

## **Redes**

### Laboratorio 3 - Algoritmos de Enrutamiento

---

## **Descripción de la Práctica**

En este laboratorio, se analizarán los algoritmos de enrutamiento y sus aplicaciones prácticas en un entorno simulado utilizando el protocolo XMPP. El enfoque principal es entender cómo las tablas de enrutamiento gestionan y optimizan el flujo de datos en una red dinámica, adaptándose a cambios estructurales. Implementaremos y probaremos dos algoritmos: Flooding y Link State Routing. Cada uno de estos algoritmos será ejecutado en un nodo específico dentro de nuestra red simulada en alumchat.lol, permitiendo a cada nodo actuar como un cliente que puede enviar y recibir mensajes.

## **Descripción de los algoritmos**

Como se mencionó anteriormente, los algoritmos a implementar serán Flooding y Link State Routing. El primero consiste en un algoritmo de enrutamiento simple donde cada nodo retransmite los paquetes/mensajes entrantes a todos sus enlaces salientes, excepto por el que recibió el paquete, aprovecha todas las rutas posibles y encuentra la más corta naturalmente. Cabe mencionar que, este algoritmo puede resultar en un uso ineficiente del ancho de banda y en la duplicación de mensajes, lo que aumenta la carga en la red. En el segundo algoritmo, cada router construye un mapa de la conectividad de la red, que utiliza para calcular rutas más cortas utilizando el algoritmo de Dijkstra, esto permite una rápida convergencia y adaptación ante cambios en la red, pero requiere más uso de memoria y capacidad de procesamiento. Tanenbaum, Andrew S.; Wetherall, David J. (March 23, 2010).

## **Implementación**

Ambos algoritmos fueron implementados usando javascript, para la implementación del algoritmo de flooding, al estar en línea, se cargan los contactos y permite al usuario enviar mensajes, los cuales son reenviados a todos los contactos si no encuentran directamente al destinatario. Además, se analiza y envía mensajes basados en la interacción del usuario y actualiza dinámicamente las rutas de mensajes según la topología de la red. Para la implementación de Link State Routing, al conectarse, se solicita y procesa la topología de la red para luego permitir el envío de mensajes. Se implementa el algoritmo link state para

determinar rutas óptimas mediante Dijkstra y se actualizan dinámicamente las rutas basadas en la topología recibida.

## Resultados

```
PROBLEMAS  SALIDA  TERMINAL  PUERTOS  COMENTARIOS  CONSOLA DE DEPURACIÓN
---online
Enter the message: c: [ 'alb210041@alumchat.lol' ]
Enter the message: Received message from alb210041@alumchat.lol/web: {"type":"echo","from":"alb210041@alumchat.lol","to":"her21000@alumchat.lol","hops":0,"headers":{"request":"topology"},"payload":"Hello, I need the topology"}
Username: her21000
Received message from alb210041@alumchat.lol/web: {"type":"echo","from":"alb210041@alumchat.lol","to":"her21000@alumchat.lol","hops":0,"headers":{"request":"topology"},"payload":"Hello, I need the topology"}
Username: her21000
Enter the message: hey g6
Enter the destination: grupo6@alumchat.lol
Created message: {"type":"message","from":"her21000@alumchat.lol","to":"grupo6@alumchat.lol","hops":0,"headers":{},"payload":"hey g6 "}
Message: {"type":"message","from":"her21000@alumchat.lol","to":"grupo6@alumchat.lol","hops":0,"headers":{},"payload":"hey g6 "} sent to alb210041@alumchat.lol
Message sent ✓
>
```

```
PROBLEMAS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS
PS C:\Users\marka\Coding\UVG\Redes\Proyectos\Redes-L3> node .\linkState.js E alb210041@alumchat.lol 221756
---online
c: [ 'grupo6@alumchat.lol', 'her21000@alumchat.lol' ]
Requesting topology...
Created message: {"type":"echo","from":"alb210041@alumchat.lol","to":"grupo6@alumchat.lol","hops":0,"headers":{"request":"topology"},"payload":"Hello, I need the topology"}
Created message: {"type":"echo","from":"alb210041@alumchat.lol","to":"her21000@alumchat.lol","hops":0,"headers":{"request":"topology"},"payload":"Hello, I need the topology"}
Enter the message:
PS C:\Users\marka\Coding\UVG\Redes\Proyectos\Redes-L3> node .\flooding.js E alb210041@alumchat.lol 221756
---online
Enter the message: c: [ 'grupo6@alumchat.lol', 'her21000@alumchat.lol' ]
Received message from her21000@alumchat.lol/web: {"type":"message","from":"her21000@alumchat.lol","to":"grupo6@alumchat.lol","hops":0,"headers":{},"payload":"hey g6 "}
Username: alb210041
Message: {"type":"message","from":"her21000@alumchat.lol","to":"grupo6@alumchat.lol","hops":0,"headers":{},"payload":"hey g6 "} sent to grupo6@alumchat.lol
Message sent ✓
```

```
PS C:\Users\javie\OneDrive\Documents\GitHub\Redes-L3> node flooding.js B grupo6@alumchat.lol 1234
---online
Enter the message: c: [ 'alb210041@alumchat.lol' ]
Received message from alb210041@alumchat.lol/web: {"type":"echo","from":"alb210041@alumchat.lol","to":"grupo6@alumchat.lol","hops":0,"headers":{"request":"topology"},"payload":"Hello, I need the topology"}
Username: grupo6
Enter the message: Received message from alb210041@alumchat.lol/web: {"type":"echo","from":"alb210041@alumchat.lol","to":"grupo6@alumchat.lol","hops":0,"headers":{"request":"topology"},"payload":"Hello, I need the topology"}
Username: grupo6
Received message from alb210041@alumchat.lol/web: {"type":"message","from":"her21000@alumchat.lol","to":"grupo6@alumchat.lol","hops":0,"headers":{},"payload":"hey g6 "}
Username: grupo6
Enter the message:
```

### Prueba con flooding

```
{
  "type": "names",
  "config": {
    "A": "alb210041@alumchat.lol",
    "B": "grupo6@alumchat.lol",
    "E": "alb210041@alumchat.lol",
    "F": "her21000@alumchat.lol"
  }
}
{
  "type": "topo",
  "config": {
    "A": ["B"],
    "B": ["E"],
    "E": ["B", "F"],
    "F": ["E", "A"]
  }
}
```

### Configuración de flooding

```
PROBLEMAS  SALIDA  TERMINAL  PUERTOS  COMENTARIOS  CONSOLA DE DEPURACIÓN  node + - [ ] [ ] ... ^ x

'alb210041@alumchat.lol': [ 'grupo6@alumchat.lol', 'her21000@alumchat.lol' ],
'alb21004@alumchat.lol': [ 'grupo6@alumchat.lol' ],
'grupo6@alumchat.lol': [ 'alb210041@alumchat.lol' ]
}
Received echo message: [Object Object]
Created message: {"type":"info","from":"her21000@alumchat.lol","to":"alb21004@alumchat.lol","hops":0,"headers":{"request":"topology"},"payload":{"alb210041@alumchat.lol","alb21004@alumchat.lol"}}
Topology keys: {
  'alb210041@alumchat.lol',
  'alb21004@alumchat.lol',
  'grupo6@alumchat.lol'
}
Current topology: {
  'alb210041@alumchat.lol': [ 'grupo6@alumchat.lol', 'her21000@alumchat.lol' ],
  'alb21004@alumchat.lol': [ 'grupo6@alumchat.lol' ],
  'grupo6@alumchat.lol': [ 'alb210041@alumchat.lol' ]
}
Created message: {"type":"message","from":"alb210041@alumchat.lol","to":"alb21004@alumchat.lol","hops":1,"headers":{"alb210041@alumchat.lol","her21000@alumchat.lol","alb21004@alumchat.lol"},"payload":"hey 004"}
Current topology: {
  'alb210041@alumchat.lol': [ 'grupo6@alumchat.lol', 'her21000@alumchat.lol' ],
  'alb21004@alumchat.lol': [ 'grupo6@alumchat.lol' ],
  'grupo6@alumchat.lol': [ 'alb210041@alumchat.lol' ]
}
Enter the message:

hey 004
Enter the destination: alb21004@alumchat.lol
Created message: {"type":"message","from":"alb210041@alumchat.lol","to":"alb21004@alumchat.lol","hops":0,"headers":{"payload":"hey 004"}}
Link State Algorithm!!!
Weights: {
  'her21000@alumchat.lol': { 'alb210041@alumchat.lol': 1, 'alb21004@alumchat.lol': 1 },
  'grupo6@alumchat.lol': { 'alb210041@alumchat.lol': 1 },
  'alb21004@alumchat.lol': { 'grupo6@alumchat.lol': 1 },
  'alb210041@alumchat.lol': { 'grupo6@alumchat.lol': 1, 'her21000@alumchat.lol': 1 }
}
Distances: {
  'her21000@alumchat.lol': 1,
  'grupo6@alumchat.lol': 1,
  'alb21004@alumchat.lol': 2,
  'alb210041@alumchat.lol': 0
}
Predecessors: {
  'her21000@alumchat.lol': 'alb210041@alumchat.lol',
  'grupo6@alumchat.lol': 'alb210041@alumchat.lol',
  'alb21004@alumchat.lol': 'her21000@alumchat.lol',
  'alb210041@alumchat.lol': null
}
Path: [
  'alb210041@alumchat.lol',
  'her21000@alumchat.lol',
  'alb21004@alumchat.lol'
]
Message sent to: her21000@alumchat.lol
Enter the message:

}
Current topology: {
  'grupo6@alumchat.lol': [ 'alb210041@alumchat.lol' ],
  'alb210041@alumchat.lol': [ 'grupo6@alumchat.lol', 'her21000@alumchat.lol' ],
  'her21000@alumchat.lol': [ 'alb210041@alumchat.lol', 'alb21004@alumchat.lol' ]
}
Received message: {
  type: 'message',
  from: 'alb210041@alumchat.lol',
  to: 'alb21004@alumchat.lol',
  hops: 1,
  headers: [
    'alb210041@alumchat.lol',
    'her21000@alumchat.lol',
    'alb21004@alumchat.lol'
  ],
  payload: 'hey 004'
}
Enter the message: 
```

## Prueba con Link State

## Discusión

Para el algoritmo de flooding, el user her21000 envía un mensaje al user grupo6, se puede observar que ya que este contacto no tiene al grupo6 se envía el mensaje a los contactos disponibles de her21000 para que posteriormente envíen el mensaje a sus contactos hasta finalmente llegar al destinatario. Para reducir la redundancia y duplicación de mensajes se realizan comprobaciones antes de enviar y al recibir un mensaje. Antes de enviar se verifica si el destinatario indicado pertenece a la lista de contactos, si es así solo a ese contacto se le envía. Adicionalmente al recibir un mensaje se verifica que antes de reenviarlo no haya sido enviado por la misma cuenta actual.

En el caso del algoritmo de link state, se utilizó la configuración presentada [anteriormente](#). En este caso, se pretende buscar la ruta más corta entre el nodo E al nodo A para enviar un mensaje, para esto existen dos posibles rutas, la primera sería de E a B y regresar a E nuevamente, pero esto no sería lo más óptimo y la segunda, de E a F y luego a A, siendo esta la más corta, se crea el path y se envía el mensaje a F, que posteriormente lo reenvía a A.

Ruta más corta:  $E \Rightarrow F \Rightarrow A$

Ruta menos óptima:  $E \Rightarrow B \Rightarrow E \Rightarrow \dots \Rightarrow A$

Con esto se puede comprobar que gracias al uso de Dijkstra se logra encontrar la ruta más corta.

## Comentarios

Con la implementación de estos algoritmos se logró identificar que a pesar de que flooding es más sencillo, este puede ocasionar distintos problemas, esto debido al uso ineficiente del ancho de banda y en la duplicación de mensajes. Consideramos que el algoritmo de Link state es mejor ya que no tiene las debilidades mencionadas anteriormente. Este laboratorio nos permitió comprender ambos algoritmos a profundidad y las ventajas y desventajas que ambos pueden tener.

## Conclusiones

- Link State es un algoritmo que necesita una robusta sincronización entre clientes (nodos) para poder operar correctamente.
- El protocolo XMPP no es óptimo para enviar mensajes de “solicitudes”, ya que no es posible esperar directamente a la respuesta de un mensaje específico solicitando alguna información.
- Se determinó que la dificultad del algoritmo de link state es mayor pero es más eficiente al no presentar una duplicación de mensajes.

## Referencias

Tanenbaum, Andrew S.; Wetherall, David J. (March 23, 2010). Computer Networks (5th ed.). Pearson Education.

## Repositorio

<https://github.com/Kojimena/Redes-L3.git>