Universidad del Valle de Guatemala
Facultad de Ingeniería

Redes
Laboratorio #3 Parte 1

Algoritmos de Enrutamiento

JUAN ANGEL CARRERA SOTO 20593
JOSE MARIANO REYES HERNANDEZ 20074
JUAN CARLOS BAJAN CASTRO 20109

Explicación de la práctica

Cada uno de los integrantes del grupo debe de implementar al menos un algoritmo (el lenguaje de programación es libre, y puede ser el mismo para los tres algoritmos, pero es altamente sugerido que sea uno donde se tenga experiencia trabajando con el protocolo XMPP: revisar enunciado segunda parte). Los algoritmos para implementarse son (en negrita los obligatorios):

1. Dijkstra
2. Flooding
3. Distance vector routing
4. Link state routing

Las implementaciones deben de estar debidamente identificadas y publicadas en un repositorio privado (el día de la entrega lo hacen público), junto con sus requerimientos para su uso e instalación. Las implementaciones idealmente deben de poder ejecutarse en distintas plataformas de forma sencilla (makefiles, python venv, .jar). Dependiendo del algoritmo, se deberá simular/proveer los inputs requeridos por cada uno para funcionar adecuadamente. En este laboratorio estaremos simulando o hard-coding tales valores; en la Parte 2 es que procederemos a recibir tales inputs de forma automática de los demás nodos. Lo que requiere cada algoritmo incluye:

- Dijkstra: Topología (nodos, aristas)
- Flooding: ninguno (a lo sumo la Topología).
- Distance Vector: Las Tablas de los Vecinos, Topología
- Link State Routing: Las Tablas de los demás Nodos (de ella se extrae la Topología)

Los nodos deben indicar cuándo están listos para transmitir.

Dijkstra:

El algoritmo de Dijkstra es un algoritmo de búsqueda en grafos que resuelve el problema del camino más corto de una sola fuente para un grafo con costes de camino de aristas no negativos.

Inicio: El algoritmo comienza en el nodo origen e inicializa la distancia a todos los demás nodos a infinito. La distancia al origen se fija en 0.

Procesamiento: A continuación, extrae el nodo con la distancia mínima, calcula la distancia a través de él a cada uno de sus vecinos, y actualiza la distancia del vecino si es menor.

Actualización: Este proceso de extracción del mínimo y relajación de sus vecinos se repite hasta que se ha determinado la distancia a todos los nodos.

Construcción del camino: Una vez finalizado, se construye el árbol del camino más corto retrocediendo desde cada nodo hasta el origen utilizando las distancias calculadas.

Características principales:
- Garantiza la búsqueda del camino más corto para grafos con pesos no negativos.
- Rápido para grafos dispersos y con un número reducido de aristas.
- Sólo funciona para grafos acíclicos dirigidos.
- Complejidad computacional O(|E| + |V|log|V|).

En resumen, el algoritmo de Dijkstra encuentra eficientemente los caminos más cortos desde un origen expandiendo iterativamente el árbol de caminos más cortos hacia el exterior. Se trata de un algoritmo fundamental de búsqueda de grafos utilizado en el encaminamiento y la búsqueda de caminos.

Flooding:

El algoritmo de flooding, o inundación en español, es un método simple y básico para la propagación de información en redes. Su funcionamiento es sencillo:

Inicio: Un nodo que quiere transmitir un mensaje a todos los demás nodos en la red comienza enviándolo a todos sus vecinos.

Reenvío: Cada vez que un nodo recibe el mensaje por primera vez, lo reenvía a todos sus vecinos, excepto al nodo del cual acaba de recibir el mensaje.

Evitar Bucles: Para evitar que el mensaje se propague indefinidamente, se emplean técnicas como:

- Utilizar un contador en el mensaje que se decrementa en cada retransmisión y se detiene cuando llega a cero.
- Registrar mensajes ya vistos, evitando retransmitir un mensaje que ya ha sido procesado.

Finalización: El proceso finaliza para un nodo particular cuando ha recibido el mensaje desde todos sus vecinos o cuando reconoce el mensaje como uno que ya ha retransmitido.

El resultado es que el mensaje se propaga por toda la red, alcanzando a todos los nodos, siempre que haya un camino disponible para llegar a cada nodo.

Aunque el algoritmo de flooding garantiza que el mensaje llegará a todos los nodos de la red, no es eficiente en términos de recursos, ya que puede generar una gran cantidad de tráfico redundante. Por ello, se utiliza principalmente en situaciones donde la robustez y la simplicidad son más críticas que la eficiencia, o como base para algoritmos más complejos.

En resumen, el algoritmo de flooding es como lanzar una piedra en un estanque y ver cómo las ondas se expanden en todas direcciones hasta que cubren toda la superficie del agua.

Link-State:

Descubrimiento de vecinos: Cada nodo en la red recopila información sobre sus vecinos directamente conectados. Esta información incluye identificadores de los nodos vecinos y los costos (distancias) asociados a los enlaces que los conectan.

Construcción de la base de datos de enlace: Cada nodo crea una base de datos que almacena información sobre los enlaces y los nodos vecinos. Esta base de datos se llama la "base de datos de estado de enlace".

Cálculo de caminos más cortos: Utilizando la información recopilada en la LSDB, cada nodo ejecuta el algoritmo de Dijkstra

Actualización de la base de datos estado enlace: Si ocurren cambios en la red, como la caída de un enlace o la incorporación de un nuevo nodo, estos cambios se propagan mediante inundación a todos los nodos en la red.

Cálculo de las rutas: Después de calcular los caminos más cortos, cada nodo tiene información sobre cómo enrutar paquetes hacia cualquier otro nodo en la red.

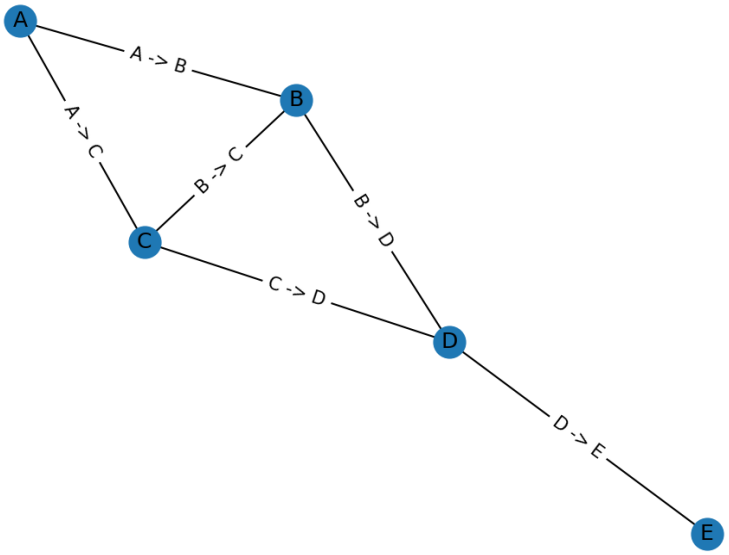
El resultado del algoritmo de estado de enlace es una tabla de enrutamiento en cada nodo, que contiene la mejor ruta (o camino más corto) hacia todos los demás nodos en la red. Esta tabla permite a los nodos determinar rápidamente cómo enviar paquetes a su destino de manera eficiente.

Resultados

Flooding

Para poder visualizar de mejor forma la topología y el traspaso de información nuestro algoritmo es capaz de representar gráficamente las conexiones y lleva una bitácora de los mensajes enviados. De esta forma es más fácil entender el comportamiento y observar la comunicación entre nodos.

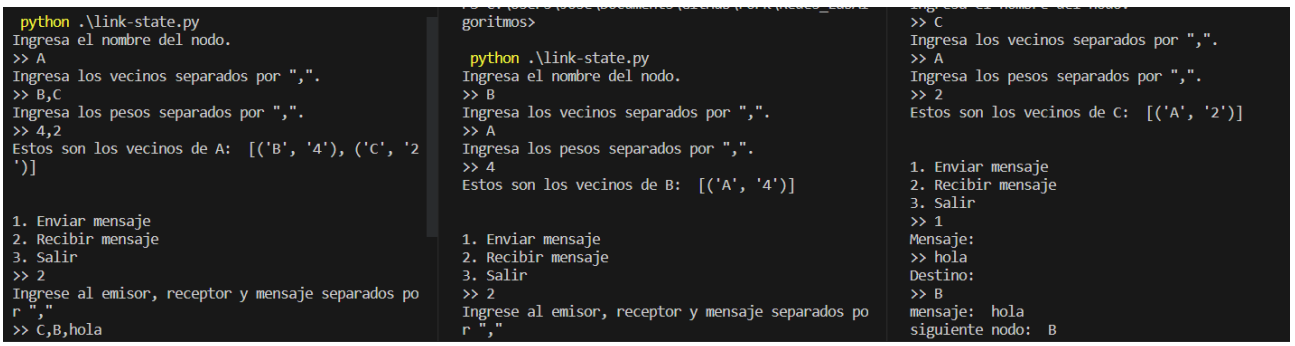
Grafo de conexión



Logs:
{'type': 'info', 'headers': {'from': 'A', 'to': 'B', 'hop_count': 2}, 'payload': 'Hello from A to E!'}
{'type': 'info', 'headers': {'from': 'B', 'to': 'D', 'hop_count': 3}, 'payload': 'Hello from A to E!'}
{'type': 'info', 'headers': {'from': 'D', 'to': 'C', 'hop_count': 4}, 'payload': 'Hello from A to E!'}
{'type': 'info', 'headers': {'from': 'D', 'to': 'E', 'hop_count': 4}, 'payload': 'Hello from A to E!'}

Como se puede observar, a pesar de las múltiples conexiones de la topología, el sistema muestra el mensaje desde A hacia E. A pesar de que los saltos necesarios solo son 3, dado que el algoritmo es flooding, se envían los mensajes a todos los nodos de todas formas

Link-State



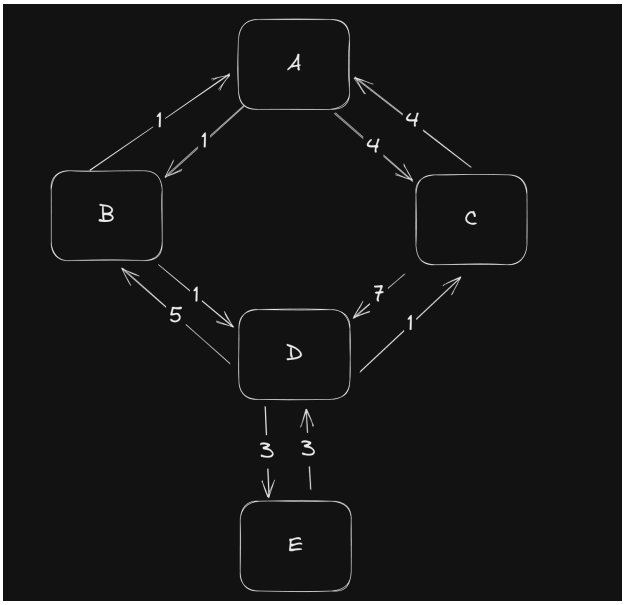
Como tal en esta primera fase definimos la topología manualmente en este caso solo usamos 3 Nodos para simular el funcionamiento:

```
{
    "A": "[('B', 4), ('C', 2)]",
    "B": "[('A', 4)]",
    "C": "[('C', 2)]"
}
```

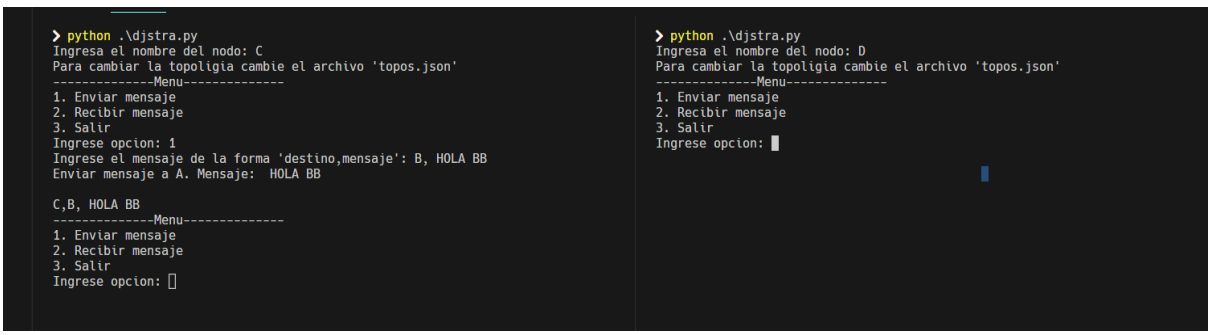
La simulación del recorrido más corto la hacemos sobre la topología con ayuda del algoritmo de Dijkstra y así podemos ver en qué momento de la comunicación va el proceso. Todo es bastante manual por lo mismo que no tenemos comunicación entre los nodos, pero la primera captura mostraría su funcionamiento, donde sin tener que pedir el Emisor, Receptor y Mensaje podríamos llegar a recibir el mensaje de un nodo externo cuando implementemos la segunda parte del algoritmo.

Dijkstra

Para el ejemplo creado se uso esta topología para enviar los mensajes.



Ejemplo mensaje entre C a B:



En la simulación podemos ver que tomó el camino más corto a través de C-> A-> B ya que si se hubiera hecho el recorrido a través de D hubiera sido más lento.

Discusión

Los algoritmos de Dijkstra, Link-State y Flooding se utilizan en redes para el encaminamiento y la determinación de rutas, pero tienen algunas diferencias clave:

El algoritmo de Dijkstra es muy eficaz para encontrar el camino más corto desde un único origen a todos los demás nodos de un grafo. Proporciona caminos óptimos y funciona en cualquier grafo. Sin embargo, requiere conocer de antemano todas las distancias de los enlaces y requiere más memoria. Además, no se adapta bien a medida que aumenta el tamaño de la red.

Los algoritmos de enlace-estado como OSPF construyen un mapa topológico completo de la red inundando la información de los enlaces. A partir de este mapa, cada router calcula de forma independiente el camino más corto a cualquier otro router mediante el algoritmo de Dijkstra. La ventaja es un enrutamiento altamente escalable con tiempos de convergencia rápidos. Pero el link-state requiere más memoria y capacidad de procesamiento en cada encaminador.

Flooding es una técnica de difusión simple en la que cada nodo envía un mensaje a todos sus vecinos. Es fácil de implementar, pero tiene una gran sobrecarga de mensajes redundantes. Además, no determina las rutas de reenvío óptimas. Sin embargo, su simplicidad la hace útil cuando no se dispone de un conocimiento global de la red.

En resumen, Dijkstra proporciona los caminos más cortos óptimos, pero no es escalable. Link-state utiliza Dijkstra con conocimiento topológico global para un encaminamiento descentralizado eficiente, a costa de mayores requisitos de routers. Flooding es sencillo y útil cuando el conocimiento de la red es limitado, a pesar de su mayor sobrecarga. La elección depende de las especificaciones

Comentario Grupal

Al simular algoritmos de encaminamiento como los de Dijkstra y Link-State utilizando sólo un puñado de terminales para representar nodos de red, nuestro equipo se enfrentó a varias dificultades fundamentales. Generar nuevos nodos y topologías no era demasiado difícil, pero hacer que se comunicaran correctamente para emular las condiciones reales de la red resultó complicado.
Cada miembro abordó los problemas de forma diferente: algunos se centraron en la sincronización y la mensajería, otros en detectar cambios en la topología y otros en mostrar y visualizar las rutas. Sin embargo, todos nos enfrentamos a problemas como evitar el bloqueo durante el paso de mensajes entre nodos, seguir las mutaciones de la topología a medida que evoluciona la red y manejar aspectos como la detección de duplicados en medio de la concurrencia.
A pesar de su pequeña escala, la dinámica de la simulación y la complejidad de la coordinación de los nodos terminales plantean grandes retos.
Conseguir que los terminales trabajarán juntos para emular correctamente la ejecución del algoritmo y comunicar los cambios de estado del enrutamiento exigió un esfuerzo considerable y una resolución creativa de los problemas.

Conclusiones

- La implementación del algoritmo de Link-State es fundamental en el ámbito de las redes de comunicación para lograr un enrutamiento eficiente y confiable. A través de la recopilación de información sobre la topología de la red y el cálculo de rutas óptimas, este algoritmo permite a los nodos tomar decisiones informadas sobre cómo dirigir los paquetes de datos de manera efectiva hacia su destino.
- Link-State también proporciona una visión más precisa de la topología de la red, lo que conlleva una mayor adaptabilidad a los cambios en la infraestructura y de igual manera otorga la capacidad de calcular rutas óptimas mejora el rendimiento general de la red, minimizando las latencias y maximizando la velocidad de entrega de los paquetes.
- El algoritmo de Dijkstra fue bastante bueno para obtener el camino más corto de una topología. Esto cuando en la topología es conocido y los pesos entre cada nodo están definidos.
- Utilizar el algoritmo de de flooding puede ser muy eficiente y útil cuando se precisa de un 100% de seguridad de que el mensaje llegará eventualmente al receptor, sin embargo en tiempo de llegada puede variar hasta lo indefinido pues intenta inundar todo el sistema con esta petición.