Implementação do Algoritmo *Link State* em Redes de Computadores

Relatório Técnico

Daniel Rodrigues de Sousa

Universidade Federal do Piauí
Campus Senador Helvídio Nunes de Barros

Picos - PI
2024

Links Complementares

A seguir, estão disponibilizados os links que complementam a entrega deste trabalho, oferecendo acesso aos recursos adicionais desenvolvidos durante o projeto. O repositório no *GitHub* contém todo o código-fonte, *scripts* de teste, documentação e instruções de uso, permitindo a reprodução e análise da implementação. Já o vídeo no *YouTube* apresenta uma demonstração prática do sistema em funcionamento, incluindo a explicação da topologia utilizada, os testes executados e os principais resultados obtidos.

- **Repositório GitHub:** `https://github.com/DanielRodri87/link_state_network`
- **Demonstração YouTube:** `https://youtu.be/qkxRrY2eHKk`

Introdução

Em redes de computadores, o roteamento baseado no estado de enlace (*link-state*) é uma abordagem na qual cada nó constrói uma visão completa da topologia da rede com base em informações trocadas entre vizinhos. Essa visão global permite o cálculo de rotas otimizadas utilizando algoritmos como o de Dijkstra. Diferente dos protocolos por vetor de distância, que mantêm apenas dados locais, os protocolos *link-state* propagam informações sobre o estado dos enlaces por toda a rede, garantindo maior precisão e rápida convergência diante de alterações topológicas.

Nas redes móveis ad hoc (MANETs), essa abordagem enfrenta desafios significativos devido à alta mobilidade e instabilidade dos enlaces. A proposta de [Chen et al. 2004] introduz um protocolo de roteamento sob demanda, multi-caminho e baseado em estado de enlace, desenvolvido para atender aos requisitos de Qualidade de Serviço (QoS) mesmo em contextos dinâmicos. A técnica permite a manutenção de múltiplas rotas viáveis e a rápida adaptação a mudanças na topologia da rede, aumentando a confiabilidade na transmissão de dados.

Além disso, a modelagem estocástica de carga dinâmica de rede, como proposta por [Osorio et al. 2011], contribui para uma avaliação mais realista do desempenho de protocolos *link-state*. O modelo permite derivar distribuições probabilísticas do estado dos enlaces, reconhecendo a variabilidade do tráfego em cenários reais. Complementarmente, o estudo de [Park and Corson 1998] compara o algoritmo TORA com um modelo ideal *link-state*, revelando diferenças de desempenho, consumo de recursos e tempo de convergência entre as abordagens.

Neste trabalho, é apresentado um estudo aprofundado sobre a implementação e análise do algoritmo de roteamento *link-state*, com ênfase em sua aplicação em redes de média escala. A implementação foi baseada no protocolo *Open Shortest Path First* (*OSPF*), possibilitando a simulação realista do comportamento de roteadores em um ambiente dinâmico. O simulador, desenvolvido em Python com bibliotecas especializadas, permitiu avaliar métricas como tempo de convergência, eficiência das rotas e sobrecarga de controle. Os resultados obtidos demonstram a viabilidade da proposta e sua eficácia na adaptação a mudanças topológicas, contribuindo para a compreensão prática dos mecanismos de roteamento modernos.

Metodologia

Este trabalho foi conduzido com base em uma abordagem sistemática para a implementação e análise do algoritmo *Link State* em redes computacionais. A metodologia foi cuidadosamente estruturada para garantir consistência nos resultados, contemplando desde a configuração do ambiente de desenvolvimento até a definição das métricas de avaliação utilizadas na análise de desempenho.

3.1 Ambiente de Desenvolvimento

A implementação foi realizada utilizando a linguagem de programação Python, na versão 3.8, escolhida pela sua clareza sintática e pela ampla gama de bibliotecas voltadas ao tratamento de grafos e análise de redes. Dentre essas bibliotecas, destaca-se o uso do *NetworkX* para a modelagem e manipulação da topologia de rede, bem como do *Matplotlib* para a geração de visualizações gráficas que auxiliam na interpretação dos resultados. O desenvolvimento seguiu o paradigma de programação orientada a objetos, promovendo a modularização e a reutilização do código. Para garantir a consistência das dependências e a reprodutibilidade dos experimentos, foi configurado um ambiente virtual Python (*venv*), no qual todas as bibliotecas necessárias foram instaladas de forma controlada.

3.2 Arquitetura da Rede

O projeto foi estruturado de forma modular, organizando os componentes em diretórios específicos que refletem suas responsabilidades. No diretório raiz *link_state_network*, encontram-se os subdiretórios principais: *docker* para os arquivos de configuração e execução dos contêineres, *generate_compose* para os geradores de topologia, e *docs* para a documentação do projeto. O diretório *docker* é subdividido em *router* e *host*, cada um contendo seus respectivos arquivos de configuração e classes de implementação. No diretório *router*, a pasta *class_net* agrupa as classes principais do sistema: *LSAManager* para gerenciamento de anúncios de estado de enlace, *VizinhosManager* para monitoramento de vizinhos, *GerenciadorDeRotas* para cálculo de rotas, e *AtualizadorDeRotas* para atualização das tabelas de roteamento. A pasta *test* contém os scripts de teste para verificação de conectividade, análise de rotas e visualização de topologia. O diretório *generate_compose* contém os arquivos necessários para geração dinâmica das diferentes topologias de rede suportadas (anel, estrela, árvore e linha), incluindo o gerador de arquivos YAML e templates para o *Docker Compose*.

A Figura 3.1 contém a estrutura de pastas usada no projeto. Que pode ser consultado via GitHub anexado.

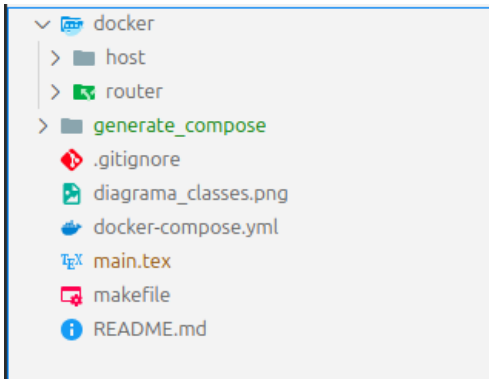

Figure 3.1: Representação da arquitetura

3.3 Implementação do Algoritmo

A implementação do algoritmo foi baseada nos princípios do protocolo *Open Shortest Path First* (*OSPF*), uma das abordagens mais conhecidas para roteamento baseado em estado de enlace. Para permitir a troca de informações entre os roteadores, foi utilizado o protocolo UDP, com comunicação assíncrona na porta 5000. A principal estrutura trocada entre os roteadores são os *Link State Advertisements* (*LSAs*), que contêm dados essenciais como o identificador único do roteador de origem, seu endereço IP, um número de sequência (para controle de atualizações) e uma lista dos vizinhos ativos, junto com o custo associado a cada ligação.

O componente chamado *LSAManager* é responsável por enviar *LSAs* periodicamente a cada 0,5 segundo, além de receber os *LSAs* vindos de outros roteadores. Quando um roteador recebe um *LSA*, ele verifica se as informações são mais recentes — comparando o número de sequência — e, se forem, atualiza sua base de dados de estado de enlace (*LSDB*). Em seguida, o *LSA* é repassado para os demais vizinhos, seguindo o princípio da inundação controlada. Esse mecanismo garante que todos os roteadores tenham uma visão consistente da topologia da rede, como previsto pelo protocolo *OSPF*.

3.4 Topologias

O sistema foi testado com quatro tipos principais de topologias de rede, cada uma com características e aplicações específicas:

3.4.1 Topologia em Linha

A topologia em linha (ou barramento) conecta os roteadores sequencialmente, onde cada roteador intermediário possui exatamente duas conexões com seus vizinhos imediatos, enquanto os roteadores das extremidades possuem apenas uma conexão. Esta configuração é caracterizada por sua simplicidade e economia de links, sendo adequada para redes que seguem um caminho linear, como em corredores de prédios ou ao longo de rodovias. A principal desvantagem é que uma falha em qualquer roteador intermediário divide a rede em duas partes desconectadas.

3.4.2 Topologia em Árvore

Na topologia em árvore, os roteadores são organizados hierarquicamente, começando com um roteador raiz que se ramifica para outros roteadores em níveis subsequentes. Cada roteador (exceto a raiz) possui exatamente uma conexão com um roteador do nível superior (pai) e pode ter múltiplas conexões com roteadores do nível inferior (filhos). Esta estrutura é particularmente útil em redes corporativas ou campus universitários, onde existe uma hierarquia natural na distribuição do tráfego. A topologia facilita o gerenciamento e a escalabilidade da rede, embora também seja vulnerável a falhas no roteador raiz.

3.4.3 Topologia em Estrela

A topologia em estrela centraliza todas as conexões em um único roteador central, com os demais roteadores conectados diretamente apenas a ele. Esta configuração oferece simplicidade no gerenciamento e facilidade na detecção de falhas, pois cada roteador periférico opera independentemente dos outros. É comum em redes locais onde um roteador central atua como ponto de concentração. A principal limitação é a dependência do roteador central, que se torna um ponto único de falha para toda a rede.

3.4.4 Topologia em Anel

Na topologia em anel, cada roteador conecta-se exatamente a dois outros roteadores, formando um circuito fechado. Esta configuração oferece redundância natural, pois existem sempre dois caminhos possíveis entre quaisquer dois roteadores. A topologia em anel é frequentemente utilizada em redes metropolitanas e *backbones*, onde a redundância é crucial. Uma característica interessante é que o tráfego pode ser distribuído em ambas as direções do anel, otimizando o uso da largura de banda e oferecendo caminhos alternativos em caso de falha em algum enlace.

3.4.5 Exemplo de Uso

Para a elaboração deste relatório, foi utilizada uma topologia em linha composta por 10 roteadores, sendo que cada roteador está conectado a 2 *hosts*. Dessa forma, a rede totaliza 20 *hosts* distribuídos entre os 10 roteadores. A Figura 3.2 ilustra um exemplo simplificado dessa configuração, apresentando apenas 4 roteadores com 2 *hosts* conectados a cada um, a fim de facilitar a visualização da estrutura adotada.

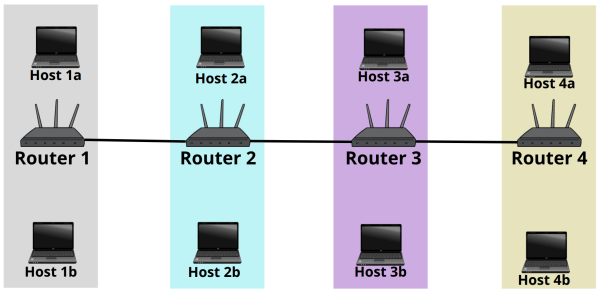

Figure 3.2: Exemplo de Uso com 4 Roteadores com 2 *Hosts*

3.5 Diagrama de Classes

A arquitetura do sistema foi organizada em cinco classes principais, que atuam de forma coordenada para simular o funcionamento de um roteador *OSPF*. A classe central é a *RoteadorApp*, responsável por inicializar os componentes e gerenciar as múltiplas *threads* que realizam as tarefas simultâneas. Um dos principais componentes é o *LSAManager*, que cuida da criação, envio e recebimento dos *LSAs* (*Link State Advertisements*) por meio de *sockets* UDP. Esses *LSAs* contêm informações sobre a topologia da rede e são fundamentais para a construção da base de dados de estado de enlace (*LSDB*).

Além disso, o *VizinhosManager* monitora os roteadores vizinhos ativos, utilizando *pings* ICMP para verificar sua disponibilidade e mantendo uma lista atualizada de conexões válidas. Com base nessas informações, o *GerenciadorDeRotas* aplica o algoritmo de Dijkstra para calcular as rotas mais eficientes na rede. Por fim, o *AtualizadorDeRotas* realiza a atualização prática das rotas no sistema operacional, utilizando comandos como *ip route* para refletir as mudanças no roteamento real. Juntas, essas classes formam uma estrutura modular e eficiente para simular o comportamento de um roteador *OSPF*.

3.6 Rede de Petri

A seguir, a Figura 3.3 ilustra a arquitetura geral do sistema e a interação entre suas principais classes:

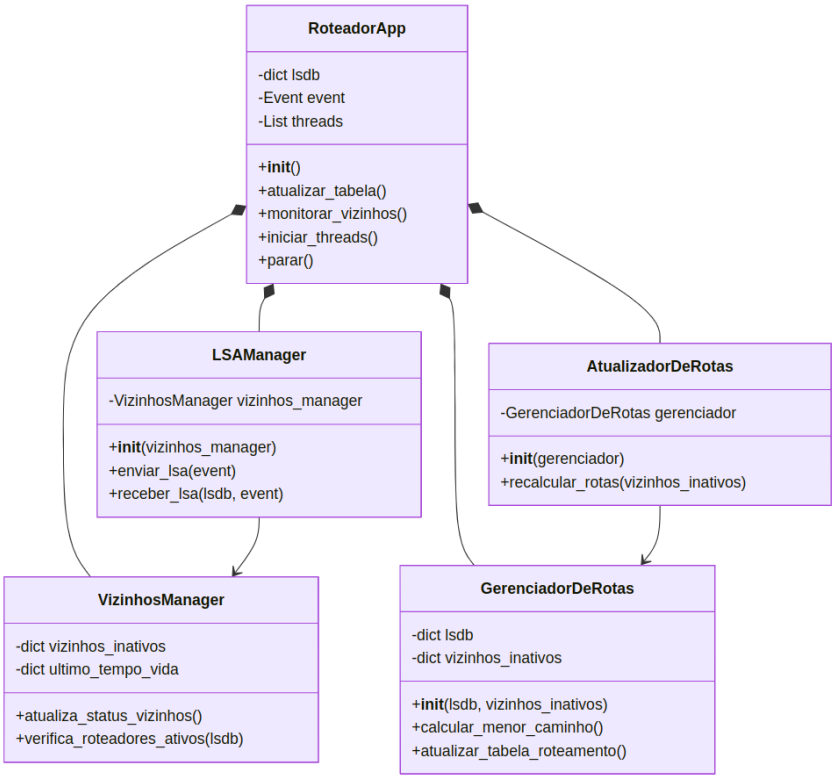

Figure 3.3: Representação da arquitetura

Resultados

Nesta seção, serão apresentados os resultados obtidos durante a fase de testes, bem como a análise das questões propostas pelo professor. As reflexões abordarão os seguintes pontos: quais são os limites e o nível de estresse suportado pelo sistema; quais as principais vantagens da abordagem adotada; se é possível que um *host* realize um *ping* com sucesso para outro *host*; e, por fim, qual foi a estratégia de implementação utilizada e os motivos que justificam essa escolha.

4.1 Limiares/Stress

Para responder a essa pergunta, optei por analisar duas métricas fundamentais: tempo e taxa de acerto. Essas métricas refletem diretamente os objetivos centrais em redes de computadores — alcançar comunicações cada vez mais rápidas e eficientes. O tempo representa a agilidade do sistema em propagar informações e estabelecer rotas, enquanto a taxa de acerto avalia a precisão das decisões de roteamento, indicando se os pacotes estão sendo encaminhados corretamente ao destino final.

4.1.1 Tempo de Resposta Entre Roteadores

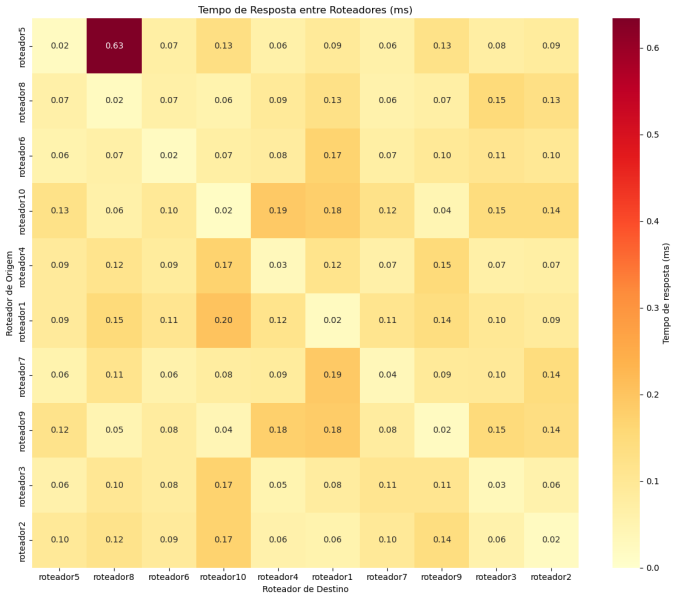

Figure 4.1: Tempo de Resposta Entre Roteadores

A Figura 4.1 apresentada é um mapa de calor que representa o tempo de resposta entre pares de roteadores em milissegundos (ms). Cada célula indica o tempo médio de comunicação entre um roteador de origem (eixo vertical) e um roteador de destino (eixo horizontal). Em geral, os tempos de resposta são baixos, variando predominantemente

entre 0.02 ms e 0.20 ms, o que demonstra um desempenho eficiente na maioria das comunicações. No entanto, há um ponto de destaque em vermelho entre o *roteador5* e o *roteador8*, com um tempo significativamente mais alto (0.63 ms), sugerindo um gargalo ou instabilidade momentânea naquela conexão.

Outro aspecto relevante a ser analisado diz respeito aos valores presentes na diagonal principal da matriz de tempos de resposta, os quais representam o *ping* realizado pelos roteadores para si mesmos. Esses tempos são significativamente menores em comparação com os tempos de comunicação entre diferentes roteadores, o que é esperado, uma vez que não envolvem tráfego real na rede física. Esse comportamento confirma que o sistema está operando corretamente, já que as respostas locais ocorrem com latência mínima, refletindo a eficiência interna de processamento e encaminhamento. Além disso, essa observação serve como uma referência importante para avaliar o desempenho relativo das demais comunicações na rede.

4.1.2 Taxa de Acertividade entre os Roteadores

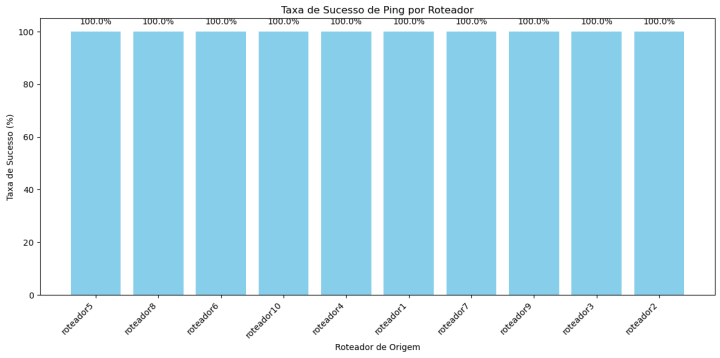

Figure 4.2: Taxa de Acertividade entre os Roteadores

O gráfico apresentado mostra a taxa de sucesso dos testes de *ping* realizados por cada roteador da rede, evidenciando que todos os roteadores atingiram 100% de sucesso. Esse resultado indica que não houve perda de pacotes durante a comunicação entre os nós, o que reforça a confiabilidade e estabilidade da rede implementada. Além disso, demonstra que o algoritmo de roteamento está operando corretamente, garantindo a entrega eficiente das mensagens entre os dispositivos conectados.

4.2 Vantagens e Desvantagens

4.3 *Ping* entre *Hosts*

Para verificar a conectividade fim a fim na rede, foi desenvolvido um *script* de teste específico, que realiza *pings* entre os *hosts* conectados aos roteadores. Esse teste é executado por meio do seguinte comando:

```
make ping_host
```

Esse comando automatiza o envio de requisições ICMP entre os *hosts*, permitindo avaliar a comunicação direta através da infraestrutura de roteadores. A seguir, apresenta-se a saída gerada pela execução do script.

Table 4.1: Vantagens e Desvantagens do Sistema

Vantagens	**Desvantagens**
Eficiência: O algoritmo de Dijkstra utilizado para calcular as rotas é eficiente e garante a escolha do caminho mais curto.	**Complexidade:** A implementação do algoritmo de Dijkstra e o gerenciamento de vizinhos podem aumentar a complexidade do sistema.
Robustez: O sistema é capaz de lidar com falhas de roteadores e *hosts*, garantindo a continuidade do serviço.	**Atualização de Rotas:** A atualização constante das rotas pode gerar um *overhead* significativo, especialmente em redes com alta mobilidade.
Atualização Dinâmica: O sistema é capaz de atualizar suas rotas dinamicamente, adaptando-se a mudanças na topologia da rede.	**Limitações de Escalabilidade:** Embora o sistema seja escalável, a adição de muitos roteadores e *hosts* pode levar a um aumento significativo na complexidade e no tempo de processamento.

4.3.1 Saída do Comando *Ping Host*

Figure 4.3: Resultado do teste de *ping* entre os *hosts* conectados à rede

A Figura 4.3 demonstra que os pacotes foram transmitidos com sucesso entre os diferentes *hosts* da rede, evidenciando o correto funcionamento do algoritmo de roteamento e a estabilidade da topologia implementada.

4.4 Protocolo de Comunicação Utilizado

O protocolo adotado para a comunicação entre os *hosts* e os roteadores foi o UDP (*User Datagram Protocol*). Trata-se de um protocolo de transporte leve, que permite a troca de datagramas entre dispositivos de forma rápida e eficiente. Ao contrário do TCP (*Transmission Control Protocol*), o UDP não estabelece uma conexão previamente nem garante a entrega dos pacotes, o que o torna menos confiável, porém substancialmente mais ágil.

Esse protocolo é amplamente empregado em aplicações sensíveis à latência, como transmissões de áudio e vídeo em tempo real, jogos online e sistemas de controle em tempo real, justamente por evitar a sobrecarga de verificação de entrega e controle de fluxo presentes no TCP.

Justificativa para o Uso do UDP

A escolha do UDP neste projeto fundamenta-se na necessidade de comunicação de baixa latência entre os roteadores e *hosts*. Em um ambiente de simulação de rede, como o aqui proposto, a agilidade na transmissão de mensagens é essencial para representar com fidelidade o comportamento dinâmico da topologia. O UDP proporciona exatamente essa característica, ao permitir o envio assíncrono de pacotes com o mínimo de sobrecarga. Além disso, sua simplicidade de implementação e configuração o torna especialmente adequado para o escopo deste projeto, onde o foco está na eficiência e no desempenho do sistema, e não na confiabilidade absoluta da entrega de pacotes.

Bibliography

[Chen et al. 2004] Chen, Y.-S., Tseng, Y.-C., Sheu, J.-P., and Kuo, P.-H. (2004). An on-demand, link-state, multi-path qos routing in a wireless mobile ad-hoc network. *Computer Communications*, 27(1):27–40.

[Osorio et al. 2011] Osorio, C., Flötteröd, G., and Bierlaire, M. (2011). Dynamic network loading: a stochastic differentiable model that derives link state distributions. *Procedia-Social and Behavioral Sciences*, 17:364–381.

[Park and Corson 1998] Park, V. D. and Corson, M. S. (1998). A performance comparison of the temporally-ordered routing algorithm and ideal link-state routing. In *Proceedings Third IEEE Symposium on Computers and Communications. ISCC'98.(Cat. No. 98EX166)*, pages 592–598. IEEE.