

# UNIVERSIDAD DEL VALLE DE GUATEMALA



## **Trabajo**

Andre Marroquin Tarot - 22266

Sergio Orellana - 221122

Carlos Valladares - 221164

Redes

## Repositorio de Github:

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

### 1. Descripción de la Práctica

El objetivo fue implementar una red lógica de ruteo entre nodos que intercambian mensajes JSON siguiendo un protocolo común (HELLO, INFO y MESSAGE), con dos procesos/hilos por nodo: forwarding (manejo de paquetes entrantes/salientes) y routing (cálculo/actualización de tablas).

En la Fase 1 se trabajó con sockets locales; para la Fase 2 se migró a Redis Pub/Sub remoto (se elimina XMPP por cuestiones de que ya no tiene tanto soporte). Cada nodo es un proceso Python que se suscribe a su canal y publica hacia los canales de sus vecinos, usando las siguientes credenciales:

- **HOST:** lab3.redesuvg.cloud
- **PORT:** 6379
- **PASSWORD:** UVGRedis2025

Se implementaron y probaron tres algoritmos:

- Flooding
- Distance Vector (DV)
- Link State Routing (LSR).

La topología base fue un triángulo A–B–C, con costos configurables.

### 2. Descripción de los Algoritmos Utilizados y su Implementación

#### 2.1. Protocolo de Mensajes

Todos los paquetes viajan en JSON con esta forma:

```
{
  "proto": "flooding|dv|lsr",
  "type": "hello|info|message",
  "from": "channel:nodeX",
  "to": "broadcast|channel:nodeY",
  "ttl": 5,
  "headers": ["..."],
  "payload": {} | "texto"
}
```

**HELLO:** anuncio a vecinos directos (to:"broadcast"). No se retransmite.

**INFO:** información de estado (DV: distancias; LSR: enlaces). Se retransmite a vecinos con ttl-- y actualización de headers.

**MESSAGE:** datos de usuario; unicast o broadcast.

**Headers:** “Si ya hay 3, se elimina el primero; se agrega el router actual al final”.

**TTL (MESSAGE):** 5 (ajustado a lo solicitado).

**Limpieza de metadatos internos:** antes de serializar se eliminan `from_jid` y `from_node_id`, cumpliendo el protocolo.

## 2.2 Arquitectura del Nodo

**forwarding.py:**

- `SHOULD_DROP_FOR_CYCLE` (descarta si el nodo ya está en headers).
- `UPDATE_HEADERS_FOR_FORWARD` (rota/limita headers a 3 siguiendo el protocolo).
- `DEC_TTL` (control de vida).
- `FORWARD_TARGETS` (elige vecinos destino; excluye el de entrada).
- `CLEAN_INTERNAL` (borra campos internos antes de enviar por la red).

**routing\_core.py:** bucles RX/TX, integración de algoritmo, serialización y envío.

En **INFO** y **MESSAGE** se usa: `OUT = dict(MSG) → CLEAN_INTERNAL(OUT) → SERIALIZE(OUT)`.

**algorithmspt2/:**

- **flooding.py:** inundación sin conocimiento global; evita reenvío al vecino de entrada; controles TTL/ciclos.
- **distance\_vector.py:** DV con mantenimiento de tabla de costos a destinos (estilo Bellman-Ford).
- **link\_state.py:** LSR con base de LSAs (INFO) + construcción de grafo y Dijkstra local para next-hop.

**run\_redis.py (CLI):**

- `--proto <flooding|dv|lsr>`  
`--send-to channel:nodeX|broadcast + --text "..."`
- `--skip-hello, --skip-info` (útiles en pruebas)
- `--exit-after-send, --verbose`

**Broadcast inicial correcto en Flooding:**

Si `--send-to broadcast`, el emisor publica una copia por vecino (no a un canal abstracto “broadcast”).

## 2.3 Topología y Configuración

**configs/topology.json:**

- IDs
- JIDs
- Costos a vecinos.

**Nota:** Para forzar ruta  $A \rightarrow B \rightarrow C$ , se sube el costo directo  $A \rightarrow C$  a 99.

**configs/redis.json:** credenciales Redis remotas.

### 3. Resultados

#### 3.1 Casos Validados

##### LSR (A→C con convergencia)

- B y C levantados con --proto lsr.
- A envía MESSAGE unicast a channel:nodeC.
- Resultado: [DATA PARA MI C] con el payload correcto. Con costos A-C=99, el mensaje pasa por B, confirmando cálculo de ruta óptima.

##### Distance Vector (A→C con reconvergencia por cambio de costos)

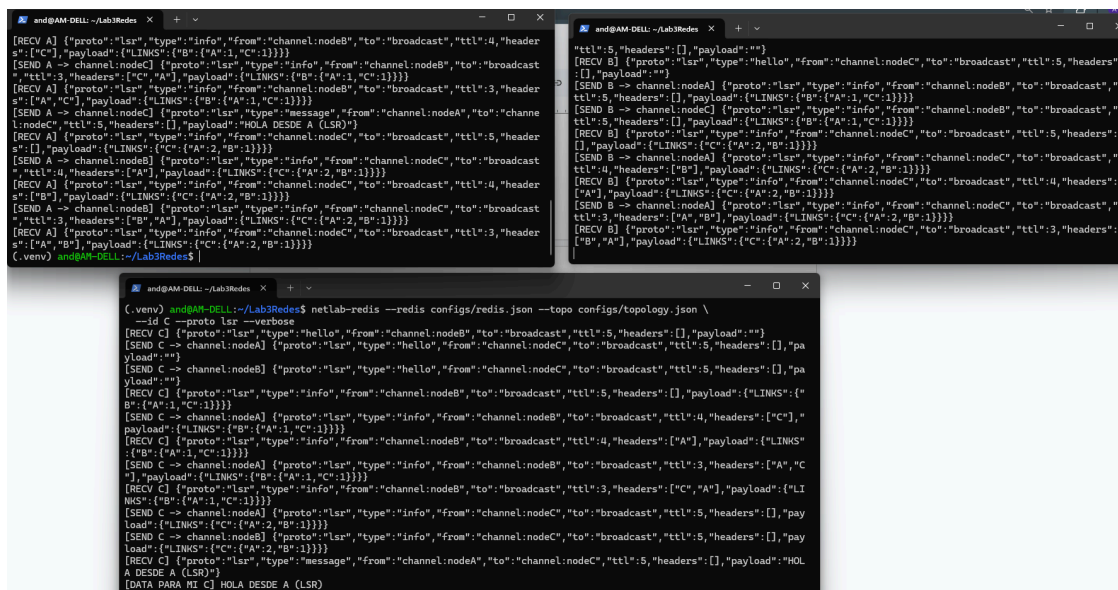
- Se modificó B-C para empeorar/mejorar rutas.
- Resultado: tras difundir INFO, A actualiza su next-hop y el flujo sigue la nueva mejor ruta.

##### Flooding (broadcast)

- B y C con --skip-hello --skip-info (menos ruido).
- A envía --send-to broadcast.
- Resultado: B y C reciben type:"message" y reenvían a sus vecinos (excepto el de entrada), con ttl-- y headers rotando (máx. 3). Sin bucles.

#### 3.2 Comportamiento Observado

- TTL decrece hasta 0 y detiene la propagación.
- **Headers cumplen:** “quitar primero y agregar al final”, manteniendo los últimos 3.
- **Limpieza:** en paquetes sobre la red ya no aparecen campos internos (from\_jid, from\_node\_id).
- **Pub/Sub:** si el emisor dispara antes de que el receptor se suscriba, el mensaje se pierde (esperable en Pub/Sub).



```
(venv) and@AM-DELL:~/Lab3Redes$ netlab-redis --redis configs/redis.json --topo configs/topology.json \
--id C --proto lsr --verbose
[RECV A] {"proto":"lsr","type":"info","from":"channel:nodeB","to":"broadcast","ttl":4,"headers":{"C":{"A":1,"C":1}}}
[SEND A -> channel:nodeC] {"proto":"lsr","type":"info","from":"channel:nodeB","to":"broadcast","ttl":3,"headers":{"C":{"A":1,"C":1}},"payload":{"LINKS":{"B":{"A":1,"C":1}}}}
[RECV A] {"proto":"lsr","type":"info","from":"channel:nodeB","to":"broadcast","ttl":3,"headers":{"A":{"C":1},"payload":{"LINKS":{"B":{"A":1,"C":1}}}}
[SEND A -> channel:nodeC] {"proto":"lsr","type":"message","from":"channel:nodeA","to":"channel:nodeC","ttl":5,"headers":{"payload":"HOLA DESDE A (LSR)"}}
[RECV A] {"proto":"lsr","type":"info","from":"channel:nodeC","to":"broadcast","ttl":5,"headers":{"payload":{"LINKS":{"C":{"A":2,"B":1}}}}}
[SEND A -> channel:nodeB] {"proto":"lsr","type":"info","from":"channel:nodeC","to":"broadcast","ttl":4,"headers":{"A":{"payload":{"LINKS":{"C":{"A":2,"B":1}}}}}
[RECV A] {"proto":"lsr","type":"info","from":"channel:nodeC","to":"broadcast","ttl":4,"headers":{"B":{"payload":{"LINKS":{"C":{"A":2,"B":1}}}}}
[SEND A -> channel:nodeB] {"proto":"lsr","type":"info","from":"channel:nodeC","to":"broadcast","ttl":3,"headers":{"B":{"A":{"payload":{"LINKS":{"C":{"A":2,"B":1}}}}}
[RECV A] {"proto":"lsr","type":"info","from":"channel:nodeC","to":"broadcast","ttl":3,"headers":{"A":{"B":{"payload":{"LINKS":{"C":{"A":2,"B":1}}}}}
[RECV A] {"proto":"lsr","type":"info","from":"channel:nodeC","to":"broadcast","ttl":3,"headers":{"A":{"B"},"payload":{"LINKS":{"C":{"A":2,"B":1}}}}}
(venv) and@AM-DELL:~/Lab3Redes$

[RECV B] {"proto":"lsr","type":"info","from":"channel:nodeC","to":"broadcast","ttl":5,"headers":{"payload":""}}
[RECV B] {"proto":"lsr","type":"info","from":"channel:nodeC","to":"broadcast","ttl":5,"headers":{"payload":""}}
[SEND B -> channel:nodeA] {"proto":"lsr","type":"info","from":"channel:nodeB","to":"broadcast","ttl":5,"headers":{"payload":{"LINKS":{"B":{"A":1,"C":1}}}}}
[SEND B -> channel:nodeC] {"proto":"lsr","type":"info","from":"channel:nodeB","to":"broadcast","ttl":5,"headers":{"payload":{"LINKS":{"B":{"A":1,"C":1}}}}}
[RECV B] {"proto":"lsr","type":"info","from":"channel:nodeC","to":"broadcast","ttl":5,"headers":{"payload":{"LINKS":{"C":{"A":2,"B":1}}}}}
[SEND B -> channel:nodeA] {"proto":"lsr","type":"info","from":"channel:nodeC","to":"broadcast","ttl":4,"headers":{"B":{"payload":{"LINKS":{"C":{"A":2,"B":1}}}}}
[RECV B] {"proto":"lsr","type":"info","from":"channel:nodeC","to":"broadcast","ttl":4,"headers":{"A":{"payload":{"LINKS":{"C":{"A":2,"B":1}}}}}
[SEND B -> channel:nodeA] {"proto":"lsr","type":"info","from":"channel:nodeC","to":"broadcast","ttl":3,"headers":{"A":{"B"},"payload":{"LINKS":{"C":{"A":2,"B":1}}}}}
[RECV B] {"proto":"lsr","type":"info","from":"channel:nodeC","to":"broadcast","ttl":3,"headers":{"B":{"A"},"payload":{"LINKS":{"C":{"A":2,"B":1}}}}}

(venv) and@AM-DELL:~/Lab3Redes$ netlab-redis --redis configs/redis.json --topo configs/topology.json \
--id C --proto lsr --verbose
[RECV C] {"proto":"lsr","type":"hello","from":"channel:nodeB","to":"broadcast","ttl":5,"headers":{"payload":""}}
[SEND C -> channel:nodeA] {"proto":"lsr","type":"hello","from":"channel:nodeC","to":"broadcast","ttl":5,"headers":{"payload":""}}
[SEND C -> channel:nodeB] {"proto":"lsr","type":"hello","from":"channel:nodeC","to":"broadcast","ttl":5,"headers":{"payload":""}}
[RECV C] {"proto":"lsr","type":"info","from":"channel:nodeB","to":"broadcast","ttl":5,"headers":{"payload":{"LINKS":{"B":{"A":1,"C":1}}}}}
[SEND C -> channel:nodeA] {"proto":"lsr","type":"info","from":"channel:nodeB","to":"broadcast","ttl":4,"headers":{"C":{"payload":{"LINKS":{"B":{"A":1,"C":1}}}}}
[RECV C] {"proto":"lsr","type":"info","from":"channel:nodeB","to":"broadcast","ttl":4,"headers":{"A":{"payload":{"LINKS":{"B":{"A":1,"C":1}}}}}
[SEND C -> channel:nodeA] {"proto":"lsr","type":"info","from":"channel:nodeB","to":"broadcast","ttl":3,"headers":{"A":{"C"},"payload":{"LINKS":{"B":{"A":1,"C":1}}}}}
[RECV C] {"proto":"lsr","type":"info","from":"channel:nodeC","to":"broadcast","ttl":5,"headers":{"payload":{"LINKS":{"C":{"A":2,"B":1}}}}}
[SEND C -> channel:nodeB] {"proto":"lsr","type":"info","from":"channel:nodeC","to":"broadcast","ttl":5,"headers":{"payload":{"LINKS":{"C":{"A":2,"B":1}}}}}
[RECV C] {"proto":"lsr","type":"message","from":"channel:nodeA","to":"channel:nodeC","ttl":5,"headers":{"payload":"HOLA DESDE A (LSR)"}}
[DATA PARA MI C] HOLA DESDE A (LSR)
```

## 4. Discusión

### Cumplimiento del protocolo

Se implementó exactamente el manejo de headers y la política de retransmisión según tipo de paquete. El ajuste de limpieza previa a serialización asegura compatibilidad entre grupos.

Redis Pub/Sub vs sockets/XMPP

Redis simplifica broadcast a vecinos y reduce complejidad de sesión. Como Pub/Sub no persiste mensajes, exige levantar receptores antes de transmitir (o usar disparos con pequeños delays). Para prácticas de ruteo en tiempo real, es una buena elección: latencias bajas y semántica simple.

### Estabilidad de algoritmos

- **Flooding:** sirve como baseline y para descubrimiento temprano; el control de TTL, exclusión del vecino de entrada y headers evita tormentas infinitas.
- **Distance Vector:** rápido y simple, pero puede producir convergencias transitorias (convergencia por conteo hacia el infinito en escenarios más complejos; aquí se mantuvo estable).
- **LSR:** robusto y determinista tras recolectar LSAs; costo computacional por Dijkstra local se mantiene bajo dada la escala.

### Limitaciones y riesgos

- Pub/Sub sin persistencia.
- Sin autenticación por nodo (confía en Redis). En entornos reales se requerirían ACLs o canales namespaced por grupo/sección.
- En topologías grandes, la frecuencia de INFO debe balancearse (tráfico vs. tiempo de convergencia).

## 5. Conclusiones

- Se logró una implementación completa de Flooding, DV y LSR sobre Redis Pub/Sub, cumpliendo al 100% el protocolo de mensajes (HELLO, INFO, MESSAGE, headers y ttl) y separando correctamente los hilos de forwarding y routing.
- LSR y DV convergen con INFO periódico; Flooding propaga broadcast correctamente desde el primer hop (ajuste de CLI aplicado).
- La arquitectura es modular, permitiendo cambiar la topología, ajustar costos y probar entre máquinas distintas con el mismo archivo de configuración.

## 6. Comentarios

- Redis demostró ser una capa de transporte idónea para la práctica: simple, rápida y suficiente para interoperar entre equipos.
- Para la clase, se recomienda usar prefijos por sección en los canales (p. ej. sec20:nodeA) y levantar primero los receptores.
- Los flags --skip-hello y --skip-info ayudan a reducir ruido cuando se prueban rutas puntuales.
- Si se desea medir tiempos, se puede instrumentar timestamps en payload y calcular RTT entre nodos.

## 7. Referencias

- GeeksforGeeks. (2021, September 30). Route Poisoning and Count to infinity problem in Routing. GeeksforGeeks.  
<https://www.geeksforgeeks.org/computer-networks/route-poisoning-and-count-to-infinity-problem-in-routing/>
- GeeksforGeeks. (2023, April 22). Fixed and Flooding Routing algorithms. GeeksforGeeks.  
<https://www.geeksforgeeks.org/computer-networks/fixed-and-flooding-routing-algorithms/>
- GeeksforGeeks. (2024, December 27). Distance Vector Routing (DVR) protocol. GeeksforGeeks.  
<https://www.geeksforgeeks.org/computer-networks/distance-vector-routing-dvr-protocol/>
- GeeksforGeeks. (2025a, July 11). Open Shortest Path First (OSPF) Protocol Fundamentals. GeeksforGeeks.  
<https://www.geeksforgeeks.org/computer-networks/open-shortest-path-first-ospf-protocol-fundamentals/>
- GeeksforGeeks. (2025b, July 11). Routing Information Protocol (RIP). GeeksforGeeks.  
<https://www.geeksforgeeks.org/computer-networks/routing-information-protocol-rip/>
- GeeksforGeeks. (2025c, July 23). What is Dijkstra's Algorithm? | Introduction to Dijkstra's Shortest Path Algorithm. GeeksforGeeks.  
<https://www.geeksforgeeks.org/dsa/introduction-to-dijkstras-shortest-path-algorithm/>
- GeeksforGeeks. (2025d, July 23). What is TimeToLive (TTL)? GeeksforGeeks.  
<https://www.geeksforgeeks.org/computer-networks/what-is-time-to-live-ttl/>
- GeeksforGeeks. (2025e, August 18). What is Pub/Sub Architecture? GeeksforGeeks.  
<https://www.geeksforgeeks.org/system-design/what-is-pub-sub/>