MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PIAUÍ
CAMPUS SENADOR HELVÍDIO NUNES DE BARROS
Curso de Bacharel Sistemas de Informação

REDES DE COMPUTADORES II

Primeira Avaliação: Implementação do Algoritmo de Estado de Enlace com Docker e Python - Valor 10

Objetivo

Desenvolver uma simulação de uma rede de computadores composta por *hosts* e roteadores utilizando Python e Docker, onde os roteadores implementam o algoritmo de roteamento por estado de enlace (*Link State Routing Algorithm*).

Projeto

- **A rede será composta por múltiplas subredes, cada uma contendo:**

    1. 2 hosts
    2. 1 roteador
       (todos na mesma subrede)

- **Os roteadores devem se conectar entre si em uma topologia aleatória (pelo menos parcialmente conectada).**

- **Cada roteador deve implementar o algoritmo de estado de enlace, mantendo:**

    1. Uma base de dados dos enlaces (Link State Database - LSDB)
    2. Uma tabela de roteamento atualizada com base no algoritmo de Dijkstra.

- **O processo em cada roteador deve ter no mínimo duas threads/processos:**

    1. Um para receber pacotes de estado de enlace
    2. Outro para transmitir os pacotes de estado de enlace

- **O aluno deve escolher entre os protocolos TCP ou UDP para a transmissão dos pacotes de controle da rede, justificando a escolha.**

⚙️ **Tecnologias obrigatórias**

- **Python para o desenvolvimento da lógica dos roteadores e hosts**
- **Docker para criar e simular os diferentes elementos da rede (cada *host* e roteador deve rodar em seu próprio *container*)**
- **Comando *route* para manutenção da tabela de roteamento nos roteadores. Atenção, os processos dos roteadores devem manter a tabela de roteamento atualizada de acordo com o que ele aprende no decorrer do tempo.**

Critérios de Avaliação (10 Pontos)

Critério	Descrição	Pontos
1. Estrutura da Rede em Docker	Criação correta de containers para cada host e roteador, com subredes corretamente configuradas.	**1.0**
2. Topologia Aleatória entre Roteadores	Geração e implementação de uma topologia aleatória (mínimo parcialmente conectada).	**1.0**
3. Threads/Processos de Enlace	Implementação de pelo menos duas threads/processos em cada roteador: envio e recebimento de pacotes de estado de enlace.	**1.0**
4. Formato dos Pacotes de Estado de Enlace	Definição e uso consistente de um formato de pacote de estado de enlace com identificador do roteador, custo e vizinhos.	**1.0**
5. Transmissão de Pacotes de Controle	Implementação correta da comunicação entre roteadores utilizando TCP ou UDP, com justificativa adequada da escolha.	**1.0**
6. Armazenamento do Estado de Enlace	Manutenção de uma base de dados de enlaces (LSDB) com informações recebidas dos vizinhos.	**1.0**
7. Algoritmo de Dijkstra	Implementação correta do algoritmo de Dijkstra para calcular a menor rota com base na LSDB.	**1.0**

8. Atualização da Tabela de Roteamento	Atualização dinâmica da tabela de roteamento conforme novos pacotes de estado de enlace são recebidos.	**1.0**
9. Documentação e Justificativas Técnicas	Documentação do projeto, incluindo a justificativa do protocolo usado (TCP/UDP), e instruções para execução.	**1.0**
10. Funcionamento e Demonstração	Projeto funcionando de ponta a ponta com demonstração (ex: ping entre hosts, logs de atualização de rota, etc.).	**1.0**

Entrega Esperada

- Código-fonte completo em Python (No GITHUB)
- Dockerfile(s) e docker-compose.yml
- README.md explicando:
    - Como executar o projeto
    - Justificativa do(s) protocolo(s) escolhido(s)
    - Como a topologia foi construída

- Vídeo demonstrando o funcionamento básico (No Youtube)
- Relatório com explicação do projeto:
    - Diagrama de classe
    - Rede Petri da relação dos hosts e roteadores
- Respostas:
    - Qual os limiares/stress suportado por seu sistema?
        - Mostre através de gráficos de performance
    - Qual a vantagem de sua abordagem?
    - Qual a desvantagem de sua abordagem?
    - Um *host* consegue ‘pingar’ em qualquer outro *host*?
- Data: 06/05/2025
- Trabalho individual
- Apresentação após agendamento.