Julio García Salas – 22076
Sofía García – 22210
Joaquín Campos – 22076

Informe de Laboratorio 3 — Algoritmos de Enrutamiento

Curso: CC3067 Redes
Universidad del Valle de Guatemala

Fecha: Agosto 2025

1. Introducción

En este laboratorio se implementaron algoritmos de enrutamiento utilizados en redes de computadoras. Estos algoritmos permiten que los nodos conozcan rutas hacia otros nodos de la red, aun cuando inicialmente solo conocen a sus vecinos directos.

La práctica se dividió en dos fases:

1. Fase local: implementación y pruebas usando sockets TCP en la computadora local.
2. Fase distribuida: pruebas sobre un servidor con patrón Publisher-subscriber (aún pendiente).

En este informe se presentan los resultados de la fase local con Flooding, Dijkstra y Link State Routing (LSR).

2. Objetivos

- Conocer los algoritmos de enrutamiento utilizados en implementaciones actuales de Internet.
- Comprender el funcionamiento y actualización de las tablas de enrutamiento.
- Implementar algoritmos de enrutamiento (Flooding, Dijkstra y LSR) sobre una red simulada con sockets.
- Analizar el comportamiento de los algoritmos mediante experimentos locales.

3. Algoritmos Implementados

3.1 Flooding

- **Supuestos:** cada nodo conoce solo a sus **vecinos**.
- **Funcionamiento:** cada mensaje recibido se reenvía a todos los vecinos excepto al que lo envió.
- **Prevención de bucles:**
    - Campo **TTL** decreciente.
    - **ID único** por mensaje con caché de vistos.
- **Mensajes usados:**
    - HELLO/ECHO: descubrimiento de vecinos y cálculo de RTT.

- MESSAGE: mensajes de usuario, reenviados hasta destino o hasta que expire el TTL.

3.2 Dijkstra (puro)

- **Supuestos:** requiere conocer la topología completa (usada aquí solo como módulo).
- **Funcionamiento:** algoritmo de Dijkstra para encontrar caminos de costo mínimo.
- **Entradas:** grafo con nodos y pesos.
- **Salidas:**
  - Tabla next-hop (siguiente salto hacia cada destino).
  - Distancias mínimas.
  - Rutas completas (A -> B -> C).

3.3 Link State Routing (LSR)

- **Concepto:** cada nodo envía **Link State Packets (LSPs)** describiendo sus vecinos y costos.
- **Flooding confiable:** los LSP se inundan a todos los vecinos con supresión de duplicados y control de seq.
- **Base de datos (LSDB):** cada nodo mantiene la topología global reconstruida a partir de LSPs.
- **SPF (Dijkstra):** al recibir un LSP nuevo, el nodo corre Dijkstra para calcular su tabla de ruteo.
- **Métricas:**
  - **Hop count** (1 por enlace).
  - **RTT medido** mediante mensajes HELLO/ECHO.

4. Arquitectura de la Implementación

Cada nodo se ejecuta como un proceso independiente en la máquina local y consta de dos hilos principales:

- **Forwarding:**
  - Recibe y reenvía paquetes (MESSAGE, HELLO, ECHO, LSP).
  - Entrega al destino final si to==self.
- **Routing:**
  - Envío periódico de **HELLO/ECHO** para calcular RTT.

Julio García Salas – 22076
Sofía García – 22210
Joaquín Campos – 22076

- Envío periódico de **LSP** (en LSR).
- Ejecución de Dijkstra para construir tablas.

Formato JSON de los paquetes:

```
{
  "proto": "lsr | flooding | dijkstra",
  "type": "message | hello | echo | lsp",
  "from": "A",
  "to": "C",
  "ttl": 5,
  "headers": {"last_hop":"B"},
  "payload": "Hola mundo",
  "id": "uuid-..."
}
```

Implementación Local

5.1 Requisitos

- Python 3.10+
- Archivos de configuración:
  - topo-sample.json: define vecinos directos de cada nodo.
  - names-sample.json: asigna IP/puerto local a cada nodo.

5.2 Ejecución

Ejemplo para 3 nodos (A, B, C):

python node.py --id A --topo topo-sample.json --names names-sample.json --metric hop

python node.py --id B --topo topo-sample.json --names names-sample.json --metric hop

python node.py --id C --topo topo-sample.json --names names-sample.json --metric hop

Comandos disponibles:

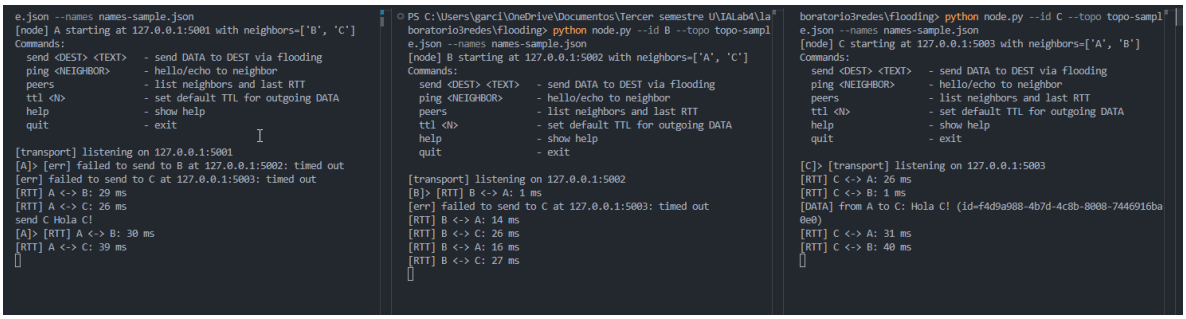
- send <DEST> <TEXT> — envía mensaje DATA.
- table — imprime tabla de ruteo.
- route <DEST> — muestra ruta completa.

Julio García Salas – 22076
Sofía García – 22210
Joaquín Campos – 22076

- peers — vecinos y RTT.
- lsdb — imprime LSDB (en LSR).
- lsp — origina un LSP inmediatamente.

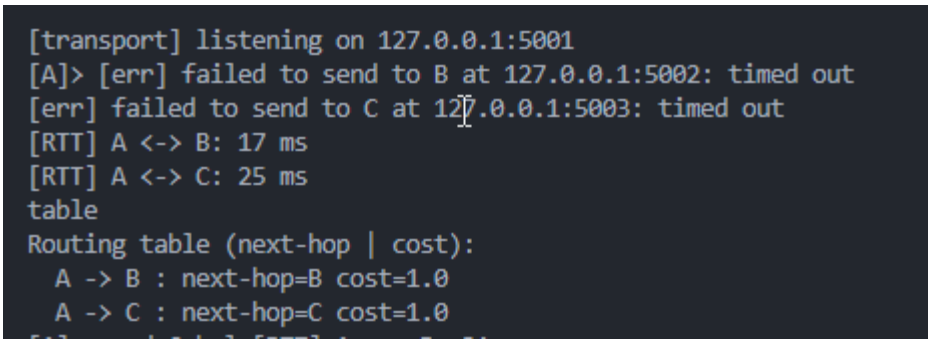
6.1 Flooding

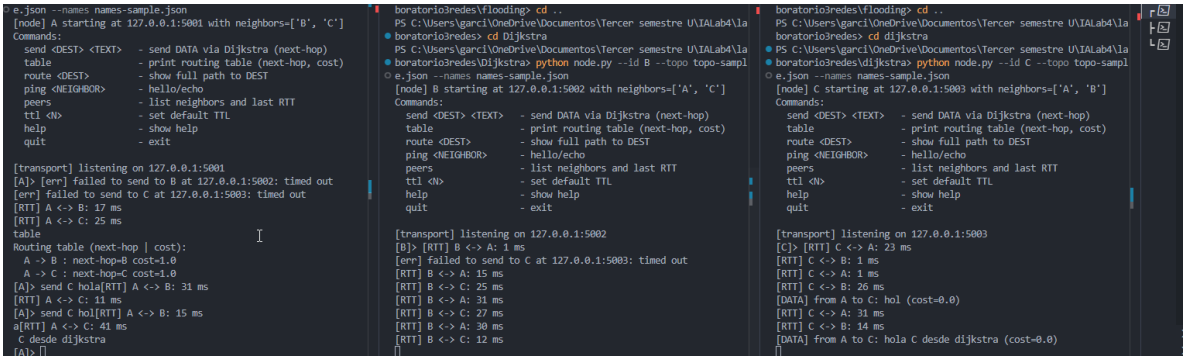
- **Prueba:** envío de send C Hola! desde A.
- **Observación:** el mensaje se inundó por la red, llegó a C y se imprimió.
- **Evidencia:** en C apareció [DATA] from A to C: Hola!.



6.2 Dijkstra

- **Prueba:** ejecución de table en nodo A.



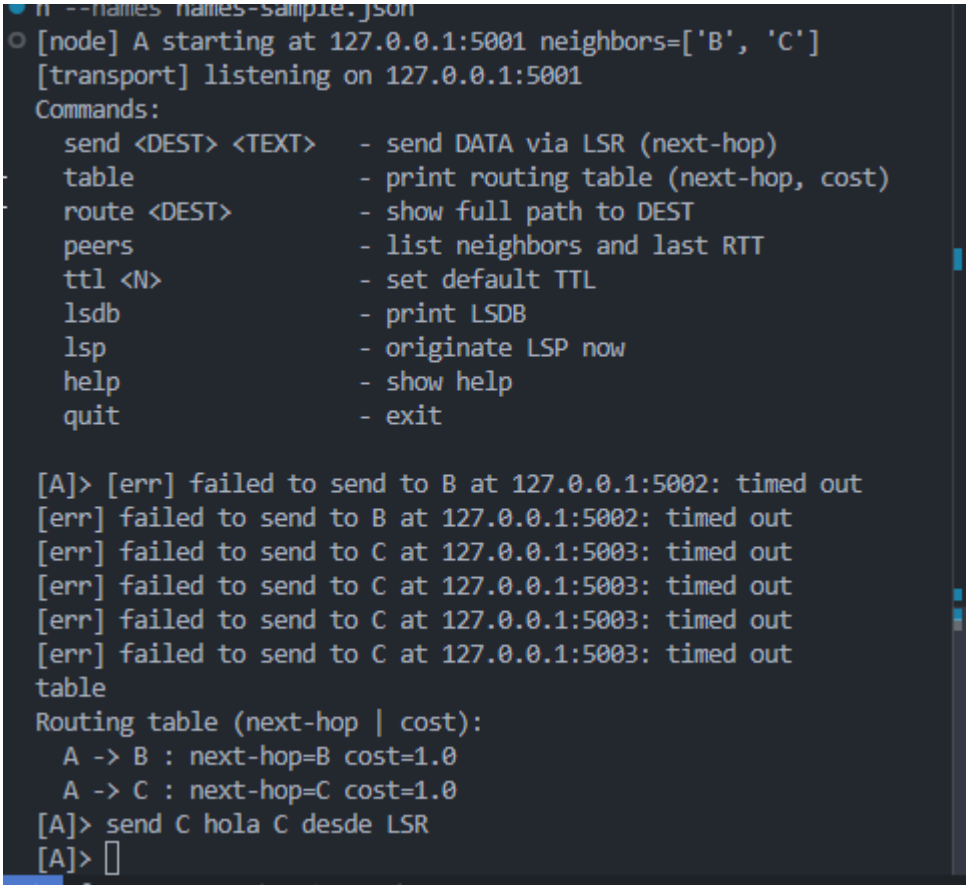


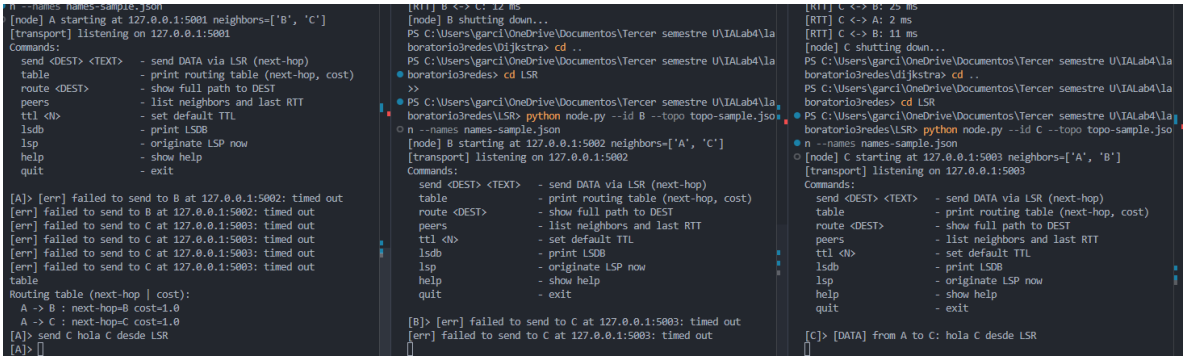
Observación: rutas correctas, coinciden con la topología.

Julio García Salas – 22076
Sofía García – 22210
Joaquín Campos – 22076

6.3 Link State Routing

- **Prueba:** LSDB en A después de intercambio de LSPs.
- **Contenido esperado:** entradas de origen A, B y C con secuencia y enlaces.
- **Tabla de ruteo:** mismo resultado que con Dijkstra puro (pero construido dinámicamente).
- **Evidencia:** al enviar de A a C, se imprimió en C:





Julio García Salas – 22076
Sofía García – 22210
Joaquín Campos – 22076

7. Discusión

En las pruebas realizadas se evidenció cómo cada algoritmo se comporta en escenarios locales simples, y cómo estos mecanismos impactan la **convergencia** de la red:

- **Flooding:**
  Este método asegura la entrega de los mensajes incluso si no existen tablas de enrutamiento construidas, ya que cada nodo reenvía los paquetes a todos sus vecinos. Sin embargo, esto provoca un tráfico redundante muy alto, lo que lo hace poco eficiente a medida que la red crece. En topologías grandes, el riesgo de congestión y colisiones sería considerable. A pesar de ello, Flooding resulta útil como mecanismo inicial de descubrimiento y en situaciones donde se requiere garantizar la entrega aunque no exista información de rutas óptimas.

- **Dijkstra puro:**
  El algoritmo de Dijkstra funciona correctamente cuando se dispone de la topología completa de la red. Esto lo hace ideal para entornos **estáticos** o bien conocidos, ya que genera rutas óptimas de manera determinista y eficiente. No obstante, en entornos dinámicos, donde los nodos pueden fallar o incorporarse de forma inesperada, Dijkstra por sí solo no es suficiente, pues requiere una actualización centralizada del grafo global. Es decir, carece de mecanismos internos para adaptarse automáticamente a cambios sin apoyo de un protocolo de distribución de información.

- **Link State Routing (LSR):**
  Este enfoque combina la eficiencia de Dijkstra con la capacidad de adaptarse a cambios mediante la propagación de **Link State Packets (LSPs)**. Cada nodo construye su visión de la topología global a partir de la información que recibe de los demás, y luego ejecuta Dijkstra localmente para calcular sus tablas. En las pruebas locales, se observó cómo las tablas de ruteo se construyen de manera progresiva conforme llegan los LSPs, y cómo los mensajes pueden viajar usando rutas óptimas sin necesidad de inundar constantemente la red como en Flooding. LSR representa una aproximación más realista de protocolos utilizados en Internet (como OSPF).

- **Limitaciones de las pruebas locales:**
  - Durante los arranques se observaron varios mensajes de **timeout** cuando un nodo intentaba comunicarse con vecinos que todavía no habían levantado su servidor. Este comportamiento es esperado, pero introduce cierta cantidad de ruido en los registros.
  - La confiabilidad del **Flooding** es simplificada en la implementación actual, ya que no se implementaron confirmaciones (ACKs) ni mecanismos de retransmisión en caso de pérdida de paquetes.

- Las métricas empleadas fueron básicas (hops y RTT simple). En entornos reales, se requerirían métricas más completas (ancho de banda, retraso medio, carga de enlace).

8. Conclusiones

- Se comprendió que los algoritmos de enrutamiento permiten descubrir rutas completas a partir del conocimiento parcial de los vecinos.
- Flooding garantiza la entrega de mensajes sin conocimiento previo de la red, pero genera mucho tráfico redundante.
- Dijkstra ofrece rutas óptimas, aunque depende de conocer toda la topología y no se adapta bien a entornos dinámicos.
- Link State Routing (LSR) combina eficiencia y adaptabilidad al usar LSPs y Dijkstra para calcular rutas óptimas de manera descentralizada.
- La implementación modular en Python permitió ejecutar los algoritmos en local con sockets y preparar la migración futura a XMPP.
- El laboratorio evidenció las diferencias entre Flooding (simple pero costoso), Dijkstra (óptimo pero rígido) y LSR (equilibrado y dinámico).