Universidad del Valle de Guatemala

Facultad de ingeniería

Redes

Catedrático: Juan Canteo

Laboratorio #3 – Algoritmos de enrutamiento

Andre Jo 22199

Gabriel Paz 221087

Christian Echeverría 221441

Guatemala, Septiembre de 2025

En este laboratorio se buscó comprender y aplicar diferentes algoritmos de enrutamiento en un entorno simulado, con el fin de analizar cómo se construyen y actualizan las tablas de enrutamiento, y cómo estas permiten el envío eficiente de mensajes en una red.

La práctica se dividió en dos fases principales. En la primera fase se implementaron los algoritmos de enrutamiento de manera local utilizando sockets TCP, lo que permitió validar el correcto funcionamiento de las implementaciones y observar cómo los nodos podían descubrir a sus vecinos y generar sus tablas de enrutamiento iniciales.

En la segunda fase, en lugar de utilizar un servidor XMPP como medio de comunicación, se decidió emplear Redis como tecnología de mensajería y almacenamiento. Redis, al funcionar como un servidor de base de datos en memoria con soporte para publicación/suscripción (pub/sub), permitió que cada nodo pudiera enviar y recibir mensajes de forma eficiente. En este esquema, cada nodo se suscribía a los canales correspondientes a sus vecinos y publicaba en los canales destinados a otros nodos, emulando así la dinámica de una red distribuida.

Gracias al uso de Redis, fue posible mantener la interoperabilidad entre los distintos nodos, manejar paquetes de datos, mensajes de descubrimiento (HELLO/PING), así como la propagación de información de tablas de enrutamiento. De esta manera, se logró simular una red dinámica en la que los algoritmos implementados (Flooding, Distance Vector y Link State Routing) pudieran ser probados en condiciones similares a una red real. A continuación, se presentarán las implementaciones de los algoritmos propuestos.

Algoritmo Link State Routing (LSR)

El algoritmo LSR implementa el protocolo de estado de enlaces, donde cada nodo mantiene una base de datos completa de la topología de red conocida como LSDB (Link State Database). Este enfoque permite que cada nodo tenga una vista global de la red y pueda calcular independientemente las rutas óptimas hacia todos los destinos utilizando el algoritmo de Dijkstra.

El proceso operativo del LSR comienza con el descubrimiento de vecinos mediante el envío periódico de paquetes HELLO que permiten a cada nodo identificar sus vecinos directos y mantener información actualizada sobre su disponibilidad. Una vez

establecida la conectividad básica, los nodos intercambian paquetes INFO que contienen información sobre el estado de sus enlaces locales, incluyendo identificadores de vecinos y costos asociados.

La implementación mantiene una LSDB que se actualiza cada vez que se recibe información de estado de enlaces de otros nodos. Cuando ocurren cambios en la topología, ya sea por la caída de un enlace o la aparición de nuevas conexiones, el sistema recalcula automáticamente las rutas utilizando Dijkstra sobre el grafo completo almacenado en la LSDB. Para optimizar el rendimiento, se implementa un mecanismo de debouncing que evita recálculos excesivos cuando se reciben múltiples actualizaciones en rápida sucesión.

El LSR incluye características avanzadas como la detección automática de vecinos caídos mediante timeouts configurables en los paquetes HELLO, purga automática de información obsoleta en la LSDB, y reconfiguración inmediata de rutas cuando se detectan cambios topológicos. Esta implementación garantiza convergencia rápida y rutas óptimas, aunque requiere mayor overhead de memoria y comunicación comparado con otros algoritmos.

Algoritmo Distance Vector Routing (DVR)

El DVR implementa una versión distribuida del algoritmo de Bellman-Ford, donde cada nodo mantiene un vector de distancias que contiene la distancia mínima conocida hacia cada destino, junto con el próximo salto necesario para alcanzarlo. A diferencia del LSR, los nodos no mantienen una vista completa de la topología, sino que toman decisiones de enrutamiento basadas únicamente en información local y vectores recibidos de vecinos.

El proceso inicia con cada nodo conociendo únicamente las distancias hacia sus vecinos directos, típicamente con costo 1. Periódicamente, cada nodo anuncia su vector completo de distancias a todos sus vecinos. Cuando un nodo recibe el vector de distancias de un vecino, aplica la ecuación de Bellman-Ford para determinar si existe una ruta más corta hacia algún destino pasando por ese vecino. Si se encuentra una mejora, actualiza su vector local y propaga la información en el siguiente anuncio.

La implementación incluye mecanismos críticos para prevenir problemas inherentes al DVR como el count-to-infinity. Se implementa split-horizon con poison reverse, donde un nodo anuncia distancia infinita hacia destinos que alcanza a través del vecino receptor del anuncio. Adicionalmente, se implementa expiración automática de entradas que no se actualizan dentro de un período configurable, permitiendo que el algoritmo se recupere de caídas de nodos.

El DVR ofrece menor complejidad computacional y de memoria comparado con LSR, ya que cada nodo solo mantiene su vector de distancias sin necesidad de almacenar la

topología completa. Sin embargo, la convergencia puede ser más lenta, especialmente en topologías complejas o ante cambios frecuentes en la red.

Algoritmo Dijkstra Estático

La implementación de Dijkstra estático representa una aproximación determinística donde las rutas se calculan una única vez al inicio utilizando una topología predefinida y fija. Este algoritmo lee la configuración completa de la red desde archivos de configuración y aplica el algoritmo de Dijkstra clásico para determinar las rutas óptimas desde el nodo local hacia todos los demás nodos.

El proceso es straightforward: al inicializar, el servicio construye un grafo no dirigido donde cada enlace tiene peso unitario basado en la configuración estática. Posteriormente ejecuta Dijkstra desde el nodo local para calcular el árbol de rutas mínimas y extrae el próximo salto para cada destino. Esta información se instala directamente en la tabla de enrutamiento y permanece inalterada durante toda la ejecución.

La principal ventaja de este enfoque es la ausencia completa de overhead de comunicación para el protocolo de enrutamiento, ya que no requiere intercambio de paquetes de control. Las rutas son óptimas por construcción y la convergencia es instantánea. Sin embargo, esta aproximación sacrifica completamente la adaptabilidad, ya que no puede responder a cambios dinámicos en la topología como caídas de enlaces o nodos.

Algoritmo de Flooding

El algoritmo de flooding implementa la estrategia más simple de diseminación de información, donde cada paquete se reenvía a todos los vecinos excepto al nodo desde el cual se recibió. Aunque conceptualmente simple, la implementación incluye mecanismos sofisticados de control para prevenir problemas como loops infinitos y congestión excesiva.

El flooding controlado implementado utiliza varios mecanismos de protección. Cada paquete incluye un campo TTL (Time To Live) que se decrementa en cada salto, descartándose cuando alcanza cero para limitar la propagación. Los paquetes mantienen un header con el path recorrido, permitiendo detectar y prevenir ciclos. Adicionalmente, se implementa de-duplicación mediante identificadores únicos de mensaje, evitando el procesamiento repetido del mismo paquete.

A pesar de su simplicidad conceptual, el flooding garantiza entrega del mensaje si existe al menos una ruta hacia el destino, incluso en topologías altamente dinámicas. No requiere mantenimiento de tablas de enrutamiento ni intercambio de información de control. Sin embargo, genera significativo overhead de red al enviar cada paquete por

múltiples rutas, y su escalabilidad está limitada por el crecimiento exponencial del tráfico con el tamaño de la red.

Servicios de Soporte y Infraestructura

El ForwardingService actúa como el motor central de procesamiento de paquetes, proporcionando funcionalidad común independiente del algoritmo de enrutamiento específico. Este servicio recibe todos los paquetes entrantes, realiza validación de esquemas utilizando Pydantic, aplica reglas de anti-duplicación y anti-loop, y delega las decisiones de enrutamiento al algoritmo activo. Maneja tres tipos principales de paquetes: HELLO para descubrimiento y mantenimiento de vecinos, INFO para intercambio de información de enrutamiento, y MESSAGE para datos de usuario que requieren reenvío.

La gestión de estado se centraliza en el módulo State, que mantiene estructuras de datos críticas incluyendo información de vecinos directos con timestamps de último HELLO recibido, la LSDB utilizada por LSR, tablas de enrutamiento calculadas por cada algoritmo, y un cache TTL para prevenir procesamiento duplicado de mensajes. Este módulo proporciona funcionalidades transversales como construcción de grafos de red desde la LSDB, cálculo de costos mínimos, detección automática de vecinos inactivos mediante timeouts, y capacidades de persistencia de estado.

El sistema de comunicación soporta múltiples protocolos de transporte de manera transparente. La implementación Redis utiliza el patrón publish/subscribe para comunicación eficiente en entornos locales, mientras que el soporte XMPP permite comunicación distribuida siguiendo estándares abiertos. Esta abstracción permite que los algoritmos de enrutamiento operen independientemente del mecanismo de transporte subyacente.

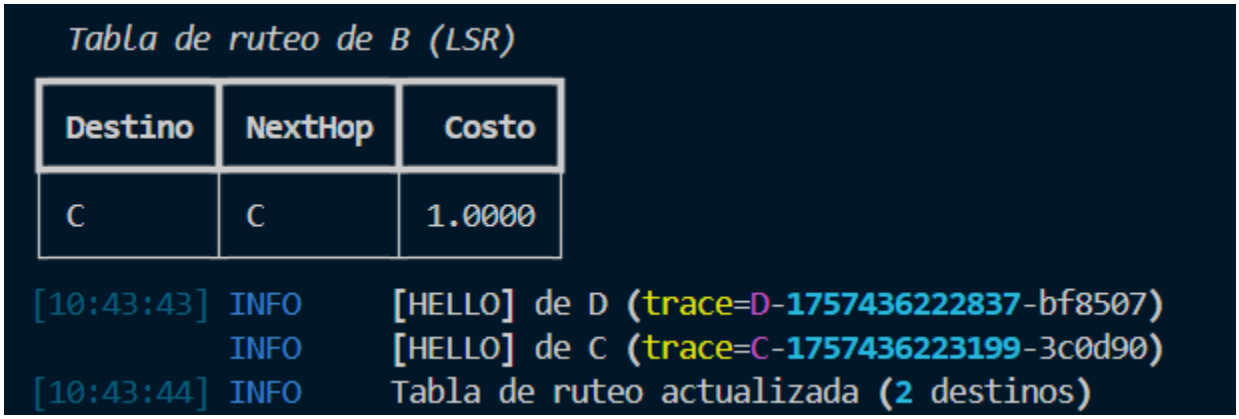
El sistema incorpora múltiples optimizaciones para mejorar rendimiento y robustez. El mecanismo de debouncing en LSR previene recálculos excesivos cuando se reciben múltiples actualizaciones topológicas en rápida sucesión, agrupando cambios y procesándolos en batch. El cache de mensajes vistos con TTL automático previene procesamiento duplicado mientras limita el consumo de memoria mediante expiración automática de entradas obsoletas.

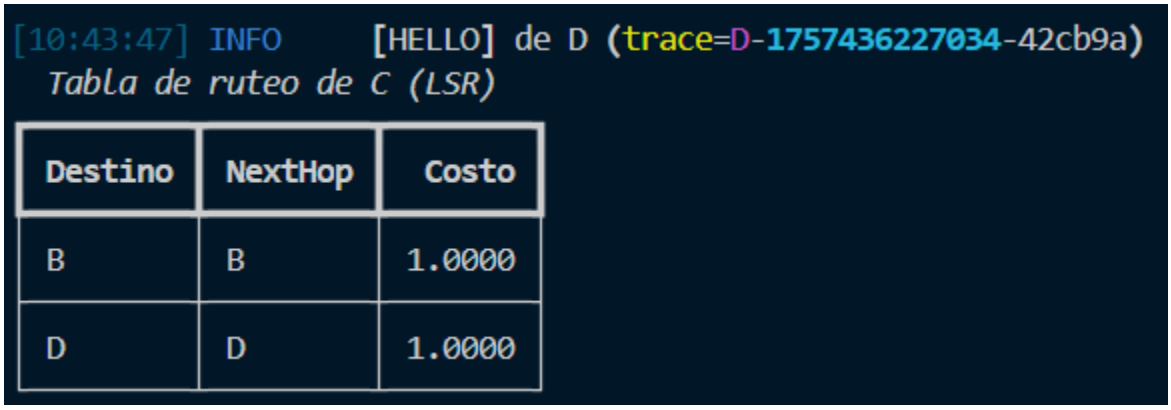
La tolerancia a fallos se implementa mediante timeouts configurables para detección de caídas de vecinos, purga automática de información obsoleta en bases de datos de estado, y reconfiguración automática de rutas cuando se detectan cambios. El sistema incluye logging detallado y capacidades de monitoreo en tiempo real, permitiendo observar tablas de enrutamiento, estado de vecinos, y estadísticas de paquetes procesados.

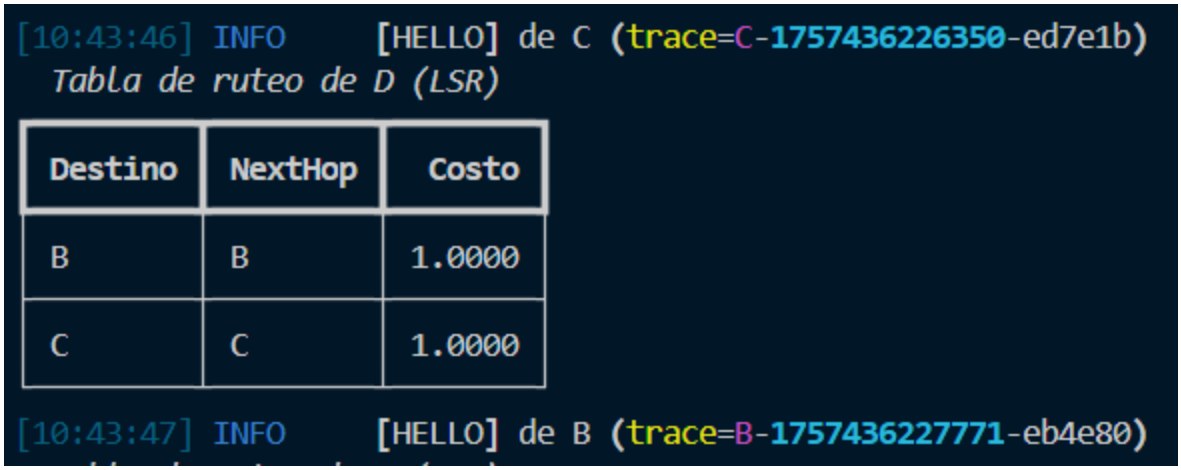
La configurabilidad extensiva permite ajustar intervalos de anuncio, timeouts de detección de fallos, y parámetros específicos de cada algoritmo mediante archivos de configuración externos. Esta flexibilidad facilita la experimentación con diferentes parámetros operativos y la adaptación a diversos escenarios de red.

Esta implementación modular y completa proporciona una plataforma robusta para comparar eficazmente el comportamiento, rendimiento, y características de convergencia de diferentes algoritmos de enrutamiento bajo condiciones controladas y reproducibles.

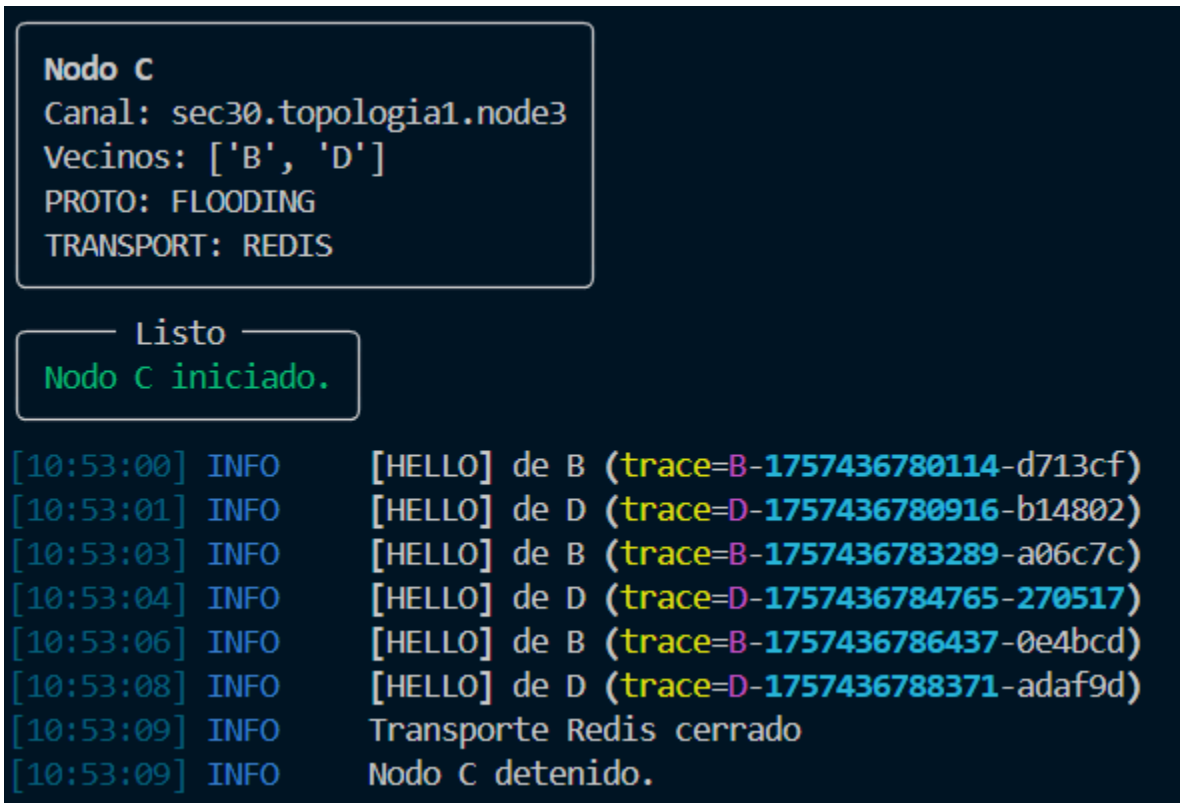
Resultados







Utilizando LSR, cada nodo mantiene una visión global de la red construida a partir de la información de estado de enlaces, y calcular rutas óptimas con Dijkstra. La tabla que se observa, es la representación donde los vecinos son los primeros a recibir el mensaje y luego se irá completando conforme se propague la información de la red.



En la prueba realizada con el algoritmo de Flooding, se observa que el Nodo C al recibir un mensaje inicia el proceso de reenvío hacia todos sus vecinos directos, en este caso B y D. Este comportamiento es característico del algoritmo de flooding, en el cual cada nodo replica el mensaje a todos los enlaces disponibles, excepto hacia el nodo desde el cual lo recibió. Posteriormente, dichos vecinos continúan propagando el mensaje hacia sus propios vecinos, lo que asegura que la información se difunda por toda la red.

En la prueba realizada con el Nodo B utilizando el algoritmo de Dijkstra (LSR), se observa que la tabla de enrutamiento refleja tanto los vecinos directos como los destinos alcanzables a través de ellos. De esta manera, B identifica rutas directas hacia A, C y D con un costo de 1.0, mientras que para alcanzar al nodo E determina que la mejor opción es hacerlo a través de D, con un costo acumulado de 2.0. Esto evidencia la capacidad del algoritmo de Dijkstra para calcular caminos óptimos en función de la información global de la red, permitiendo que cada nodo seleccione de manera eficiente el next hop hacia cualquier destino, a diferencia del flooding que simplemente reenvía los mensajes sin calcular rutas mínimas.

Conclusiones