Laboratorio No. 3

Algortimos de enrutamiento

Marco Ramirez 21032
Josué Morales 21116
Alejandro Ortega 18248

Indice

Indice **1**
Descripción del laboratorio **2**
Algoritmos **3**
Flooding 3
Link State Routing 3
Implementación **4**
Código principal 4
Descripción 4
main.js: 4
xmppClient.js: 4
Flooding 5
Descripción 5
flooding.js: 5
Link State Routing 6
Descripción 6
lsr.js 6
Resultados **8**
Discusión **9**
Conclusiones **10**
Comentarios **11**
Referencias **12**

Descripción del laboratorio

En este laboratorio, nos enfocamos en la implementación y prueba de dos algoritmos de enrutamiento: Flooding y Link State Routing, utilizando JavaScript. Estos algoritmos nos sirven para comprender cómo se gestionan las rutas y el flujo de datos en redes complejas, como las que conforman el Internet. La implementación se llevó a cabo en un entorno simulado sobre el protocolo XMPP, permitiéndonos observar cómo los nodos interconectados intercambian información para construir tablas de enrutamiento eficientes. A través de las pruebas realizadas, pudimos analizar cómo cada algoritmo maneja la propagación de mensajes y la actualización dinámica de las rutas en respuesta a cambios en la topología de la red. Esta práctica nos permitió tener conocimientos en redes.

Algoritmos

Flooding

El algoritmo de flooding es una técnica de enrutamiento utilizada en redes de computadoras, donde cada paquete entrante es reenviado a todas las salidas posibles excepto aquella por la cual llegó. Este método no requiere de un conocimiento previo de la topología de la red, lo que lo convierte en un enfoque no adaptativo. Flooding garantiza que los paquetes de datos eventualmente lleguen a todos los nodos dentro de la red, distribuyendo así la información de manera rápida (Terrell, 2023).

Link State Routing

El algoritmo de link state routing permite que cada nodo tenga una visión completa de la red. Esto se consigue mediante el intercambio regular de información sobre el estado de los enlaces entre los nodos. Con esa información, cada nodo aplica el algoritmo de Dijkstra para calcular las rutas más cortas hacia todos los demás nodos. Aunque este método es más eficiente en el uso de recursos comparado con Flooding, también implica un procesamiento más demandante en cada nodo, debido a la necesidad de mantener una base de datos actualizada y realizar cálculos más complejos (Computer Network | Link State Routing Algorithm - Javatpoint, s. f.).

Implementación

Código principal

Descripción

main.js:

- **Función principal**: Coordina la ejecución del nodo XMPP basado en el algoritmo seleccionado (Flooding o Link State Routing).
- **showMenu()**: Muestra un menú interactivo para el usuario con opciones para enviar mensajes, mostrar la topología de la red, ver la tabla de enrutamiento, o salir.
- **getUserCredentials()**: Solicita al usuario su JID y contraseña para conectarse al servidor XMPP.
- **main()**: Controla el flujo del programa, seleccionando y ejecutando el algoritmo especificado.

xmppClient.js:

- **initializeXmppClient(jid, password)**: Inicializa y configura un cliente XMPP utilizando las credenciales proporcionadas.
- **Eventos**: Maneja los eventos de conexión (online), recepción de mensajes (stanza), y errores (error).

Flooding

El algoritmo de flooding fue implementado en el protocolo XMPP para permitir la difusión de mensajes a través de una red de nodos. La implementación consiste en enviar un mensaje a todos los vecinos de un nodo, excluyendo al remitente inmediato, para garantizar que el mensaje llegue a su destino. Cada nodo en la red verifica si ya ha recibido el mensaje anteriormente para evitar la retransmisión redundante. En esta implementación, se utiliza XMPP para manejar la transmisión de mensajes entre nodos, donde cada nodo se conecta al servidor XMPP, recibe mensajes, y los reenvía a sus vecinos según las reglas del algoritmo de flooding.

Descripción

flooding.js:

- **FloodingAlgorithm**: Clase que implementa el algoritmo de flooding.
- **processIncomingStanza(stanza)**: Procesa cada mensaje recibido y lo maneja según las reglas del flooding.
- **processMessage(message, sender)**: Verifica si el mensaje ya fue procesado; si no, lo reenvía a los vecinos.
- **forwardMessage(message, sender)**: Reenvía el mensaje a todos los vecinos excepto al remitente.
- **sendXmppMessage(message, recipient)**: Envía un mensaje XMPP a un destinatario específico.
- **sendInitialChatMessage(recipient, content)**: Crea y envía un mensaje inicial.
- **logShortestRoute()**: Registra la ruta más corta encontrada para la entrega de un mensaje.

Link State Routing

El algoritmo de Link State Routing fue implementado en el protocolo XMPP para permitir que los nodos intercambien información sobre su estado de enlace y construyan tablas de enrutamiento óptimas para la transmisión de mensajes a través de la red. Cada nodo en la red comparte su estado de enlace con sus vecinos, lo que permite a todos los nodos construir una visión global de la red. Este enfoque facilita el cálculo de las rutas más cortas utilizando el algoritmo de Dijkstra. La implementación maneja la recepción y propagación de mensajes de estado de enlace, la actualización de la base de datos de estado de enlace, y el reenvío de mensajes según la tabla de enrutamiento calculada.

Descripción

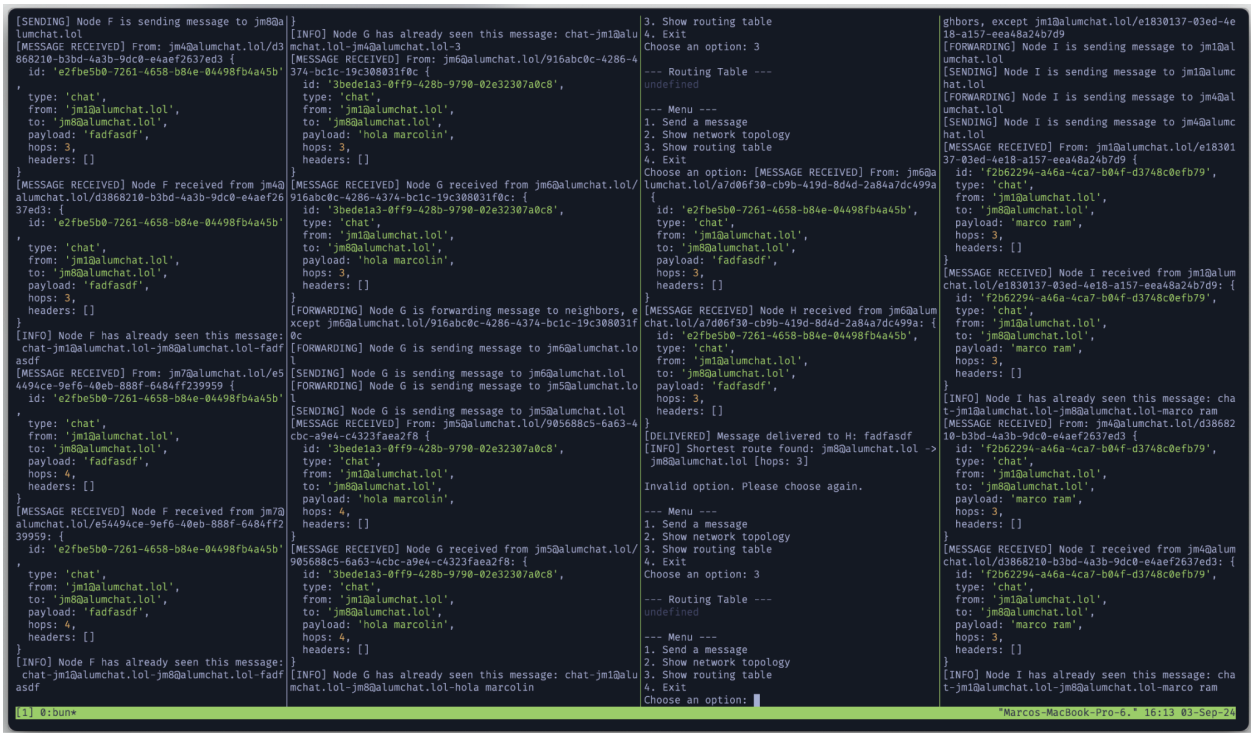
lsr.js

- **LinkStateRouting**: Clase que implementa el algoritmo de Link State Routing.
- **handleStanza(stanza)**: Maneja la recepción de mensajes y los procesa según su tipo.
- **handleMessage(message, from)**: Procesa el mensaje recibido y decide si debe ser reenviado o si es para entrega final.
- **shareLinkState()**: Envía el estado de enlace actual a todos los vecinos del nodo.
- **floodMessage(message, from)**: Propaga el mensaje de estado de enlace a todos los vecinos, excluyendo al remitente inmediato.
- **updateLinkStateDB(sourceJid, costs)**: Actualiza la base de datos de estado de enlace con la información recibida de otro nodo.
- **sendMessage(message, recipient)**: Envía un mensaje XMPP a un destinatario específico.

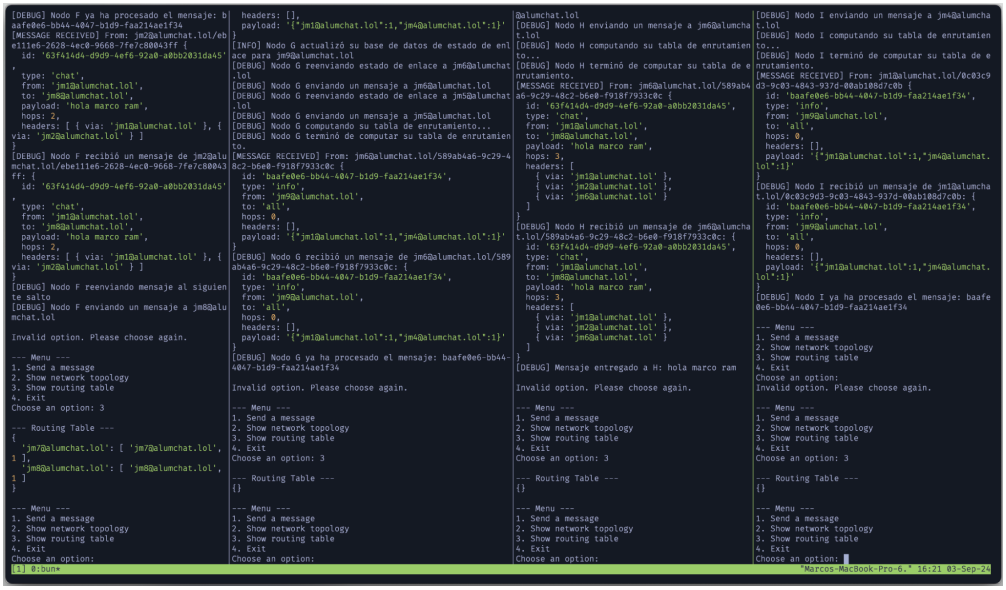
- **computeRoutingTable():** Calcula la tabla de enrutamiento del nodo utilizando el algoritmo de Dijkstra, basada en la base de datos de estado de enlace.
- **forwardMessage(message)**: Reenvía un mensaje al siguiente nodo en la ruta calculada hacia el destino final.
- **getNextHop(destination)**: Obtiene el siguiente nodo en la ruta hacia el destino especificado.

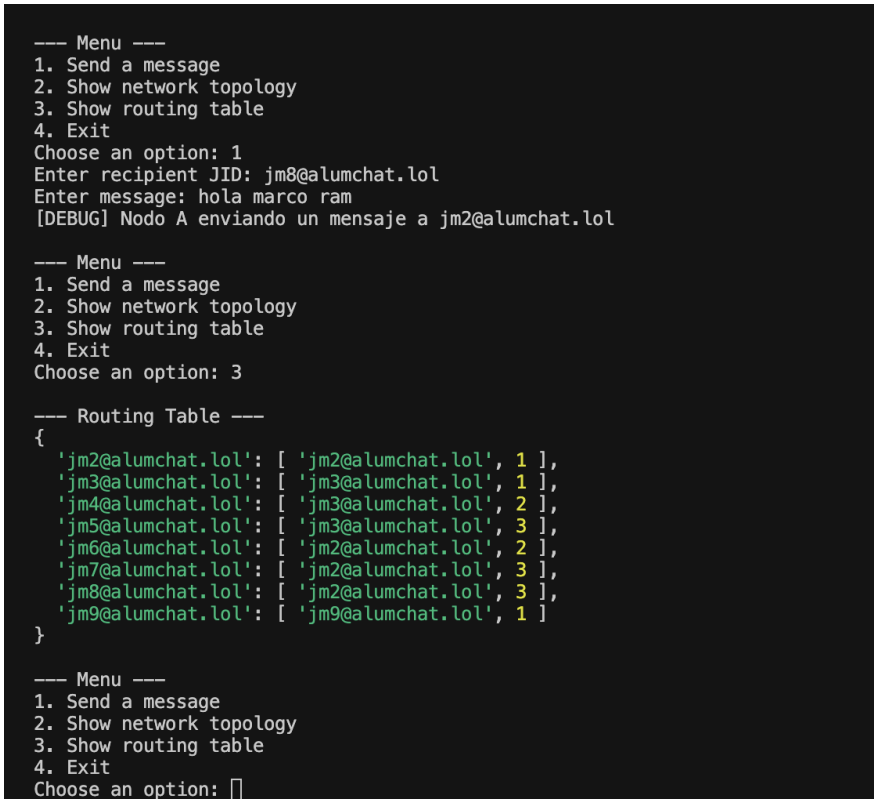
Resultados

Flooding



Link State Routing





Discusión

Al realizar las pruebas utilizando los algoritmos de Flooding y LSR, se observaron comportamientos distintos que reflejan las diferencias fundamentales entre ambos métodos de enrutamiento.

En el caso de Flooding, el mensaje se reenvía a todos los nodos vecinos sin importar cual sea, lo que resulta en muchos mensajes redundantes dentro de la misma red. Esto significa que aunque llega a su destino, la carga en la red aumenta debido a la duplicación de mensajes, especialmente cuando el mensaje debe pasar por muchos nodos. En redes pequeñas o con topologías simples, este método puede funcionar de manera aceptable, pero su eficiencia disminuye drásticamente en redes grandes o complejas debido al sobrecargado de la red.

Por otro lado, el algoritmo de LSR optimiza el proceso de enrutamiento al permitir que cada nodo mantenga una base de datos de la topología de la red, actualizada mediante el intercambio de información de estado de enlace. Esto permite a cada nodo calcular la ruta más eficiente hacia el destino utilizando el algoritmo de Dijkstra, lo que resulta en un uso más eficiente de los recursos de la red y en la eliminación de la redundancia que se produce al usar Flooding. Como resultado, LSR no solo minimiza la cantidad de mensajes transmitidos, sino que también garantiza que los datos lleguen de manera más directa y eficiente a su destino.

Flooding puede ser útil en situaciones muy específicas y en redes pequeñas donde no se toma en cuenta los recursos de la red, LSR es muy superior en términos de eficiencia y escalabilidad. Flooding puede generar un tráfico muy pesado, lo que puede llevar a retrasos innecesarios y una red congestionada, mientras que LSR permite que los los recursos de la red se usen de manera

más inteligente, asegurando un mayor rendimiento incluso en redes de mayor tamaño y complejidad.

Conclusiones

- Flooding funciona bien cuando no es prioridad la eficiencia, así como cuando se puede aceptar cierto nivel de redundancia, sin embargo en redes grandes genera demasiado tráfico en la red y se vuelve poco viable, por el tiempo de espera que produce.
- LSR es más eficiente y optimiza el uso de recursos de la red, lo cual es ideal para redes grandes y dinámicas.
- Flooding es fácil de implementar y se muestra fuerte ante fallos, pero su mayor desventaja es el exceso de tráfico que produce.
- LSR, a pesar de ser más complicado de mantener, se adapta mejor a redes grandes y ofrece un rendimiento superior.

Comentarios

- Las pruebas permitieron comparar directamente el comportamiento de Flooding y LSR en un entorno controlado, destacando cómo la elección del algoritmo impacta tanto en el rendimiento como en la eficiencia de la red.
- A pesar de la complejidad, LSR demostró ser más adaptable a redes dinámicas y de mayor escala, lo que lo convierte en una mejor opción para infraestructuras modernas que demandan alta eficiencia y rendimiento.
- El laboratorio brindó una comprensión más detallada de cómo la complejidad y el esfuerzo de mantenimiento se equilibran con el rendimiento y la escalabilidad en el diseño de redes. Esto es clave para tomar decisiones al momento de tener que elegir un algoritmo de enrutamiento.

Referencias

Computer Network | Link State Routing Algorithm - javatpoint. (s. f.). www.javatpoint.com. https://www.javatpoint.com/link-state-routing-algorithm

Terrell, H (2023). What is network flooding?. TechTarget. [¿Qué es la inundación de la red? TechTarget.] https://www.techtarget.com/whatis/definition/flooding-network