

Guillermo Santos 191517  
Sara María Paguaga 20634

### Descripción de la práctica

Esta práctica consistió en implementar el algoritmo de enrutamiento seleccionado *Link State Routing* para enviar mensajes, actualizar tablas de enrutamiento y adaptar a cambios en la infraestructura. Para la implementación de *Link State Routing* fue necesario utilizar los algoritmos de *Dijkstra* y *Flooding*.

### Descripción de los algoritmos

#### **Dijkstra**

El algoritmo Dijkstra permite encontrar el camino más corto desde un nodo inicial hacia un nodo en un grafo con pesos. Para aplicar el algoritmo se siguen los siguientes pasos:

1. Seleccionar un nodo inicial y asignarle una distancia de 0 y un valor de infinito a los demás nodos.
2. Para cada nodo vecino del nodo actual que aún no ha sido visitado:
  - 3.1 Calcular una distancia desde el nodo actual hacia su vecino.
  - 3.2 Si la distancia calculada en paso anterior es menor que la distancia actual hacia el vecino, se actualiza la distancia hacia el nodo vecino con el cálculo del paso anterior.
3. Marcar el nodo actual como visitado.
4. Si hay nodos no visitados, seleccionar el nodo vecino con la menor distancia, establecerlo como nodo actual y repetir el paso 3. De lo contrario se termina el algoritmo.

(*Dijkstra's algorithm*, s. f.)

## Link State Routing

Este algoritmo es el reemplazo de *Distance Vector Routing*. Variantes de este algoritmo todavía se usan en la actualidad (Tanenbaum, 2010). Tiene cinco pasos:

1. Descubrir a los vecinos y sus direcciones.
2. Calcular distancias a los vecinos.
3. Construir un *packet* con la tabla de distancias.
4. Enviar el *packet* a los demás nodos utilizando Flooding u otro método y recibir las tablas de los vecinos.
5. Calcular la distancia más corta a cada router.
- 6.

Entonces la topología está distribuida en cada uno de los nodos, lo cuál lo hace bastante escalable y versátil. Uno de los retos es manejar correctamente las tablas que se reciben de los demás nodos. Dado que estas tablas pueden estar cambiando constantemente y es necesario tener la versión más actualizada. Por ejemplo, tenemos el siguiente ejemplo utilizado en el libro de Tanenbaum:

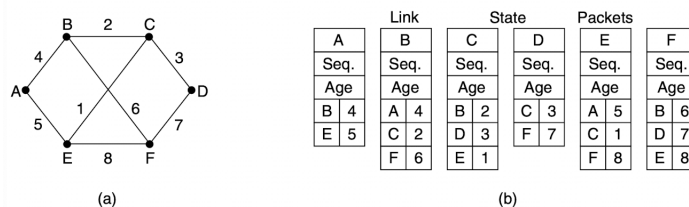


Figure 5-12. (a) A network. (b) The link state packets for this network.

Imagen 1 - Red de ejemplo - Link state routing

Desde el punto de vista del nodo A, al obtener todos los paquetes puede crear la topología completa de la red y posteriormente calcular las distancias.

## Flooding

El algoritmo de inundación, usado en redes, envía paquetes a todas las salidas disponibles, excluyendo la de origen, garantizando una distribución robusta y alcanzando siempre la ruta más corta. Para evitar la duplicación infinita de paquetes, se implementa un contador de saltos que disminuye con cada tránsito, y un registro de paquetes ya vistos, evitando retransmisiones y sobrecargas excesivas en la red (Tanenbaum, 2010).

A continuación, se explica a detalle el funcionamiento del algoritmo:

- **Inicialización:** Cada nodo recibe un paquete con un contador de saltos y un identificador único.
- **Verificación:** Al recibir un paquete, el nodo verifica si ya ha procesado un paquete con el mismo identificador. Si es así, descarta el paquete para evitar la duplicación.

- **Decremento del Contador de Saltos:** El nodo disminuye el contador de saltos del paquete por cada transmisión.
- **Reenvío o Descarte:**
  - Si el nodo es el destino, procesa el paquete.
  - Si el contador de saltos es mayor que cero, el nodo reenvía el paquete a todos sus vecinos, excepto al que le envió el paquete.
  - Si el contador de saltos llega a cero, el paquete se descarta para evitar ciclos infinitos.
- **Registro:** El nodo registra el identificador del paquete para no procesarlo nuevamente en el futuro.

Este proceso se repite hasta que el paquete alcanza su destino o es descartado por haber alcanzado un número máximo de saltos (contador de saltos igual a cero).

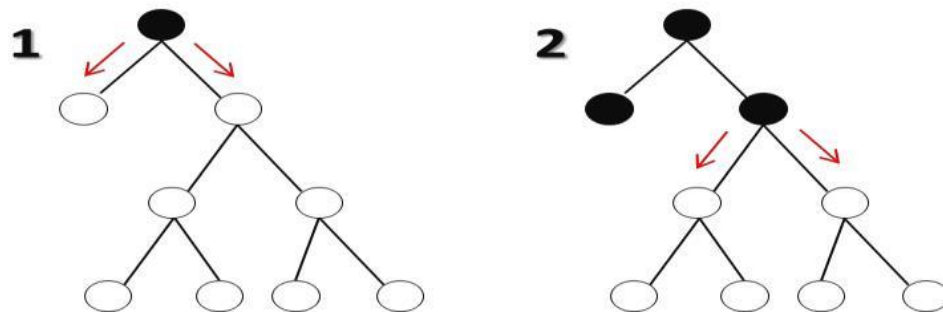
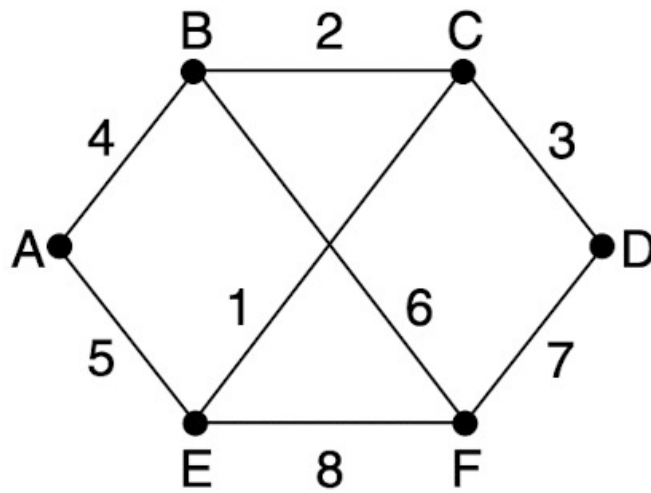


Imagen 3 - Funcionamiento del algoritmo Flooding

## Resultados

### Dijkstra



(a)

Imagen 2 - Red de ejemplo - Dijkstra

Forma en que se arma el grafo:

```
const graph = new Graph();

const nodeA = new Node("A");
const nodeB = new Node("B");
const nodeC = new Node("C");
const nodeD = new Node("D");
const nodeE = new Node("E");
const nodeF = new Node("F");

graph.addNode(nodeA);
graph.addNode(nodeB);
graph.addNode(nodeC);
graph.addNode(nodeD);
graph.addNode(nodeE);
graph.addNode(nodeF);
```

Se asignan los pesos a las aristas:

```
graph.addEdge(nodeA, nodeB, 4);  
graph.addEdge(nodeA, nodeE, 5);  
  
graph.addEdge(nodeB, nodeC, 2);  
graph.addEdge(nodeB, nodeF, 6);  
  
graph.addEdge(nodeC, nodeD, 3);  
graph.addEdge(nodeC, nodeE, 1);  
  
graph.addEdge(nodeE, nodeF, 8);  
graph.addEdge(nodeF, nodeD, 7);
```

Se coloca el nodo inicial y el nodo al que se quiere llegar.

```
const startNode = nodeA;  
const targetNode = nodeD;  
  
const djikstra = new Djikstra(startNode, targetNode, graph);  
djikstra.apply();
```

Para prueba obtenemos el siguiente camino:

```
> ts-node index.ts  
Dijkstra:  
Camino hacia el nodo D: A -> B -> C -> D  
Distancia: 9
```

En este caso se prueba llegar de D a E:

```
const startNode = nodeD;  
const targetNode = nodeE;  
  
const djikstra = new Djikstra(startNode, targetNode, graph);  
djikstra.apply();
```

y se obtiene el siguiente resultado:

```
> ts-node index.ts
Dijkstra:
Camino hacia el nodo B: D -> C -> B
Distancia: 5
```

## Link State Routing

Utilizando el ejemplo de la imagen 1, creamos el nodo A y posteriormente se simula la recepción de los paquetes. Por último, cuando ya se tiene la información de todos los nodos se ejecuta el algoritmo.

```
// Arrange
const node = new LinkStateRoutingNode('A');
// Simulating node discovery
node.addNeighbor('B', 4);
node.addNeighbor('E', 5);
// Simulating packets being received
const packetB = new LinkStateRoutingPacket(
  'B', new Map([['A', 4], ['C', 2], ['F', 6]]),
);
const packetC = new LinkStateRoutingPacket(
  'C', new Map([['B', 2], ['D', 3], ['E', 1]]));
const packetD = new LinkStateRoutingPacket(
  'D', new Map([['C', 3], ['F', 7]]));
const packetE = new LinkStateRoutingPacket(
  'E', new Map([['A', 5], ['C', 1], ['F', 8]]));
const packetF = new LinkStateRoutingPacket(
  'F', new Map([['B', 6], ['D', 7], ['E', 8]]));

node.processPacket(packetB);
node.processPacket(packetC);
node.processPacket(packetD);
node.processPacket(packetE);
node.processPacket(packetF);
// Act
const paths = linkStateRouting(node);
```

A continuación se muestran los resultados

```
debugger attached
console.log
Dijkstra:
Camino hacia el nodo B: A -> B
Distancia: 4

    at Dijkstra.showPath (src/algorithms/Dijkstra.ts:88:15)

console.log
Dijkstra:
Camino hacia el nodo E: A -> E
Distancia: 5

    at Dijkstra.showPath (src/algorithms/Dijkstra.ts:88:15)

console.log
Dijkstra:
Camino hacia el nodo C: A -> B -> C
Distancia: 6

    at Dijkstra.showPath (src/algorithms/Dijkstra.ts:88:15)

console.log
Dijkstra:
Camino hacia el nodo F: A -> B -> F
Distancia: 10

    at Dijkstra.showPath (src/algorithms/Dijkstra.ts:88:15)

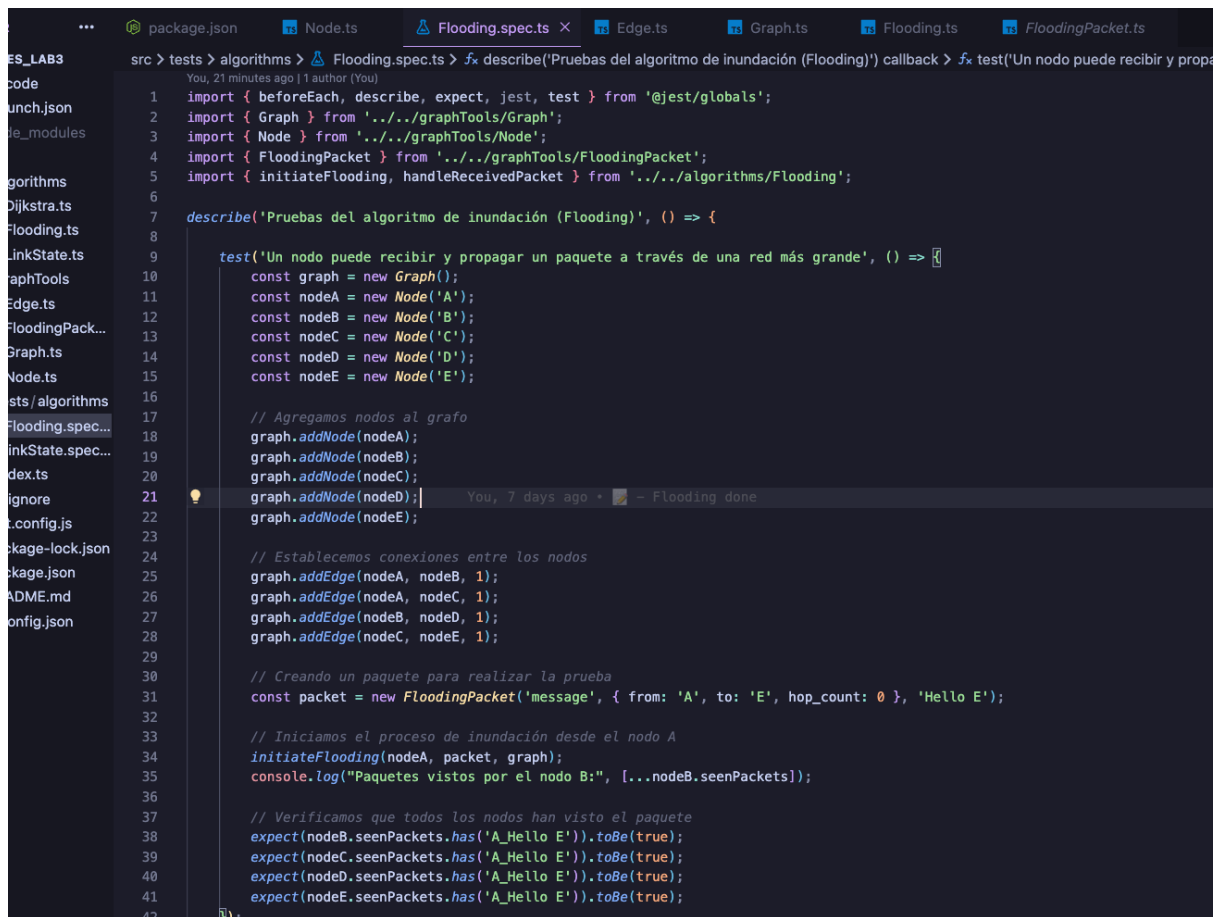
console.log
Dijkstra:
Camino hacia el nodo D: A -> B -> C -> D
Distancia: 9
```

## Flooding

Se realizaron dos pruebas importantes para verificar el correcto funcionamiento del algoritmo de Flooding. Dichas pruebas fueron realizadas utilizando Jest y consisten en lo siguiente:

- **Propagación de Paquetes:** Verifica que un paquete se propaga adecuadamente a través de una red amplia, confirmando que todos los nodos registran el paquete originado desde un punto específico.
- **Evitación de Re-procesamiento:** Asegura que un nodo no procesa de nuevo un paquete que ya ha visto, evitando ciclos infinitos, lo cual se confirma mediante el contador de saltos (hop count) del paquete.

Ambas pruebas confirman la funcionalidad fundamental y la eficiencia del algoritmo de inundación.



```
src > tests > algorithms > Flooding.spec.ts > describe('Pruebas del algoritmo de inundación (Flooding)') callback > test('Un nodo puede recibir y propa
You, 21 minutes ago | 1 author (You)
code
unch.json
e_modules
gorithms
Dijkstra.ts
Flooding.ts
LinkState.ts
raphTools
Edge.ts
FloodingPack...
Graph.ts
Node.ts
ests/algorithms
Flooding.spec...
LinkState.spec...
dex.ts
gnore
t.config.js
ckage-lock.json
ckage.json
ADME.md
nfig.json
1
import { beforeEach, describe, expect, jest, test } from '@jest/globals';
2
import { Graph } from '../../graphTools/Graph';
3
import { Node } from '../../graphTools/Node';
4
import { FloodingPacket } from '../../graphTools/FloodingPacket';
5
import { initiateFlooding, handleReceivedPacket } from '../../algorithms/Flooding';
6
7
describe('Pruebas del algoritmo de inundación (Flooding)', () => {
8
9
test('Un nodo puede recibir y propagar un paquete a través de una red más grande', () => {
10
const graph = new Graph();
11
const nodeA = new Node('A');
12
const nodeB = new Node('B');
13
const nodeC = new Node('C');
14
const nodeD = new Node('D');
15
const nodeE = new Node('E');
16
17
// Agregamos nodos al grafo
18
graph.addNode(nodeA);
19
graph.addNode(nodeB);
20
graph.addNode(nodeC);
21
graph.addNode(nodeD);
22
graph.addNode(nodeE);
23
24
// Establecemos conexiones entre los nodos
25
graph.addEdge(nodeA, nodeB, 1);
26
graph.addEdge(nodeA, nodeC, 1);
27
graph.addEdge(nodeB, nodeD, 1);
28
graph.addEdge(nodeC, nodeE, 1);
29
30
// Creando un paquete para realizar la prueba
31
const packet = new FloodingPacket('message', { from: 'A', to: 'E', hop_count: 0 }, 'Hello E');
32
33
// Iniciamos el proceso de inundación desde el nodo A
34
initiateFlooding(nodeA, packet, graph);
35
console.log("Paquetes vistos por el nodo B:", [...nodeB.seenPackets]);
36
37
// Verificamos que todos los nodos han visto el paquete
38
expect(nodeB.seenPackets.has('A_Hello E')).toBe(true);
39
expect(nodeC.seenPackets.has('A_Hello E')).toBe(true);
40
expect(nodeD.seenPackets.has('A_Hello E')).toBe(true);
41
expect(nodeE.seenPackets.has('A_Hello E')).toBe(true);
42
});
43
});
```



```
... package.json Node.ts Flooding.spec.ts x Edge.ts Graph.ts Flooding.ts FloodingPacket.ts
ES_LAB3 src > tests > algorithms > Flooding.spec.ts > fx describe('Pruebas del algoritmo de inundación (Flooding)') callback > fx test('Un nodo puede recibir y propa
code 33 // Iniciamos el proceso de inundación desde el nodo A
unch.json 34 initiateFlooding(nodeA, packet, graph);
35 console.log("Paquetes vistos por el nodo B:", [...nodeB.seenPackets]);
36
37 // Verificamos que todos los nodos han visto el paquete
38 expect(nodeB.seenPackets.has('A_Hello E')).toBe(true);
39 expect(nodeC.seenPackets.has('A_Hello E')).toBe(true);
40 expect(nodeD.seenPackets.has('A_Hello E')).toBe(true);
41 expect(nodeE.seenPackets.has('A_Hello E')).toBe(true);
42 });
43
44 test('Un nodo no vuelve a procesar un paquete ya visto en una red más grande', () => {
45   const graph = new Graph();
46   const nodeA = new Node('A');
47   const nodeB = new Node('B');
48   const nodeC = new Node('C');
49   const nodeD = new Node('D');
50
51   // Agregamos nodos al grafo
52   graph.addNode(nodeA);
53   graph.addNode(nodeB);
54   graph.addNode(nodeC);
55   graph.addNode(nodeD);
56
57   // Establecemos conexiones entre los nodos
58   graph.addEdge(nodeA, nodeB, 1);
59   graph.addEdge(nodeB, nodeC, 1);
60   graph.addEdge(nodeC, nodeD, 1);
61
62   // Creando un paquete para realizar la prueba
63   const packet = new FloodingPacket('message', { from: 'A', to: 'D', hop_count: 0 }, 'Hello D');
64
65   // Iniciamos el proceso de inundación desde el nodo A
66   initiateFlooding(nodeA, packet, graph);
67
68   // Simulamos el reenvío del mismo paquete
69   handleReceivedPacket(nodeD, packet, nodeC, graph);
70
71   // Verificamos que el contador de saltos (hop count) es el esperado
72   expect(packet.headers.hop_count).toBe(3);
73 });
74 });
75
```

```
● unclpete@Pedros-MacBook-Pro uvg_redes_lab3 % npm test
> lab03@1.0.0 test
> jest
PASS src/tests/algorithms/Flooding.spec.ts
  ● Console
    console.log
      Iniciando inundación desde el nodo A
        at initiateFlooding (src/algorithms/Flooding.ts:7:13)
    console.log
      Node A received packet from A intended for E
        at Node.receivePacket (src/graphTools/Node.ts:17:17)
    console.log
      Node A has neighbors: [ 'B', 'C' ]
        at Node.receivePacket (src/graphTools/Node.ts:19:17)
    console.log
      Node B received packet from A intended for E
        at Node.receivePacket (src/graphTools/Node.ts:17:17)
    console.log
      Node B has neighbors: [ 'A', 'D' ]
        at Node.receivePacket (src/graphTools/Node.ts:19:17)
    console.log
      Node D received packet from A intended for E
        at Node.receivePacket (src/graphTools/Node.ts:17:17)
    console.log
      Node D has neighbors: [ 'B' ]
        at Node.receivePacket (src/graphTools/Node.ts:19:17)
    console.log
      Node B received packet from A intended for E
```

```

console.log
  Node A received packet from A intended for E

  at Node.receivePacket (src/graphTools/Node.ts:17:17)

console.log
  Node A has neighbors: [ 'B', 'C' ]

  at Node.receivePacket (src/graphTools/Node.ts:19:17)

console.log
  Node B received packet from A intended for E

  at Node.receivePacket (src/graphTools/Node.ts:17:17)

console.log
  Node B has neighbors: [ 'A', 'D' ]

  at Node.receivePacket (src/graphTools/Node.ts:19:17)

console.log
  Node D received packet from A intended for E

  at Node.receivePacket (src/graphTools/Node.ts:17:17)

console.log
  Node D has neighbors: [ 'B' ]

  at Node.receivePacket (src/graphTools/Node.ts:19:17)

console.log
  Node B received packet from A intended for E

  at Node.receivePacket (src/graphTools/Node.ts:17:17)

console.log
  Node B has neighbors: [ 'A', 'D' ]

  at Node.receivePacket (src/graphTools/Node.ts:19:17)

console.log
  Node C received packet from A intended for E

  at Node.receivePacket (src/graphTools/Node.ts:17:17)

```

```

  at Node.receivePacket (src/graphTools/Node.ts:19:17)

console.log
  Node B received packet from A intended for E

  at Node.receivePacket (src/graphTools/Node.ts:17:17)

console.log
  Node B has neighbors: [ 'A', 'D' ]

  at Node.receivePacket (src/graphTools/Node.ts:19:17)

console.log
  Node C received packet from A intended for E

  at Node.receivePacket (src/graphTools/Node.ts:17:17)

console.log
  Node C has neighbors: [ 'A', 'E' ]

  at Node.receivePacket (src/graphTools/Node.ts:19:17)

console.log
  Node E received packet from A intended for E

  at Node.receivePacket (src/graphTools/Node.ts:17:17)

console.log
  Node E has neighbors: [ 'C' ]

  at Node.receivePacket (src/graphTools/Node.ts:19:17)

console.log
  Nodo E recibió el paquete: Hello E

  at Node.receivePacket (src/graphTools/Node.ts:31:21)

console.log
  Paquetes vistos por el nodo B: [ 'A_Hello E' ]

  at Object.<anonymous> (src/tests/algorithms/Flooding.spec.ts:35:17)

console.log
  Iniciando inundación desde el nodo A

```

```
    at Object.<anonymous> (src/tests/algorithms/Flooding.spec.ts:35:17)

console.log
  Iniciando inundación desde el nodo A

    at initiateFlooding (src/algorithms/Flooding.ts:7:13)

console.log
  Node A received packet from A intended for D

    at Node.receivePacket (src/graphTools/Node.ts:17:17)

console.log
  Node A has neighbors: [ 'B' ]

    at Node.receivePacket (src/graphTools/Node.ts:19:17)

console.log
  Node B received packet from A intended for D

    at Node.receivePacket (src/graphTools/Node.ts:17:17)

console.log
  Node B has neighbors: [ 'A', 'C' ]

    at Node.receivePacket (src/graphTools/Node.ts:19:17)

console.log
  Node C received packet from A intended for D

    at Node.receivePacket (src/graphTools/Node.ts:17:17)

console.log
  Node C has neighbors: [ 'B', 'D' ]

    at Node.receivePacket (src/graphTools/Node.ts:19:17)

console.log
  Node B received packet from A intended for D

    at Node.receivePacket (src/graphTools/Node.ts:17:17)

console.log
  Node B has neighbors: [ 'A', 'C' ]
```

```
console.log
  Node B received packet from A intended for D

    at Node.receivePacket (src/graphTools/Node.ts:17:17)

console.log
  Node B has neighbors: [ 'A', 'C' ]

    at Node.receivePacket (src/graphTools/Node.ts:19:17)

console.log
  Node C received packet from A intended for D

    at Node.receivePacket (src/graphTools/Node.ts:17:17)

console.log
  Node C has neighbors: [ 'B', 'D' ]

    at Node.receivePacket (src/graphTools/Node.ts:19:17)

console.log
  Node B received packet from A intended for D

    at Node.receivePacket (src/graphTools/Node.ts:17:17)

console.log
  Node B has neighbors: [ 'A', 'C' ]

    at Node.receivePacket (src/graphTools/Node.ts:19:17)

console.log
  Node D received packet from A intended for D

    at Node.receivePacket (src/graphTools/Node.ts:17:17)

console.log
  Node D has neighbors: [ 'C' ]

    at Node.receivePacket (src/graphTools/Node.ts:19:17)

console.log
  Nodo D recibió el paquete: Hello D

    at Node.receivePacket (src/graphTools/Node.ts:31:21)
```

```
    },  
    LinkStatePath {  
      destination: 'D',  
      cost: 9,  
      steps: [ 'A', 'B', 'C', 'D' ]  
    }  
  ]  
  
  at linkStateRouting (src/algorithms/LinkState.ts:104:10)
```

```
Test Suites: 2 passed, 2 total  
Tests: 6 passed, 6 total  
Snapshots: 0 total  
Time: 1.174 s  
Ran all test suites.  
○ unclpete@Pedros-MacBook-Pro uvg_redes_lab3 %
```

## Enrutamiento

Para aplicar el algoritmo de enrutamiento fue simulado con la siguiente topología:

```
{  
  "type": "topo",  
  "config": {  
    "A": [  
      "B"  
    ],  
    "B": [  
      "A"  
    ]  
  }  
}
```

Y la siguiente configuración:

```
{  
  "type": "names",  
  "config": {  
    "A": "san191517@alumchat.xyz",  
    "B": "san191517test@alumchat.xyz"  
  }  
}
```

## Proceso de simulación de ruteo:

### Cliente 1:

```
> ts-node src/index.ts "san191517" "123456" "link-state"

attached listeners
(node:99048) Warning: Setting the NODE_TLS_REJECT_UNAUTHORIZED environment variable to '0' makes TLS connections and HTTPS requests insecure by disabling certificate verification.
(Use `node --trace-warnings ...` to show where the warning was created)
Send message to: Router is online.
Sending echo to san191517test {"type":"echo","headers":{"from":"A","to":"B"},"payload":{"timestamp1":"1694541636214","timestamp2":""}}

Sending echo to san191517test {"type":"echo","headers":{"from":"B","to":"A"},"payload":{"timestamp1":"1694541640602","timestamp2":"1694541640857"}}
Sending link state packet to san191517test {"type":"info","headers":{"from":"A","to":"B"},"payload":{"B":255}}
Link state packet received
```

### Cliente 2:

```
pyenv shell 3.11.3
> pyenv shell 3.11.3
> npm start san191517test 123456 link-state

> 01_proyecto_xmpp@1.0.0 start
> ts-node src/index.ts "san191517test" "123456" "link-state"

attached listeners
(node:99052) Warning: Setting the NODE_TLS_REJECT_UNAUTHORIZED environment variable to '0' makes TLS connections and HTTPS requests insecure by disabling certificate verification.
(Use `node --trace-warnings ...` to show where the warning was created)
Send message to: Router is online.
Sending echo to san191517 {"type":"echo","headers":{"from":"B","to":"A"},"payload":{"timestamp1":"1694541640602","timestamp2":""}}
Sending link state packet to san191517 {"type":"info","headers":{"from":"B","to":"A"},"payload":{"A":255}}
Link state packet received
```

Lo primero que se hace es enviar un mensaje de tipo echo y se procede a enviar un paquete con el algoritmo Link State, se puede ver en ambos clientes que el paquete enviado con el algoritmo ha sido recibido.

### Puntos extras:

La configuración se actualiza dinámicamente por medio del siguiente método:

```
private handleConfiguration(config: ConfigDTO) {
  if (config.type === 'names') {
    for (let [key, value] of Object.entries(config.config)) {
      this.routerNames.set(key, value.split('@')[0]);
    }
  } else if (config.type === 'topo') {
    for (let [key, value] of Object.entries(config.config)) {
      this.topography.set(key, value);
    }
  }
}
```

Se utiliza al cargar los archivos de configuración:

```
loadConfigFiles() {  
  // loading names  
  const namesConfig: ConfigDTO = JSON.parse(fs.readFileSync(path.join(__dirname, './names.json'), 'utf-8'));  
  this.handleConfiguration(namesConfig);  
  // loading topography  
  const topoConfig: ConfigDTO = JSON.parse(fs.readFileSync(path.join(__dirname, './topo.json'), 'utf-8'));  
  this.handleConfiguration(topoConfig);  
}
```

## Discusión

El algoritmo Link State Routing se utiliza para la simulación de red y a los algoritmos *Dijkstra* y *Flooding* como base para funcionar. Los distintos tipos de paquetes, tanto echo, info y message fueron de beneficio para poder implementar la simulación. Fue de importancia tener estructuras dinámicas en caso que un nodo ‘cayera’ o se conectara un nuevo nodo y de esta forma tener información fiable en las tablas de enrutamiento y no afectar la comunicación entre nodos.

## Conclusiones

- Las estructuras dinámicas son esenciales para una comunicación eficaz.
- El algoritmo Link State permite gestionar correctamente las tablas recibidas de otros nodos a pesar de los cambios que se puedan llegar a dar.

## Comentarios del grupo

- Este fue un laboratorio interesante ya que nos permitió conocer y aplicar un algoritmo de enrutamiento y el proceso a llevar a cabo para recibir y enviar mensajes, así como para actualizar las tablas de enrutamiento.

## Referencias

- Tanenbaum, A. S., & Wetherall, D. J. (2010). Computer Networks (5th ed.). Pearson.
- *Dijkstra's algorithm*. (s. f.). <https://www.programiz.com/dsa/dijkstra-algorithm>