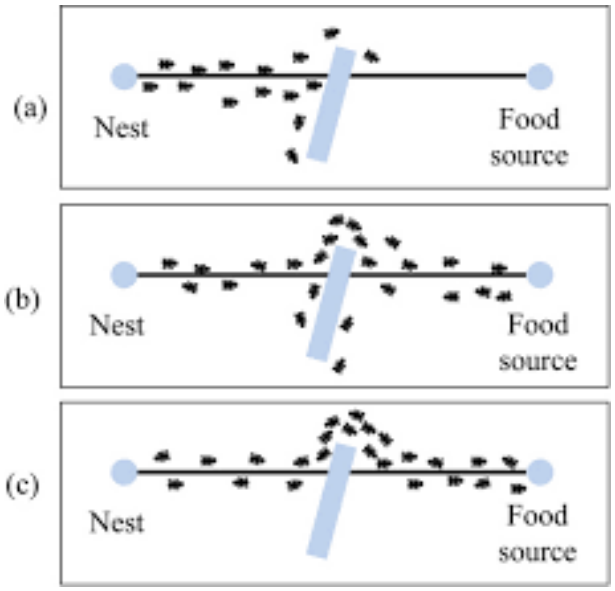
Algoritmo de Roteamento (ACO)

O algoritmo utiliza Ant Colony Optimization (ACO) inspirado no comportamento de formigas. Cada roteador mantém uma tabela de feromônios que representa a qualidade dos caminhos para cada destino. A escolha do próximo salto é probabilística, considerando níveis de feromônio e latência medida via ping. Podemos ajustar os pesos conforme desejamos dessa heuristica.



Protocolo de Roteamento

O protocolo utiliza mensagens UDP para troca de informações entre roteadores. Mensagens 'Hello' identificam vizinhos ativos, enquanto mensagens 'Update' compartilham tabelas de feromônios. A evaporação de feromônios e a remoção de rotas obsoletas ocorrem periodicamente.

- **Mensagens Hello**: Enviadas a cada HELLO_INTERVAL (5s) para descobrir vizinhos ativos.
- **Mensagens Update**: Enviadas a cada UPDATE_INTERVAL (30s) com a tabela de feromônios.
- **Verificação de Rotas**: Executada a cada CHECK_INTERVAL (180s) para evaporar feromônios e remover rotas obsoletas.
- Usa UDP na porta configurada (ex.: 5000) para comunicação entre roteadores.
- Cada mensagem contém o tipo (hello ou update), origem (source) e, para updates, a tabela de feromônios (pheromones).

Implementação

- O protocolo está implementado em contêineres Docker (redes drp-net, IPs fixos: 172.20.0.2, 172.20.0.3, 172.20.0.4).
    - Cada roteador (router1, router2, router3) roda o script router.py, configurado via config.json.
    - Detalhes:
        - **Contêineres**: router1 (172.20.0.2), router2 (172.20.0.3), router3 (172.20.0.4).
        - **Rede**: drp-net (sub-rede 172.20.0.0/16).
        - **Porta** UDP: 5000.
- A interface web (app.py e index.html) monitora as tabelas de roteamento, feromônios e eventos em tempo real.

Topologia

A topologia está configurada na rede Docker `drp-net`:

- Router1 (172.20.0.11): Vizinhos 172.20.0.12, 172.20.0.13.
- Router2 (172.20.0.12): Vizinhos 172.20.0.11, 172.20.0.13.
- Router3 (172.20.0.13): Vizinhos 172.20.0.11, 172.20.0.12.

Avaliação e Comparação de Desempenho

Apresentamos os resultados dos testes de inicialização, falhas e taxa de transmissão para os algoritmos ACO (Otimização por Colônia de Formigas) e RIP (Routing Information Protocol), implementados em ambiente simulado com Docker e Quagga respectivamente.

Avaliação

- O tempo de convergência de 862 ms reflete o desempenho do ACO após a inicialização do Docker, sendo mais rápido que o RIP (~2 s).
    - O atraso de 90 segundos é devido à criação dos contêineres e configuração da rede drp-net, não ao protocolo.
- O ACO gera mais mensagens que o RIP devido aos hello, mas converge mais rápido em condições normais.
- Referente aos testes de falhas de router, derrubamos primeiramente o router2, após isso foi identificado pelos outros routers a falha entre 60s a 180s. Realizamos em triplicata e eles mostraram esse comportamento. Porém isso é configurado referente aos intervalos pré determinados pela rede.

- Referente a taxas de transmissão do protocolo enviamos em torno de 5 segundos cada hello de descoberta então a cada minuto temos 12 atualizações, ou melhor dizendo verificações de estado.

Analise de Resultados

Métrica	ACO	QUAGGA
Tempo de Convergência	862 ms	2 segundos
Mensagens (Inicialização)	50 Hello Messages: ~24 enviadas (12 ciclos de 5s × 2 vizinhos) e ~26 recebidas, consistente com hello_interval: 5. 12 Updates: ~6 enviadas (3 ciclos de 30s × 2 vizinhos) e ~6 recebidas, consistente com update_interval: 30. 5 Atualizações de Tabela: Algumas mensagens update não alteraram a tabela.	8 mensagens em 32 segundos (6 enviadas, 2 recebidas), incluindo atualizações acionadas e multicast.
Tempo de Recuperação	60-180 segundos	240-300 segundos
Taxa de Transmissão	12 atualizações por minuto	2 atualizações por minuto

É perceptível que as métricas do ACO variariam dependendo das configurações.