Laboratorio 3 – Algoritmos de Enrutamiento

Parte 3.1: Implementación de Algoritmos

Andy Fuentes Davis Roldán Diego Linares

Universidad del Valle de Guatemala
Facultad de Ingeniería
Curso: Redes de Computadoras

Índice

1. Antecedentes

En la primera parte del Laboratorio 3 se implementaron los algoritmos de enrutamiento clásicos (Flooding, Distance Vector y Link State Routing) utilizando sockets TCP locales. Esto permitió comprender cómo se construyen las tablas de ruteo y cómo los nodos descubren vecinos y comparten información de red.

Para la fase final, el laboratorio migra hacia un esquema de Publisher/Subscriber (Pub/Sub) usando Redis como middleware de mensajería. En este esquema, cada nodo es un proceso independiente que:

- Publica mensajes de control y datos en un canal Redis.
- Se suscribe a los canales de sus vecinos.
- Emite mensajes de tipo HELLO, INFO y MESSAGE, siguiendo el protocolo definido en la guía.
- Mantiene dos procesos/hilos lógicos:
    - Forwarding (manejo de recepción/reenvío de paquetes).
    - Routing (ejecuta el algoritmo LSR con Dijkstra).

2. Objetivos

- Conocer el funcionamiento de algoritmos de enrutamiento clásicos en redes.
- Implementar los algoritmos **Flooding**, **Distance Vector** y **Link State Routing (LSR)**.
- Probar los algoritmos en una red simulada localmente mediante sockets TCP.
- Analizar el comportamiento de las tablas de enrutamiento en cada protocolo.
- Migrar la simulación de nodos desde TCP local hacia un sistema distribuido real con **Redis Pub/Sub**.
- Implementar la lógica de **HELLO (descubrimiento de vecinos)**, **INFO (LSPs y tablas de ruteo)** y **MESSAGE (datos de usuario)** bajo el protocolo especificado.
- Demostrar que las tablas de enrutamiento se actualizan dinámicamente y que los mensajes llegan al destino correcto.

3. Algoritmos Implementados

3.1. Flooding

Aunque no hay un archivo explícito de *flooding.py*, el sistema implementa **flooding controlado** cuando un mensaje no tiene ruta conocida. Esto ocurre en el método send_message: si no hay next_hop para un destino, el mensaje se envía a todos los vecinos (con control de duplicados mediante msg_id).

3.2. Link State Routing (LSR)

Cada nodo envía periódicamente un **HELLO** a sus vecinos directos, para confirmar conectividad. Los nodos difunden paquetes **INFO** con el estado de sus enlaces (LSP), lo cual permite que cada router construya una copia global de la topología. Al recibir un nuevo LSP, el nodo ejecuta el **algoritmo de Dijkstra** para recalcular las rutas más cortas y actualizar su tabla de ruteo.

4. Topología y configuración

Para las pruebas se utilizó un archivo **topo.json** con la siguiente configuración:

```
{
  "type": "topo",
  "config": {
    "A": ["B"],
    "B": ["A", "C"],
    "C": ["B"]
  }
}
```

Y un archivo **names.json** con los canales Redis asignados a cada nodo:

```
{
  "type": "names",
  "config": {
    "A": "sec20.topologia1.nodeA",
    "B": "sec20.topologia1.nodeB",
    "C": "sec20.topologia1.nodeC"
```

```
  }
}
```

El .env contenía las credenciales de Redis:

```
REDIS_HOST=lab3.redesuvg.cloud
REDIS_PORT=6379
REDIS_PWD=UVGRedis2025
SECTION=sec20
TOPO=topologia1
NODE=A
NAMES_PATH=./configs/names.json
TOPO_PATH=./configs/topo.json
PROTO=lsr
```

5. Pruebas y Resultados

5.1. Inicio de nodos

Al ejecutar python -m src.main en los nodos A, B y C, se observó:

- Cada nodo imprime sus vecinos detectados.
- Se envían **HELLO/INFO iniciales**.
- Las tablas de ruteo comienzan a llenarse y actualizarse con los costos más bajos.

5.2. Comunicación entre nodos

Desde la CLI se envió un mensaje:

```
python -m src.main --send C --body "hola C, soy A"
```

En la consola del nodo C se recibió:

```
[C] DELIVER DATA from=A payload=hola C, soy A
trace=[{'hop':'A'}, {'hop':'B'}]
```

La topolog´ıa utilizada fue:

```
{
  "type": "topo",
  "config": {
    "A": ["B"],
    "B": ["A", "C"],
    "C": ["B"]
  }
}
```

Ejemplo de salida en nodo A:

```
[INFO] NODE-A: Vecinos de A: ['B']
[INFO] NODE-A: Canal propio: sec20.topologia1.nodeA
[INFO] LSR-A: RoutingLSRService iniciado
[INFO] FWD-A: ForwardingService iniciado
[INFO] NODE-A: HELLO/INFO iniciales enviados
[INFO] LSR-A: Tabla de ruteo actualizada (2 destinos)
```

Confirmando que el mensaje viajó por A → B → C, usando la tabla de ruteo calculada con Dijkstra.

6. Conclusiones

- Se logró migrar de un esquema local con Sockets TCP hacia un esquema distribuido real con **Redis Pub/Sub**, cumpliendo los requisitos de la Parte 2.

- Los nodos implementan correctamente los mensajes de control **HELLO** e **INFO**, además de reenvío de datos con **MESSAGE**.

- El algoritmo de **LSR con Dijkstra** mantiene actualizadas las tablas de ruteo en todos los nodos.

- El mecanismo de **flooding controlado** garantiza entrega incluso cuando no hay rutas conocidas.

- Se comprobó en pruebas reales con varios nodos (A, B, C) que los mensajes viajan por la ruta óptima definida por los costos de la topología.

- Este laboratorio sienta la base para la simulación distribuida entre estudiantes, donde cada nodo es un participante real en la red.