UNIVERSIDAD DEL VALLE DE GUATEMALA

Redes

Sección 20

“Laboratorio 3 Parte 2”

ESTEBAN ALDANA GUERRA 20591
JESSICA PAMELA ORTIZ IXCOT 20192

GUATEMALA, Septiembre 2023

Descripción de la Práctica

El objetivo principal de este laboratorio es comprender y aplicar los conceptos fundamentales relacionados con los algoritmos de enrutamiento de Internet. Para ello, se implementan y simulan diferentes algoritmos utilizando una herramienta basada en Python. Estos algoritmos requieren una comunicación efectiva entre ellos para su correcto funcionamiento. Para lograr esta comunicación, se ha incorporado el protocolo XMPP a través del servidor alumchat.xyz, permitiendo a los algoritmos intercambiar información de manera eficiente y en tiempo real.

Los principales algoritmos implementados en este laboratorio son: Flooding, Distance Vector y Link State Routing.

Flooding: Es un algoritmo simple de enrutamiento donde todos los paquetes entrantes son enviados por cada interfaz de salida, excepto por la que fueron recibidos. Esta naturaleza garantiza que un paquete será entregado, siempre que sea posible. A pesar de su facilidad de implementación, su enfoque puede considerarse bruto e ineficiente.

Distance Vector: En este algoritmo, cada nodo intercambia tablas de enrutamiento con sus vecinos y actualiza sus tablas según la información recibida. Es uno de los algoritmos de enrutamiento más significativos y utiliza el algoritmo de Bellman-Ford para determinar las rutas.

Link State: Aquí, cada nodo es consciente del estado de sus propios enlaces y comparte esta información con todos los demás nodos en la red. Los protocolos link-state utilizan el algoritmo SPF (Shortest Path First) creado por Edsger Dijkstra para calcular la mejor ruta basada en el menor costo acumulado a lo largo del trayecto.

Implementación / Resultados

Para la implementación de estos tres algoritmos, se aprovechó la herramienta de Python mencionada. A través del protocolo XMPP y el servidor alumchat.xyz, los algoritmos pueden comunicarse entre sí, intercambiando información esencial para su operación. Esta incorporación no solo mejora la eficiencia de la comunicación entre los algoritmos sino que también abre nuevas posibilidades de simulación y experimentación en el laboratorio.

Flooding

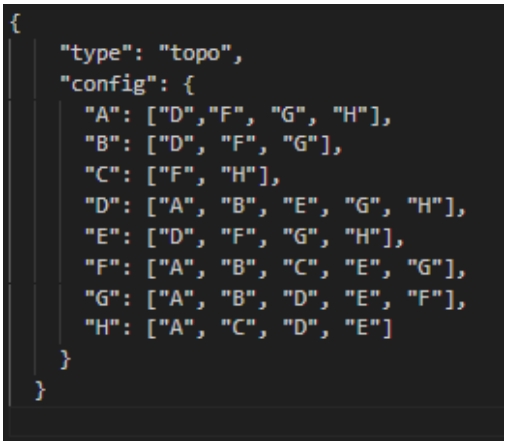
Se implementó una simulación del algoritmo de flooding utilizando un enfoque basado en la comunicación XMPP. Cada terminal actúa como un nodo individual en una topología predefinida. Cuando se envía un mensaje desde un nodo, este se propaga a través de la topología definida hasta que llega a su destinatario.

Para visualizar y manejar este proceso, se desarrolló una interfaz que permite a los usuarios iniciar una sesión de chat, crear una nueva cuenta o salir de la aplicación. Una vez que un usuario inicia una sesión de chat, tiene la opción de enviar un mensaje o salir de la aplicación.

El sistema fue diseñado utilizando el módulo slixmpp. El proceso de propagación se maneja en la clase Flooding, que hereda de slixmpp.ClientXMPP. Esta clase tiene métodos para manejar eventos de inicio de sesión, enviar mensajes y recibir mensajes. La función flood_send es responsable de emular el proceso de flooding al propagar un mensaje a través de la topología. Los mensajes se envían en formato JSON e incluyen información como el nodo fuente, el nodo destino, el número de saltos, la distancia recorrida y los nodos visitados.

Cuando un nodo recibe un mensaje, verifica si es el destinatario final. Si no lo es, el mensaje se reenvía a los nodos vecinos según la topología definida. Además, para evitar la retransmisión infinita, se lleva un registro de los mensajes ya enviados. Si se detecta un mensaje que ya ha sido enviado anteriormente, se descarta.

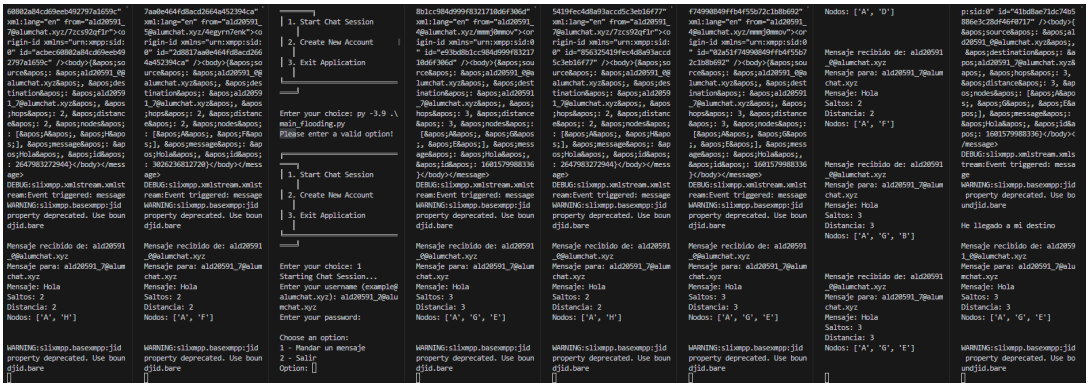
Topología usada



Usuario usados para pruebas


```
{
    "type": "names",
    "config": {
      "A": "ald20591_0@alumchat.xyz",
      "B": "ald20591_1@alumchat.xyz",
      "C": "ald20591_2@alumchat.xyz",
      "D": "ald20591_3@alumchat.xyz",
      "E": "ald20591_4@alumchat.xyz",
      "F": "ald20591_5@alumchat.xyz",
      "G": "ald20591_6@alumchat.xyz",
      "H": "ald20591_7@alumchat.xyz"
    }
}
```


Prueba de uso



Distance Vector

Se desarrolló una simulación para el algoritmo Distance Vector utilizando una estructura basada en la comunicación XMPP. Cada usuario funciona como un nodo individual dentro de una topología previamente establecida. A diferencia del algoritmo de flooding, donde los mensajes se propagan a todos los nodos vecinos, el Distance Vector actualiza y comparte tablas de enrutamiento con sus nodos vecinos.

La interfaz proporcionada facilita a los usuarios iniciar una sesión de chat, establecer una nueva cuenta o abandonar el programa. Una vez que un usuario está en su sesión de chat, puede optar por actualizar su tabla de enrutamiento, visualizarla, enviar mensajes o salir.

El diseño del sistema se basa en el módulo slixmpp. La dinámica del algoritmo Distance Vector es gestionada por la clase `DistanceVector`, que extiende de slixmpp.ClientXMPP. Esta clase posee métodos para gestionar eventos como el inicio de sesión, el envío y recepción de

mensajes, así como la actualización y compartición de las tablas de enrutamiento. La función `update_table` es crucial, ya que actualiza la tabla de enrutamiento usando la tecnica de ballman-ford del nodo basándose en la topología y la comparte con sus vecinos. Los mensajes y las tablas de enrutamiento se transmiten en formato JSON, incluyendo detalles como el nodo origen, el destino, la distancia y el siguiente salto.

Cuando un nodo recibe una tabla de enrutamiento de otro nodo, compara las distancias y actualiza su propia tabla si encuentra una ruta más corta. Esta es la esencia del algoritmo Distance Vector. Adicionalmente, para garantizar que los mensajes lleguen a su destino de manera eficiente, se utiliza una tabla de enrutamiento para determinar el siguiente nodo al que se debe enviar un mensaje. De esta manera, cada mensaje sigue la ruta más corta basándose en la información de la tabla de enrutamiento.

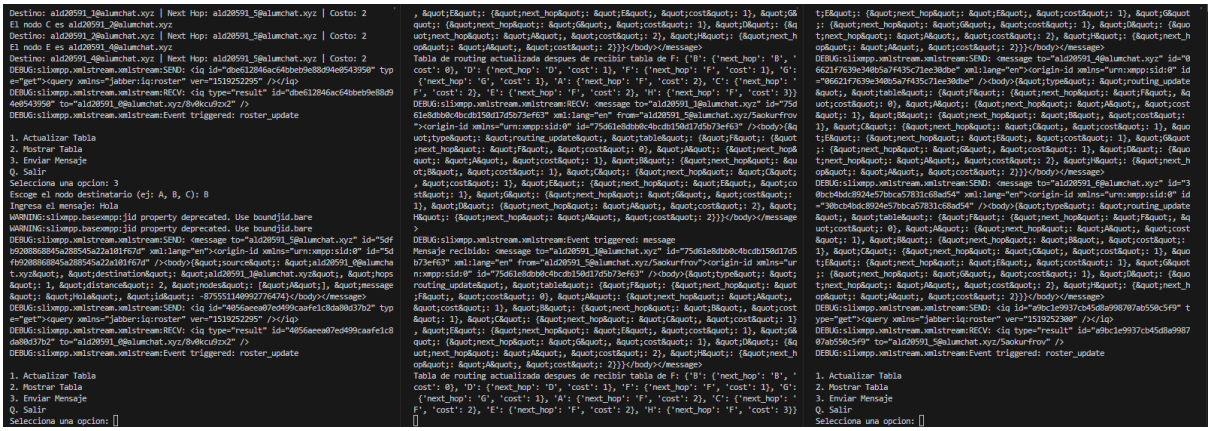
Topología Usada


```
{
    "type": "topo",
    "config": {
      "A": [["D", 1], ["F", 1], ["G", 1], ["H", 1]],
      "B": [["D", 1], ["F", 1], ["G", 1]],
      "C": [["F", 1], ["H", 1]],
      "D": [["A", 1], ["B", 1], ["E", 1], ["G", 1], ["H", 1]],
      "E": [["D", 1], ["F", 1], ["G", 1], ["H", 1]],
      "F": [["A", 1], ["B", 1], ["C", 1], ["E", 1], ["G", 1]],
      "G": [["A", 1], ["B", 1], ["D", 1], ["E", 1], ["F", 1]],
      "H": [["A", 1], ["C", 1], ["D", 1], ["E", 1]]
    }
  }
```


Nombres usados para pruebas


```
{
    "type": "names",
    "config": {
      "A": "ald20591_0@alumchat.xyz",
      "B": "ald20591_1@alumchat.xyz",
      "C": "ald20591_2@alumchat.xyz",
      "D": "ald20591_3@alumchat.xyz",
      "E": "ald20591_4@alumchat.xyz",
      "F": "ald20591_5@alumchat.xyz",
      "G": "ald20591_6@alumchat.xyz",
      "H": "ald20591_7@alumchat.xyz"
    }
  }
```


Pruebas de funcionamiento



LinkState

La simulación del algoritmo Link State se estructura alrededor de la comunicación XMPP, en la que cada usuario representa un nodo individual en una topología definida previamente. A diferencia de algoritmos de enrutamiento como Flooding, Link State se caracteriza por compartir su tabla de enrutamiento con todos los nodos en la red. De este modo, cada nodo puede obtener una perspectiva global y, utilizando algoritmos como Dijkstra, determinar la ruta óptima hacia cualquier otro nodo.

A través de una interfaz intuitiva, el usuario tiene la capacidad de gestionar su tabla de enrutamiento y comunicarse con otros nodos en la red. Una vez que el usuario ha iniciado sesión, tiene la opción de actualizar su tabla de enrutamiento, visualizarla, enviar mensajes o simplemente desconectarse.

La base de esta simulación radica en el módulo slixmpp. La clase `LinkState`, derivada de `slixmpp.ClientXMPP`, es el corazón de esta implementación. Esta clase ofrece funcionalidades clave como cargar la topología y los nombres de usuario, aplicar el algoritmo de Dijkstra para determinar rutas óptimas, y gestionar tanto la actualización como el envío de tablas de enrutamiento a otros nodos.

Con el algoritmo Dijkstra implementado en el método `dijkstra_algorithm`, la simulación puede determinar las rutas más cortas desde el nodo actual hacia cualquier otro nodo en la red. Posteriormente, esta información se comparte con otros nodos mediante el método `share_table`.

Al recibir tablas de enrutamiento de otros nodos, el método `receive_table` las procesa e integra cualquier nueva ruta óptima identificada. Por su parte, el método `send_text_message`

facilita el envío de mensajes entre nodos, aprovechando la tabla de enrutamiento para identificar y seguir la mejor ruta hacia el destino.

En esencia, esta simulación ofrece una representación práctica y eficiente del algoritmo de enrutamiento Link State, garantizando una comunicación eficaz entre nodos y la determinación de rutas óptimas en la red.

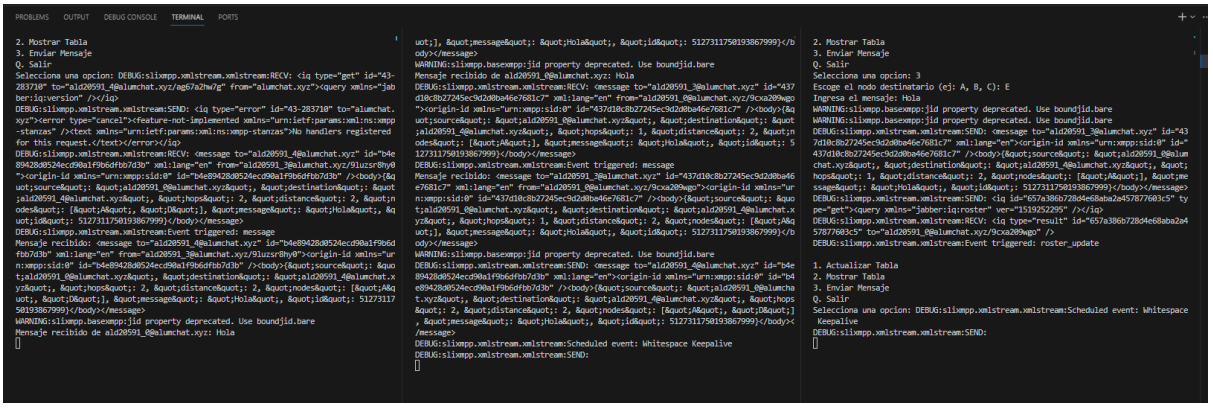
Topología utilizada


```
{
    "type": "topo",
    "config": {
      "A": [["D", 1], ["F", 1], ["G", 1], ["H", 1]],
      "B": [["D", 1], ["F", 1], ["G", 1]],
      "C": [["F", 1], ["H", 1]],
      "D": [["A", 1], ["B", 1], ["E", 1], ["G", 1], ["H", 1]],
      "E": [["D", 1], ["F", 1], ["G", 1], ["H", 1]],
      "F": [["A", 1], ["B", 1], ["C", 1], ["E", 1], ["G", 1]],
      "G": [["A", 1], ["B", 1], ["D", 1], ["E", 1], ["F", 1]],
      "H": [["A", 1], ["C", 1], ["D", 1], ["E", 1]]
    }
  }
```


Nombres de prueba usados


```
{
    "type": "names",
    "config": {
      "A": "ald20591_0@alumchat.xyz",
      "B": "ald20591_1@alumchat.xyz",
      "C": "ald20591_2@alumchat.xyz",
      "D": "ald20591_3@alumchat.xyz",
      "E": "ald20591_4@alumchat.xyz",
      "F": "ald20591_5@alumchat.xyz",
      "G": "ald20591_6@alumchat.xyz",
      "H": "ald20591_7@alumchat.xyz"
    }
  }
```


Prueba de ejecucíón



Discusión

Flooding

Usando XMPP con el servidor aumchat.xyz, pudimos observar la naturaleza del algoritmo de Flooding de manera más clara. Su sencillez, que permite reenviar mensajes a todos los nodos vecinos, garantiza la entrega de los mismos usando XMPP para la comunicación entre nodos, siempre y cuando haya un camino viable. Aunque esta simplicidad asegura confiabilidad, el aumento del tráfico en la red, evidenciado por nuestras pruebas, indica su ineficiencia. Especialmente usando un servidor como aumchat.xyz, el uso intensivo puede generar cuellos de botella o sobrecargas. Aunque la conexión XMPP proporciona confiabilidad, la necesidad de gestionar la tabla de mensajes en cada nodo podría limitar la escalabilidad en redes más amplias.

Distance Vector

El algoritmo Distance Vector, usando XMPP y aumchat.xyz, mostró mayor eficiencia en la transmisión de datos. La interacción mediante XMPP con nodos vecinos mejora la eficiencia, aunque se evidenció lentitud en la actualización de tablas en redes extensas. En redes estables, donde los cambios son mínimos, la eficiencia de este algoritmo, junto con XMPP, puede ser significativa. No obstante, se deben considerar problemas como el "count-to-infinity" y la implementación de mecanismos adicionales para garantizar la robustez.

Link State

Usando XMPP y el servidor aumchat.xyz, el algoritmo Link-State demostró su eficiencia en la adaptación a cambios de la topología. Cargando la topología desde un archivo de texto, la simulación en nuestro servidor mostró cómo se pueden actualizar rutas sin modificar el código. La robustez de esta implementación se destaca en ambientes que usan XMPP, donde la comunicación entre nodos es fluida y constante.

Conclusiones

El Flooding, aunque confiable, mostró su ineficiencia en la interacción con XMPP en aumchat.xyz. Por otro lado, Distance Vector, al interactuar con XMPP, presentó un balance entre eficiencia y complejidad, siendo adecuado para redes moderadas y estables. Link-State, con su adaptabilidad y precisión en el cálculo de rutas, y en combinación con XMPP, probó ser una solución confiable en una red interconectada.

Comentarios

La inclusión de XMPP y aumchat.xyz en las simulaciones proporcionó una perspectiva más realista de la operación de estos algoritmos en entornos actuales. Sería interesante simular cambios en tiempo real en la red usando XMPP para evaluar la adaptabilidad y respuesta de estos algoritmos en escenarios más dinámicos.

Cabe mencionar que el servidor dio muchos problemas haciendo el proceso de las pruebas muy tedioso y aburrido, ya que sin importar nuestra implementación hay los inicios de sesión no funcionan de manera correcta.

Referencias

Wikipedia. (2023, 2 de septiembre). Inundación de red. En Wikipedia. https://es.wikipedia.org/wiki/Inundaci%C3%B3n_de_red#:~:text=La%20inundaci%C3%B3n%20

W0lff4ng.org. (n.d.). Protocolos de enrutamiento Link-State. W0lff4ng.org. Obtenido el 2 de septiembre de 2023, de https://www.w0lff4ng.org/protocolos-de-enrutamiento-link-state/#:~:text=Los%20protocolos%20link%2Dstate%20utilizan,saltos)

Pluralsight. (n.d.). What are Distance Vector routing protocols?. Pluralsight Blog. Obtenido el 2 de septiembre de 2023, de https://www.pluralsight.com/blog/it-ops/dynamic-routing-protocol#:~:text=What%20are%20Distance%20Vector%20routing,packets%20lost%2C%20or%20something%20similar.

GeeksforGeeks. (n.d.). Fixed and Flooding Routing Algorithms. GeeksforGeeks. Obtenido el 2 de septiembre de 2023, de https://www.geeksforgeeks.org/fixed-and-flooding-routing-algorithms/