

# UNIVERSIDAD DEL VALLE DE GUATEMALA



## Trabajo

Andre Marroquin Tarot - 22266

Sergio Orellana - 221122

Carlos Valladares - 221164

Redes

**Repo git:** <https://github.com/mar22266/Lab3Redes.git>

## Descripción de la práctica

Se implementaron y probaron tres algoritmos de enrutamiento en un entorno local usando sockets TCP y mensajes JSON: Flooding, Dijkstra y Link State Routing (LSR). Cada nodo corre como proceso con dos hilos: forwarding escucha y reenvío y routing cálculo / propagación de estado. La topología y los puertos se definen en configs topo-sample.json y names-sample.json. Para ejecutar y orquestar pruebas se empleó un Makefile con targets para levantar nodos y enviar mensajes de prueba.

## Algoritmos utilizados y su implementación

### Flooding

- Idea: reenvía el paquete a todos los vecinos excepto al que lo envió, controlando bucles con TTL y un filtro de duplicados.
- Mensajes: type="message" con headers.MSG\_ID (UUID), ttl, from, to y payload.
- **Prevención de duplicados:**
  - En reenvío: conjunto SEEN para ignorar el mismo MSG\_ID más de una vez.
  - En entrega al destino: caché DELIVERED para imprimir el mensaje solo una vez, incluso si llega por múltiples caminos.
- **Uso típico:** robusto ante fallas de tabla de rutas, pero ineficiente en consumo de ancho de banda.

### Dijkstra (estático)

- Idea: calcula rutas de costo mínimo desde cada nodo en un grafo no ponderado coste 1/enlace por simplicidad.
- Construcción del grafo: desde topo-sample.json.
- Salida: tabla DESTINO → NEXT-HOP.

- Operación: al enviar un message, se elige el siguiente salto desde la tabla; si no hay ruta, se hace fallback a Flooding.
- Ventajas: eficiente en entrega; limitación en Fase 1: topología estática sin costos dinámicos.

## **Link State Routing (LSR)**

- Idea: cada nodo anuncia a sus vecinos mediante HELLO (descubrimiento) y difunde su LSP (Link State Packet) con su vecindario.
- Base de datos: LSDB con la última secuencia de cada nodo y sus vecinos.
- Cómputo de rutas: tras integrar LSPs, se arma el grafo y se ejecuta Dijkstra local para generar la tabla NEXT-HOP.
- Temporización : HELLO cada ~3 s, LSP cada ~5 s.
- Fallback: si no hay ruta establecida aún por ejemplo, antes de converger, se reenvía por Flooding.
- Ventaja: rutas óptimas y convergencia distribuida, costo: intercambio periódico de control.

## **Infraestructura, CLI y Makefile**

- CLI: --send no levante servidor; permite tener los 4 nodos arriba y enviar desde otra terminal sin conflicto de puertos.
- Makefile: targets flood-all, dijkstra-all, lsr-all, send-\*, stop; pensado para WSL/Linux con python3.

## **Resultados y pruebas**

- Flooding: en la topología A–B–D y A–C–D, el destino D recibía duplicado; tras añadir deduplicación en la entrega por MSG\_ID, D imprime el mensaje una única vez.
- Dijkstra: entrega correcta A→D por el camino mínimo definido por la topología; no se observaron duplicados.

- LSR: tras ~8 s de estabilización HELLO/LSP, la entrega es única y por el camino óptimo. El target make send-lsr ya incluye la espera.

```

and@AM-DELL:~/Lab3Redes$ make flood-all
make ALG=flooding run-A & \
make ALG=flooding run-B & \
make ALG=flooding run-C & \
make ALG=flooding run-D ; wait
make[1]: Entering directory '/home/and/Lab3Redes'
python3 -m src.cli --id A --algo flooding --topo configs/topo-sample.json --names configs/n
ames-sample.json
make[1]: Entering directory '/home/and/Lab3Redes'
python3 -m src.cli --id B --algo flooding --topo configs/topo-sample.json --names configs/n
ames-sample.json
make[1]: Entering directory '/home/and/Lab3Redes'
python3 -m src.cli --id C --algo flooding --topo configs/topo-sample.json --names configs/n
ames-sample.json
make[1]: Entering directory '/home/and/Lab3Redes'
python3 -m src.cli --id D --algo flooding --topo configs/topo-sample.json --names configs/n
ames-sample.json
[D] INICIANDO FLOODING :: VECINOS=['B', 'C']
[B] INICIANDO FLOODING :: VECINOS=['A', 'D']
[A] INICIANDO FLOODING :: VECINOS=['B', 'C']
[C] INICIANDO FLOODING :: VECINOS=['A', 'D']
[D] MENSAJE RECIBIDO: FLOOD-OK
[D] MENSAJE RECIBIDO: FLOOD-OK
|

and@AM-DELL:~/Lab3Redes$ make send-flooding
python3 -m src.cli --id A --algo flooding --topo configs/topo-sample.json --names config
s/names-sample.json --send --to D --text "FLOOD-OK"
[A] ENVIANDO (FLOODING) -> [D]: FLOOD-OK
[A] ENVÍO COMPLETADO -> [D]
and@AM-DELL:~/Lab3Redes$

```

Imagen 1. Algoritmo Flooding ejecución y funcionamiento

```

and@AM-DELL:~/Lab3Redes$ make send-lsr
sleep 0.5 && python3 -m src.cli --id A --algo lsr --topo configs/topo-sample.json
--names configs/names-sample.json --send --to D --text "LSR-OK"
[A] ENVIANDO (LSR) -> [D]: LSR-OK
[A] ENVÍO COMPLETADO -> [D]
and@AM-DELL:~/Lab3Redes$

and@AM-DELL:~/Lab3Redes$ make lsr-all
make ALG=lsr run-A & \
make ALG=lsr run-B & \
make ALG=lsr run-C & \
make ALG=lsr run-D ; wait
make[1]: Entering directory '/home/and/Lab3Redes'
python3 -m src.cli --id A --algo lsr --topo configs/topo-sample.json --names configs/names-sample.json
make[1]: Entering directory '/home/and/Lab3Redes'
python3 -m src.cli --id B --algo lsr --topo configs/topo-sample.json --names configs/names-sample.json
make[1]: Entering directory '/home/and/Lab3Redes'
python3 -m src.cli --id C --algo lsr --topo configs/topo-sample.json --names configs/names-sample.json
make[1]: Entering directory '/home/and/Lab3Redes'
python3 -m src.cli --id D --algo lsr --topo configs/topo-sample.json --names configs/names-sample.json
[B] INICIANDO LSR :: VECINOS=['A', 'D']
[A] INICIANDO LSR :: VECINOS=['B', 'C']
[D] INICIANDO LSR :: VECINOS=['B', 'C']
[C] INICIANDO LSR :: VECINOS=['A', 'D']
[D] MENSAJE RECIBIDO: LSR-OK
[D] MENSAJE RECIBIDO: LSR-OK
|

```

Imagen 2. Algoritmo LSR ejecución y funcionamiento

```

and@AM-DELL:~/Lab3Redes$ make send-dijkstra
python3 -m src.cli --id A --algo dijkstra --topo configs/topo-sample.json --name
s configs/names-sample.json --send --to D --text "DIJKSTRA-OK"
[A] ENVIANDO (DIJKSTRA) -> [D]: DIJKSTRA-OK
[A] ENVÍO COMPLETADO -> [D]
and@AM-DELL:~/Lab3Redes$

and@AM-DELL:~/Lab3Redes$ make dijkstra-all
make ALG=dijkstra run-A & \
make ALG=dijkstra run-B & \
make ALG=dijkstra run-C & \
make ALG=dijkstra run-D ; wait
make[1]: Entering directory '/home/and/Lab3Redes'
python3 -m src.cli --id B --algo dijkstra --topo configs/topo-sample.json --names configs/names-sample.json
make[1]: Entering directory '/home/and/Lab3Redes'
python3 -m src.cli --id A --algo dijkstra --topo configs/topo-sample.json --names configs/names-sample.json
make[1]: Entering directory '/home/and/Lab3Redes'
python3 -m src.cli --id D --algo dijkstra --topo configs/topo-sample.json --names configs/names-sample.json
make[1]: Entering directory '/home/and/Lab3Redes'
python3 -m src.cli --id C --algo dijkstra --topo configs/topo-sample.json --names configs/names-sample.json
[C] INICIANDO DIJKSTRA :: VECINOS=['A', 'D']
[B] INICIANDO DIJKSTRA :: VECINOS=['A', 'D']
[D] INICIANDO DIJKSTRA :: VECINOS=['B', 'C']
[A] INICIANDO DIJKSTRA :: VECINOS=['B', 'C']
[D] MENSAJE RECIBIDO: DIJKSTRA-OK
|

```

Imagen 3. Algoritmo DIJKSTRA ejecución y funcionamiento

## Discusión

- Robustez vs eficiencia: Flooding garantiza alcance pero es costoso en reenvíos; Dijkstra es eficiente pero depende de topología conocida; LSR equilibra ambos: descubre/propaga estado y logra eficiencia tras converger.
- Detalles prácticos: la deduplicación al destino es clave en Flooding (y en fallbacks). La corrección del CLI simplificó las pruebas multi-nodo.
- Limitaciones de Fase 1: topología local, enlaces de costo uniforme y temporizadores sencillos; no se midió desempeño (latencia/uso CPU), pues el foco fue la funcionalidad.

## Conclusiones, comentarios y referencias

### Conclusiones

1. Se implementaron correctamente Flooding, Dijkstra y LSR con el modelo de mensajes JSON y arquitectura por procesos con hilos de forwarding/routing.
2. Se validó la entrega única en destino gracias a la deduplicación por MSG\_ID.
3. LSR demostró convergencia práctica con temporizadores simples y cálculo local por Dijkstra.

### Comentarios

- El Makefile y el CLI facilitaron la ejecución reproducible de escenarios.
- Para la siguiente fase, es recomendado instrumentar métricas y soportar costos de enlace.

### Referencias

- Navone, E. C. (2023, 2 agosto). *Algoritmo de la ruta más corta de Dijkstra - Introducción gráfica y detallada*. freeCodeCamp.org.  
<https://www.freecodecamp.org/espanol/news/algoritmo-de-la-ruta-mas-corta-de-dijkstra-introduccion-grafica/>
- *Flooding algorithm on a network*. (2025.). Stack Overflow.  
<https://stackoverflow.com/questions/10549604/flooding-algorithm-on-a-network>

- GeeksforGeeks. (2025, 11 julio). *Unicast routing Link State routing*. GeeksforGeeks.  
<https://www.geeksforgeeks.org/computer-networks/unicast-routing-link-state-routing/>