es su lenta convergencia en redes grandes o complejas, así como el problema de "conteo hacia el infinito", donde la información de enrutamiento se actualiza de manera ineficiente cuando hay nodos intermedios defectuosos.

5. Conclusiones
    a. No se logró hacer una simulación de Djikstra, ya que este es un algoritmo estático, por lo que necesita configurar todos los nodos y los pesos para que se pueda realizar el algoritmo.
    b. Para realizar la simulación de flooding no es “necesario” realizar una tabla de ruteo en sí, ya que al mandar el mensaje, de igual manera, solo le envía el paquete a todos los vecinos, por lo que no le importa la información de los pesos de cada uno.
    c. El distance vector routing es capaz de calcular rutas óptimas y mantener actualizadas las tablas de enrutamiento en una red. Este algoritmo es especialmente adecuado para redes más pequeñas y se destaca por su simplicidad de implementación y adaptabilidad.

6. Comentarios
    La práctica de laboratorio se puede mencionar que la práctica de laboratorio propuesta es una excelente oportunidad para aprender y profundizar en la compresión y la aplicación de los algoritmos de Dijkstra, Flooding y Distance Vector Routing, estos ofrecen enfoques diferentes para abordar el desafío de enrutar mensajes eficientemente en una red interconectada. La simulación de estos algoritmos hace que como grupo entendamos como funcionan los algoritmos en la teoría, si no también experimentar con los mismos, lo cual es bueno, en conjunto se puede mencionar que la práctica proporciona información valiosa para que adquiramos conocimientos profundos acerca de los algoritmos de enrutamiento y la comunicación de redes.

7. Referencias

Pedraza, L. F., López, D., & Salcedo, O. (2011). 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#:~:text=El%20algoritmo%20de%20Dijkstra%20proporciona

¿Qué es una inundación? - definición de techopedia - Redes 2023. (2023). Icy Science. https://es.theastrologypage.com/flooding

Vector de distancias. (2020, September 9). Wikipedia. https://es.wikipedia.org/wiki/Vector_de_distancias