Universidad del Valle de Guatemala
Redes
Ciclo II, 2024
Fecha de entrega: 29 ago 2024

Adrián Ricardo Flores Trujillo	21500
Andrea Ximena Ramírez Recinos	21874

Laboratorio 3 y 4

Algoritmos de Enrutamiento

Descripción de la Práctica

Esta práctica consistió en el desarrollo de un cliente XMPP que implementa algoritmos de enrutamiento para el envío óptimo de mensajes en una red. Al ser la única pareja para el laboratorio, solo se implementó un cliente utilizando Link-State Routing, el cual emplea el algoritmo de Flooding para garantizar que todos los clientes tengan el mismo conocimiento de la red. A través de estos algoritmos, se busca enviar mensajes entre clientes, asegurando que el mensaje tome la ruta más eficiente hacia su destino. Para fines de la práctica, se utilizó el servidor alumchat.lol como conmutador entre los clientes, y el cliente fue hecho en JavaScript y XMPP.js.

Algoritmos Utilizados

- **Flooding:**
  Este es un algoritmo de distribución de mensajes en una red que se logra reenviando cada paquete a todos los vecinos conocidos, exceptuando el nodo que envió el mensaje. Este proceso puede generar una cantidad excesiva de paquetes en la red, por lo que es crucial implementar alguna medida que asegure la terminación del algoritmo. Usualmente, esto se consigue mediante un tiempo de vida (TTL) o un límite máximo de saltos para cada paquete. Sin embargo, para los fines de este laboratorio, cada paquete registraba los nodos por los que ya había pasado, evitando así que fuera reenviado a un nodo que ya lo hubiera recibido (Geeks4Geeks[1], 2023).

  Aunque el cliente no cuenta con un modo que utilice únicamente Flooding, este algoritmo se implementó como parte de la estrategia general de Link-State Routing. Al inicio del ciclo de vida de cada cliente, se revisan las conexiones directas del nodo según la topología de la red. Al identificar a sus vecinos, se envían mensajes tipo "ECHO" para medir el tiempo de ida y vuelta de los paquetes. Conociendo el tiempo de respuesta de cada vecino, se envían todos los tiempos conocidos de la red a los demás vecinos. Estos registran su nombre en el paquete y lo reenvían a sus vecinos, asegurando que la información se propague sin generar un ciclo interminable de mensajes. Eventualmente, todos los clientes en la red alcanzan un consenso sobre la topología de la red y los tiempos de respuesta entre cada nodo.

- **Link-State Routing:**
  Consiste en una familia de algoritmos de protocolos de enrutamiento cuyo objetivo es intercambiar mensajes con otros clientes para que cada nodo en una red conozca la topología completa de la misma. Este algoritmo se divide en fases: primero, se comienza conociendo la información sobre los vecinos inmediatos del cliente. Una vez conocidos los vecinos, se ejecuta el algoritmo de Flooding para compartir esta información con todos los nodos de la red. Cuando todos los nodos han llegado a un consenso sobre la topología de la red, se procede a calcular las rutas óptimas hacia cada nodo, generalmente utilizando el

algoritmo de Dijkstra. Este cálculo es realizado de manera individual por cada cliente, y sus resultados no se comparten, ya que el objetivo es determinar la mejor manera de enrutar los mensajes desde el nodo actual (Geeks4Geeks[2], 2023).

En este laboratorio, después de ejecutar el proceso de Flooding, se construye la tabla de enrutamiento en el cliente, basándose en la información obtenida de los demás clientes. Al aplicar el algoritmo de Dijkstra, se obtiene un diccionario que indica, para cada nodo en la red, el siguiente nodo al que se debe enviar el mensaje para garantizar la ruta óptima.

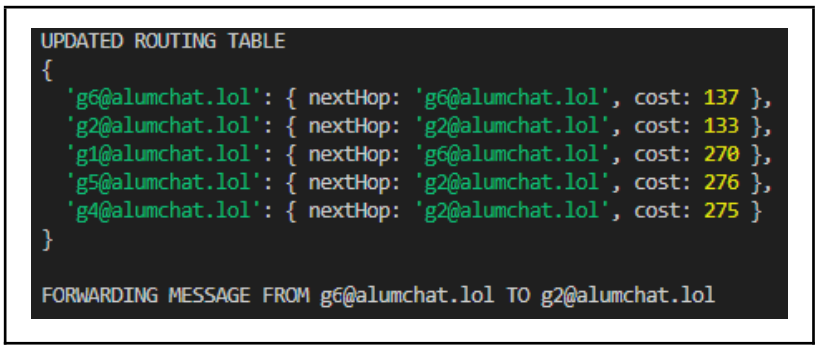

Figura 1. Ejemplo de tabla de enrutamiento, con costos medidos en milisegundos.

Resultados

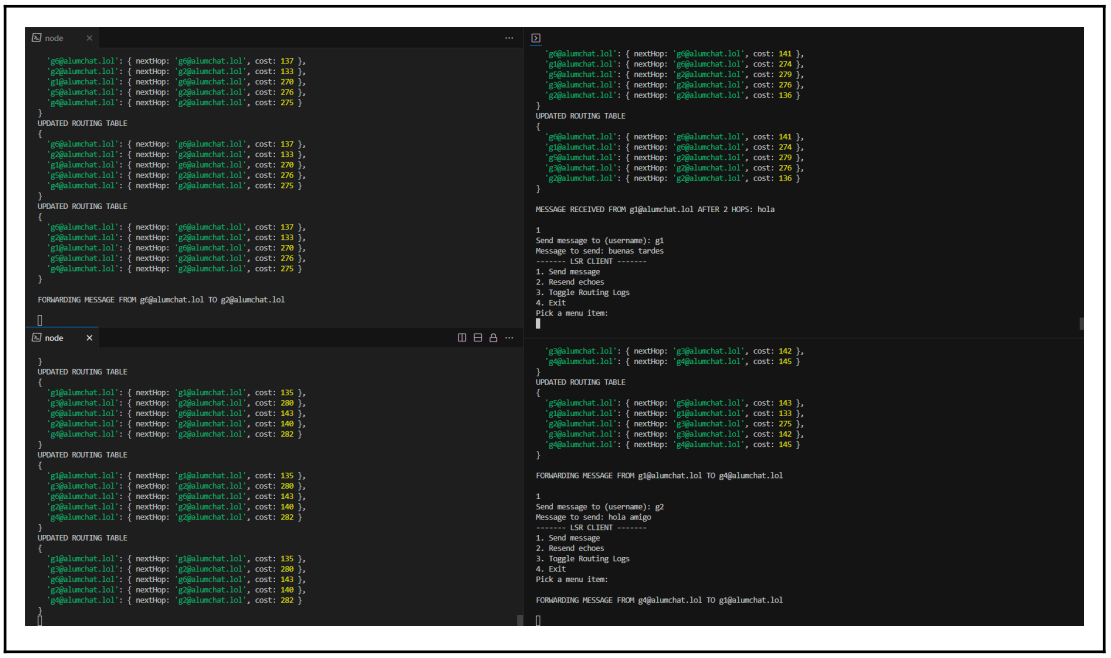

Figura 2. Cuatro clientes en la misma red corriendo simultáneamente.

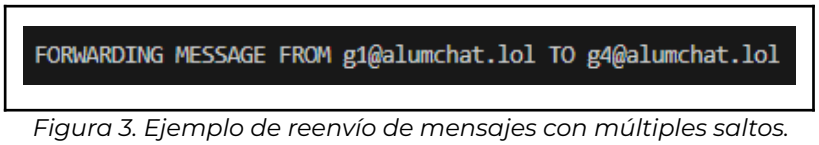

Figura 3. Ejemplo de reenvío de mensajes con múltiples saltos.

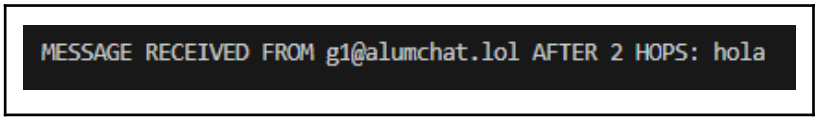

Figura 4. Ejemplo de recepción de mensajes con múltiples saltos.

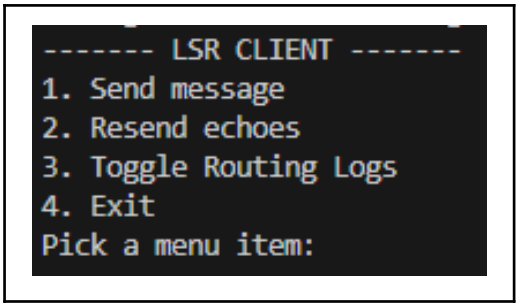

Figura 5. Menú de acciones que puede tomar el usuario.

Discusión

Para comenzar, es fundamental destacar la importancia de contar con un protocolo de comunicación acordado entre todos los nodos de la red. La implementación de los algoritmos de enrutamiento depende del envío simultáneo de una gran cantidad de paquetes, que deben interpretarse correctamente para garantizar que todos los nodos de la red tengan el mismo conocimiento sobre la topología. Además, dado el gran volumen de paquetes entrantes y salientes, es crucial que el cliente esté optimizado adecuadamente. Los resultados de los algoritmos dependen del tiempo de respuesta de cada nodo, y si uno de ellos se satura y tarda en responder, es probable que no se considere a la hora de enrutar mensajes debido a sus altos costos.

Por otro lado, es importante considerar la dinámica de la red, en la cual varios nodos pueden desconectarse y reconectarse sin previo aviso. Por esta razón, es necesario contemplar estos casos y establecer un mecanismo para "poner al día" a los nodos cuando se reconecten. Existe la posibilidad de que esto no sea un problema para algunos clientes, ya que, si se reenvía continuamente el conocimiento de toda la red, no es necesario considerar este caso específico. Sin embargo, para evitar volúmenes excesivos de paquetes, las mediciones de costos solo se realizan una vez durante el ciclo de vida del cliente, a menos que el usuario desee recalcular estos tiempos. Además, para garantizar que todos los clientes conozcan la topología, incluso si se conectan tarde, se implementó un método para responder a ‘ECHO’ de tal manera que se reciban dos respuestas: una que devuelva el ‘ECHO’ y otra que envíe el conocimiento actual del nodo, que incluye un encabezado especial ("no-flood") para evitar que este paquete específico se reenvíe.

Como resultado, se considera que el cliente es relativamente efectivo, ya que se evita reenviar mensajes innecesariamente, a cambio de períodos en los que la red no cuenta con conocimientos constantemente actualizados. Por otro lado, no se implementó la capacidad de reconocer cuando un cliente cae, debido a las limitaciones mencionadas anteriormente, lo que impide actualizar la tabla de enrutamiento en esos casos.

Conclusiones

- Es importante priorizar los bajos tiempos de respuesta en este tipo de clientes gracias al volúmen de acciones que deben realizar en periodos de tiempo cortos.
- Las redes dinámicas requieren de algoritmos de enrutamiento capaces de tolerar caídas de nodos o tiempos de respuesta elevados.
- Si los cambios en una red no se detectan y manejan de manera adecuada, es posible incurrir en pérdida de paquetes y errores de enrutamiento de mensajes.

Comentarios

- Es sumamente complicado llegar a un consenso sobre el formato de comunicación en este tipo de trabajos, especialmente durante las etapas tempranas de desarrollo.

- Consideramos que es un tanto redundante tratar el algoritmo de Flooding como un proceso independiente, ya que el Link-State Routing depende de él, y este último es obligatorio.

- Hubo casos en los que, de manera inesperada, algunos clientes recibieron paquetes muy antiguos del servidor, a pesar de que se habían reiniciado todos los nodos y no se estaba tomando ninguna acción. Estas situaciones no afectaron el desarrollo del cliente, pero es importante tenerlas en cuenta, ya que los clientes estaban recibiendo información que no era posible conocer en ciertos momentos.

Referencias

[1] Geeks4Geeks. (2023). *Fixed and Flooding Routing Algorithms*. Recuperado de: https://www.geeksforgeeks.org/fixed-and-flooding-routing-algorithms/

[2] Geeks4Geeks. (2023). *Unicast Routing - Link State Routing*. Recuperado de: https://www.geeksforgeeks.org/unicast-routing-link-state-routing/