CC3067 - Redes
Adrian Rodriguez - 21691
Brian Carrillo - 21108
Carlos López - 21666

UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

Laboratorio 3 - Algoritmos de Enrutamiento

Descripción de la práctica

La práctica consistió en comprender y aplicar los algoritmos utilizados para actualizar las tablas de enrutamiento en redes. Los tres algoritmos de interés fueron: *Flooding y Link State Routing*

El desarrollo del laboratorio se dividió en dos partes principales. La primera consistió en la implementación de los algoritmos, donde cada integrante del grupo programó uno de ellos. La segunda parte implicó la interconexión y prueba de estos algoritmos utilizando el protocolo XMPP y el servidor alumchat.lol (provisto por el profesor).

Los algoritmos se probaron en una red simulada donde cada nodo correspondía a un usuario en el servidor XMPP. Se diseñó un protocolo de comunicación basado en JSON para el intercambio de mensajes entre nodos. El objetivo fue lograr que los algoritmos se estabilizaran y los mensajes siguieran rutas óptimas, adaptándose a cambios en la topología de la red.

Algoritmos utilizados

El desarrollo del laboratorio que implementó y analizó los algoritmos de enrutamiento Flooding y Link State Routing (LSR).

El algoritmo de Flooding se caracteriza por enviar todos los paquetes entrantes a cada interfaz de salida, excepto por aquella de la que provienen. Esto asegura que el paquete llegue a su destino si existe una ruta viable, aunque puede resultar en un tráfico excesivo en la red. Este método es fácil de implementar, pero su ineficiencia puede degradar el rendimiento de la red debido al uso excesivo de ancho de banda (Wikipedia, s.f.). Para mitigar el problema de la propagación infinita de mensajes, se pueden implementar contadores de saltos que limitan cuántas veces un paquete puede ser retransmitido (Alzahrani & Alshahrani, 2020).

El algoritmo de Link State Routing, por otro lado, permite que cada nodo mantenga un conocimiento completo de la topología de la red. Esto se logra mediante el intercambio periódico de información sobre el estado de los enlaces entre nodos. Con esta información, cada nodo utiliza el algoritmo de Dijkstra para calcular las rutas más cortas hacia todos los demás nodos, lo que lo hace más eficiente en términos de uso de recursos en comparación con Flooding. Sin embargo, requiere más procesamiento en cada nodo debido a la necesidad de mantener una base de datos de estado de enlaces y realizar cálculos más complejos (Pezoa Núñez, 2001).

Implementación

Algoritmo Flooding

Clase FloodingNode:

- Propiedades: Cada instancia de esta clase representa un nodo en la red, que tiene un identificador único (nodeId), una contraseña para autenticarse, una lista de vecinos a los que puede enviar mensajes, y un conjunto de mensajes ya vistos para evitar enviar el mismo mensaje múltiples veces.
- Métodos:
    - loadConfigurations(): Carga la configuración del nodo y de sus vecinos desde archivos externos y valida la configuración.
    - setupXMPP(): Configura la conexión XMPP usando las credenciales del nodo.
    - onConnect(): Se llama cuando el nodo se conecta al servidor XMPP y envía una presencia para indicar que está en línea.
    - sendMessage(): Envía un mensaje a un vecino específico usando XMPP.
    - onStanza(): Maneja los mensajes entrantes y decide si deben ser procesados o reenviados.
    - handleMessage(), handleHello(), handleChatMessage(), floodMessage(): Métodos para manejar diferentes tipos de mensajes y reenviarlos a los vecinos según sea necesario.

Inicialización de Nodos:

- Función initializeNodesSequentially(): Inicializa los nodos en secuencia según la configuración proporcionada, asegurando que cada nodo esté en línea antes de pasar al siguiente.
- Función initializeNode(): Inicializa un nodo específico basado en el nodeId ingresado por el usuario y permite que el usuario actúe como emisor o receptor en la red.
- Función waitForNodeSelection(): Pregunta al usuario qué nodo desea inicializar y gestiona la entrada del usuario.

Algoritmo Link State Routing

Clase LSRNode:

- Propiedades:
    - Cada instancia de LSRNode representa un nodo en la red con su JID (Jabber ID), contraseña, vecinos, tabla de enrutamiento, base de datos de estado de enlace, y otros parámetros necesarios para el funcionamiento del algoritmo LSR.
    - linkStateDB: Almacena el estado de enlace, que incluye los costos de comunicación entre nodos.
    - routingTable: Almacena las rutas óptimas calculadas hacia otros nodos.
    - receivedMessages: Registra los mensajes recibidos para evitar procesar duplicados.
    - verbose: Un modo detallado que permite registrar mensajes adicionales para depuración.
- Métodos:
    - start(): Inicia la conexión XMPP.
    - onOnline(): Se ejecuta cuando el nodo se conecta correctamente al servidor XMPP y envía su presencia.
    - sendMessageTo(): Envía un mensaje al siguiente salto en la ruta hacia el destinatario final.
    - onStanza(): Procesa los mensajes entrantes, manejando tanto mensajes de información como mensajes normales.
    - floodMessage(): Propaga el estado del enlace a través de la red, enviando mensajes a todos los vecinos excepto al remitente.
    - shareLinkState(): Distribuye el estado actual del enlace a todos los vecinos, iniciando la propagación del estado de enlace.
    - computeRoutingTable(): Calcula la tabla de enrutamiento utilizando el algoritmo de Dijkstra, basándose en los estados de enlace conocidos.
    - logNetworkState(): Muestra el estado actual de la red, incluyendo la tabla de enrutamiento y la base de datos de estado de enlace.
    - onError(): Maneja errores durante la operación de la red.

Inicialización de Nodos:

- Función initializeNodesSequentially(): Inicializa los nodos de manera secuencial según la configuración proporcionada, conectándolos al servidor XMPP y estableciendo las tablas de enrutamiento iniciales.

- Función initializeNode(): Inicializa un nodo específico según la configuración elegida por el usuario, permitiendo al usuario interactuar como emisor o receptor de mensajes en la red.
- Función waitForNodeSelection(): Gestiona la selección del nodo a inicializar, preguntando al usuario a través de la línea de comandos.

Resultados

Flooding

Tras la implementación de los algoritmos de enrutamiento Flooding y Link State Routing (LSR) en una red simulada utilizando el protocolo XMPP, se llevaron a cabo varias pruebas para evaluar el rendimiento y la eficacia de cada algoritmo.

Figura 1. Topología utilizada en las pruebas de tiempo

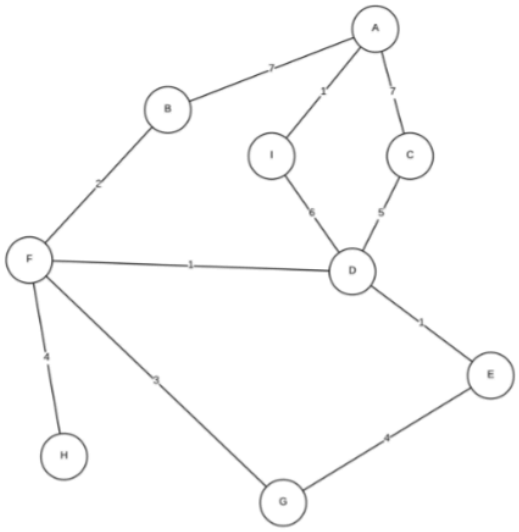

Figura 2. Logs de las pruebas realizadas con algoritmo Flooding.

Tabla 1. Resultados de tiempo para algoritmo Flooding.

Emisor/Receptor	Tiempo (ms)
C a I	166
B a G	170
A a H	262

Link State Routing Algorithm

Figura 3. Logs de las pruebas realizadas con algoritmo LSR.

Tabla 2. Resultados de tiempo para algoritmo LSR.

Emisor/Receptor	Tiempo (ms)
C a I	76
B a G	91
A a H	77

Discusión

En las pruebas realizadas con el algoritmo de Flooding, observamos que los tiempos de entrega de los mensajes están directamente relacionados con la cantidad de saltos que un mensaje debe hacer para llegar a su destino. Por ejemplo, el mensaje enviado desde el Nodo C al Nodo I, que le llevó solo un par de saltos, tardó 165.754 ms, mientras que el mensaje de Nodo A a Nodo H, que atravesó tres nodos intermedios, tardó 261.652 ms. Este comportamiento es como se esperaba, ya que cada salto adicional en el camino introduce complejidad que aumenta el tiempo total de entrega del mensaje.

Además, los tiempos de entrega fueron menores cuando el mensaje se envió entre nodos que están directamente conectados, es decir, cuando los nodos de origen y destino son vecinos directos en la topología de la red. Este fue el caso de los mensajes de C a I y de B a G, que registraron tiempos de 165.754 ms y 170.338 ms, respectivamente. Esto sugiere que en configuraciones de red donde los nodos están cercanos o directamente conectados, el algoritmo de flooding puede operar de manera relativamente eficiente. Sin embargo, cuando el mensaje debe atravesar varios nodos para llegar a su destino, como en el caso del mensaje de A a H, los tiempos de entrega aumentan significativamente.

Esto refleja una limitación clave del algoritmo de flooding, que es la redundancia en el reenvío de mensajes a todos los nodos vecinos introduce una carga adicional en la red, lo que puede resultar en tiempos de entrega más largos, especialmente en redes más grandes o complejas. En redes pequeñas o simples, el flooding puede ser lo suficientemente eficiente, pero su escalabilidad se ve comprometida a medida que el tamaño de la red y la cantidad de conexiones crecen, lo que sugiere que este método es más adecuado para redes donde la simplicidad es más importante que la eficiencia óptima.

Ahora bien, con el algoritmo de LSR, el mensaje enviado desde el Nodo C al Nodo I, que atravesó dos saltos, tardó 76 ms, mientras que el mensaje desde el Nodo A al Nodo H, con tres saltos, tomó 77 ms. Estos tiempos son significativamente menores en comparación con los tiempos registrados con el algoritmo de flooding, donde los mismos mensajes tomaron hasta 165.754 ms y 261.652 ms, respectivamente, debido a la redundancia y el mayor número de reenvíos involucrados en flooding.

El LSR selecciona rutas óptimas basadas en la información del estado de los enlaces en toda la red, lo que resulta en caminos más directos y eficientes. Además, los tiempos de entrega para conexiones cercanas, como los mensajes de C a I (76 ms) y de B a G (91 ms), muestran que el LSR mantiene una alta eficiencia incluso cuando los nodos están directamente conectados o cercanos.

En comparación con flooding, el LSR no solo proporciona tiempos de entrega más rápidos, sino también una mayor eficiencia en el uso de los recursos de la red. Mientras que el flooding puede ser útil en redes pequeñas y simples, su escalabilidad es limitada y los tiempos de entrega se vuelven insostenibles en redes más grandes. Por otro lado, LSR optimiza cada ruta de manera dinámica, reduciendo la latencia y evitando la redundancia innecesaria, lo que se traduce en tiempos de entrega significativamente mejores y un mejor rendimiento.

Conclusiones

- En conclusión, la implementación y pruebas de los algoritmos de enrutamiento Flooding y LSR demostraron la importancia de seleccionar el algoritmo adecuado según el contexto de la red.
- El algoritmo de Flooding, es ideal para aplicaciones donde la simplicidad es una característica importante y la redundancia es tolerable, pero no es adecuado para redes grandes debido al tráfico excesivo que genera.
- El algoritmo de LSR, es adecuado para redes donde la eficiencia y la optimización de recursos son críticas.
- El algoritmo de LSR es menos complejo y más sencillo de implementar, mientras que el algoritmo LSR tiene mayor complejidad en la implementación y mantenimiento.
- El algoritmo de Flooding es robusto ante fallos en la red, ya que asegura que los mensajes lleguen a todos los nodos, pero este enfoque conlleva un alto costo en términos de tráfico y eficiencia.
- El algoritmo de LSR es más escalable y adecuado para redes dinámicas y complejas, donde el uso de los recursos y la optimización de las rutas son fundamentales para mantener un rendimiento óptimo.

Comentarios

- Las pruebas realizadas en el laboratorio con los algoritmos de enrutamiento Flooding y LSR han proporcionado una visión clara de las ventajas y desventajas de cada enfoque, lo que resulta fundamental para entender cómo elegir el algoritmo adecuado en diferentes escenarios de red.
- Al trabajar con Flooding y LSR, se hizo evidente cómo las decisiones de diseño pueden influir en la eficiencia y escalabilidad de una red, lo cual es crucial para aplicar estos algoritmos en situaciones reales
- En el laboratorio también pudimos ver el grado de complejidad de la implementación de algoritmos más avanzados como LSR. Aunque ofrece un rendimiento superior, su implementación y mantenimiento requieren un mayor esfuerzo y recursos.

Referencias

Wikipedia. (s.f.). *Flooding algorithm*. En Wikipedia. Recuperado el 28 de agosto de 2024, de https://en.wikipedia.org/wiki/Flooding_algorithm

Alzahrani, A., & Alshahrani, S. (2020). Flood and contain: An optimized repeal-based flooding algorithm for resource discovery in wireless ad hoc networks. *Journal of Network and Computer Applications*, 168, 102756. https://doi.org/10.1016/j.jnca.2020.102756

Pezoa Núñez, J. E. (2001). *Redes de datos*. Universidad de Concepción, Facultad de Ingeniería, Departamento de Ingeniería Eléctrica.