

Algoritmos de enrutamiento

Descripción de la Práctica

Este laboratorio tiene como objetivo comprender y poner en práctica los algoritmos de enrutamiento utilizados en redes de computadoras, aplicando tanto un enfoque teórico como práctico mediante programación. En la primera parte, se busca implementar un algoritmo de enrutamiento (como Dijkstra) de forma centralizada para construir las tablas de encaminamiento de cada nodo y validarlas con el uso de sockets, simulando la comunicación entre routers a través del envío y reenvío de paquetes. De esta manera, se verifica que los mensajes lleguen correctamente a su destino siguiendo la ruta óptima definida por las tablas. Posteriormente, se amplía la práctica hacia otros algoritmos de enrutamiento distribuidos, incorporando conceptos como forwarding hop-by-hop, TTL, mensajes de control (HELLO, INFO) y diferentes estrategias de construcción de rutas. El laboratorio permite así reforzar el entendimiento de cómo los routers calculan caminos, intercambian información de estado y reenvían paquetes en una red real, mostrando de manera práctica el funcionamiento de protocolos de enrutamiento.

Descripción de los Algoritmos Utilizados y su Implementación

El algoritmo de Dijkstra es un método utilizado para encontrar los caminos más cortos desde un nodo origen hasta todos los demás nodos de un grafo ponderado cuyos pesos son no negativos. La idea principal consiste en ir seleccionando, de manera iterativa, el nodo con la menor distancia acumulada desde el origen y actualizar los costos de llegar a sus vecinos. Una vez que un nodo es marcado como visitado, se garantiza que la distancia registrada hacia él es la mínima posible. El resultado final es un conjunto de distancias mínimas y, a partir de estas, la construcción de una tabla de enrutamiento, donde cada destino se asocia con un costo total y un next-hop (siguiente salto) que indica por qué vecino debe comenzar el envío de un paquete para alcanzar ese destino.

Para este laboratorio, el algoritmo de Dijkstra se implementó en Python a partir de un grafo representado como un diccionario de adyacencia, donde cada nodo se relaciona con los routers vecinos y el costo de los enlaces. Se desarrolló un módulo grafo.py que permite crear la topología, y un módulo dijkstra.py que ejecuta el algoritmo y genera las tablas de enrutamiento correspondientes para cada nodo. Estas tablas se guardaron en formato JSON, lo cual facilita que cada nodo las consulte al momento de decidir

cómo reenviar un paquete.

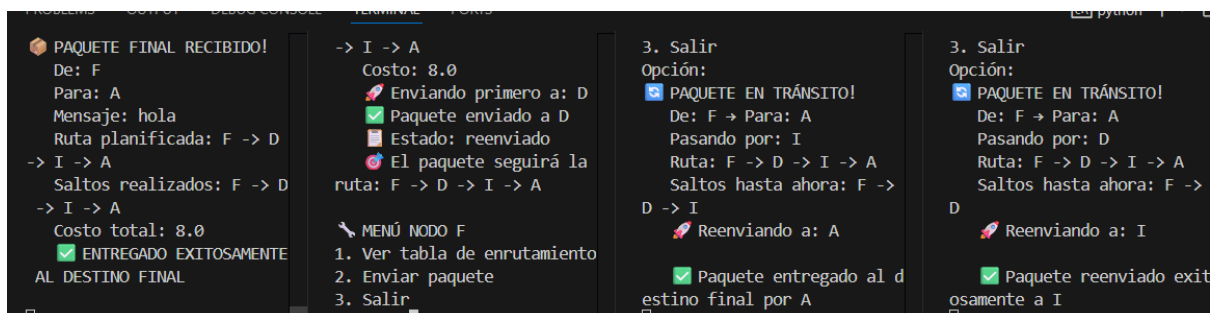
Posteriormente, se utilizó el módulo de sockets de Python para simular la comunicación entre nodos. Cada nodo corre un servidor que escucha en un puerto y, al recibir un paquete, verifica si es el destino final; de lo contrario, consulta su tabla de enrutamiento para determinar el next-hop, disminuye el valor de TTL y reenvía el paquete. Este proceso permite observar de manera práctica cómo los paquetes viajan de un origen a un destino siguiendo el camino óptimo calculado por Dijkstra, validando así el funcionamiento del algoritmo en una red simulada.

El algoritmo de Link State es un enfoque de enrutamiento dinámico en el cual cada nodo de la red descubre a sus vecinos, mide los costos de los enlaces y genera mensajes llamados *Link State Packets* (LSP). Estos LSPs se difunden por toda la red mediante flooding, de manera que cada nodo construye una base de datos global de estado de enlaces (LSDB). Con esa información, cada nodo ejecuta localmente el algoritmo de Dijkstra para calcular las rutas más cortas hacia todos los destinos. Esto asegura que todos los nodos tengan una visión consistente de la topología de la red y puedan reaccionar rápidamente a fallas o cambios en los enlaces

La implementación del algoritmo de Link State se realizó en dos etapas complementarias. Primero se desarrolló una simulación en memoria donde cada nodo, representado por la clase LinkStateNode, genera y procesa Link State Packets (LSPs), mantiene su base de datos de estado de enlaces (LSDB) y ejecuta el algoritmo de Dijkstra sobre el grafo global reconstruido para calcular su tabla de enrutamiento.

Resultados

Dijkstra



```
PAQUETE FINAL RECIBIDO!  
De: F  
Para: A  
Mensaje: hola  
Ruta planificada: F -> D  
-> I -> A  
Saltos realizados: F -> D  
-> I -> A  
Costo total: 8.0  
✓ ENTREGADO EXITOSAMENTE  
AL DESTINO FINAL
```

```
-> I -> A  
Costo: 8.0  
✶ Enviando primero a: D  
✓ Paquete enviado a D  
✶ Estado: reenviado  
✶ El paquete seguirá la  
ruta: F -> D -> I -> A
```

```
3. Salir  
Opción:  
✶ PAQUETE EN TRÁNSITO!  
De: F -> Para: A  
Pasando por: I  
Ruta: F -> D -> I -> A  
Saltos hasta ahora: F ->  
D -> I  
✶ Reenviando a: A  
✓ Paquete entregado al d  
estino final por A
```

```
3. Salir  
Opción:  
✶ PAQUETE EN TRÁNSITO!  
De: F -> Para: A  
Pasando por: D  
Ruta: F -> D -> I -> A  
Saltos hasta ahora: F ->  
D  
✶ Reenviando a: I  
✓ Paquete reenviado exit  
osamente a I
```

Link state

```
Opción: 🐦 Link State recibido de I: {'A': 1, 'D': 6}
```

```
📊 Tabla Link State actualizada: 4 destinos
```

```
📡 Ping recibido de I, esperan nodo: A
```

```
📡 Enviando identificación: A
```

```
📦 PAQUETE FINAL RECIBIDO! (LINK STATE)
```

```
De: I
```

```
Para: A
```

```
Mensaje: aa
```

```
Ruta Link State: I -> A
```

```
Salto realizado: I -> A
```

```
Costo total: 1.0
```

```
✅ ENTREGADO EXITOSAMENTE AL DESTINO FINAL
```

```
Destinos disponibles: A, D
```

```
Destino: A
```

```
Mensaje (opcional): aa
```

```
📦 ENVIANDO PAQUETE (LINK STATE):
```

```
De: I
```

```
Para: A
```

```
Ruta Link State calculada: I -> A
```

```
Costo: 1.0
```

```
📡 Enviando primero a: A
```

```
✅ Paquete Link State enviado: entregado
```

```
📡 El paquete seguirá la ruta Link State: I -> A
```

Discusión

La simulación en memoria permitió validar el funcionamiento lógico del algoritmo: se comprobó la generación y difusión de LSPs, la actualización de la LSDB y la recomputación de rutas al cambiar costos o eliminar enlaces. Posteriormente, la implementación distribuida con sockets aportó realismo, ya que los LSPs se transmiten únicamente a vecinos y se reenvían en flooding, con control de duplicados y confirmaciones. Además, se incluyeron mecanismos de envejecimiento de LSPs y métricas de monitoreo (LSPs enviados/recibidos, tablas calculadas), lo cual facilita observar la convergencia de la red en tiempo real.

En la práctica, aunque el algoritmo de Dijkstra garantiza la obtención de rutas óptimas, su aplicación directa en un escenario de enrutamiento distribuido puede generar una sobrecarga significativa en la red. Esto se debe a que, para calcular el camino más corto, los paquetes deben atravesar múltiples nodos intermedios antes de llegar al destino. Cada uno de estos saltos implica operaciones de recepción, procesamiento y reenvío, lo cual incrementa la latencia y el consumo de recursos. Además, si el tráfico de control (mensajes de actualización o intercambio de tablas) también se propaga a través de varios nodos, el ancho de banda disponible para datos útiles se ve reducido.

Conclusiones

El algoritmo de Dijkstra demostró ser una herramienta eficaz para calcular rutas óptimas en un grafo, garantizando siempre el menor costo desde un nodo origen hacia todos los destinos. Su principal fortaleza es la precisión y la consistencia de los caminos obtenidos, lo que lo hace ideal para la construcción de tablas de enrutamiento. Sin embargo, en entornos distribuidos presenta limitaciones importantes: cada cálculo requiere información global de la red y los mensajes deben atravesar múltiples nodos, lo que genera sobrecarga de tráfico de control y un mayor consumo de recursos en redes grandes. En conclusión, Dijkstra es adecuado para topologías estáticas o centralizadas, pero en escenarios dinámicos requiere complementarse con mecanismos de actualización más eficientes.

El algoritmo de **Link State**, basado en la difusión de paquetes de estado de enlace y en el uso de Dijkstra de manera local en cada nodo, permitió observar una convergencia más rápida y confiable frente a cambios de topología. Su principal ventaja es que todos los nodos construyen la misma visión de la red y recalculan rutas de forma autónoma, lo que garantiza consistencia y evita bucles. No obstante, demanda **mayor complejidad computacional y memoria**, ya que cada nodo debe almacenar la topología completa y mantener su LSDB actualizada. A pesar de este costo, Link State se mostró más robusto y cercano al funcionamiento de protocolos de enrutamiento reales como OSPF, lo que lo convierte en una solución práctica y escalable para redes de tamaño medio y grande.