Blockchain Arena: Simulating Mining Wars and Network Attacks

Assignment 1: P2P Network Simulation

Release Date: May 30, 2025

Due Date: 23:59hrs, 5th June, 2025

Objective

Welcome to the first assignment for the "Blockchain Arena" project! This assignment focuses on setting up your development environment and building a foundational Peer-to-Peer (P2P) network. This network will set the stone for building the cryptocurrency network, transaction propagation, and mining.

Task 1: Setup (Essential)

- **Git & GitHub**: Ensure you are familiar with Git and GitHub. If not, learn the basics as they are essential for version control and used widely everywhere.
  Refer to this tutorial: Git and GitHub Essentials.
  Create a local repository for this project, publish it on GitHub, and either make it public or add me as a collaborator if you want to keep it private (GitHub ID: `soumitra1854`). Share the repository link.
- **Development Environment**: Set up Python or C++ on your system and verify that all required tools and libraries (e.g., `NetworkX` for Python) are installed and functional.

Task 2: P2P Network Construction & Visualization

I hope all of you have watched the 4th video of CS 765: Introduction to Blockchain and cryptocurrency that I have shared earlier. In this assignment, you will implement a basic P2P network structure that will be used in the later stages of the project.

Problem Statement

Develop a program to generate and visualize an undirected P2P network.

Core Requirements:

- **Number of Peers**: Choose a random number between 50 to 100.
- **Peer Degree**: Each peer must connect to 3 to 6 other peers ($3 \leq \deg(v) \leq 6$).
- **Network Connectivity**: The entire graph must be connected (path between any two peers should exist).
- **Validation & Regeneration**:
  - After generation, **verify** all above conditions.
  - If invalid (not connected or degree violation), discard and **recreate the graph from scratch**. Repeat until a valid network is obtained.
  - Think carefully about how to efficiently check connectivity and degree conditions, and the implementation strategy.

Implementation Notes:

- **Language**: Python or C++.
- **Structure**: A `Network` class is recommended for managing peers and connections.
- **Connectivity Check**: Use BFS or DFS.

Visualization:

- Visualize the valid network clearly showing peers and connections.
- **Python users**: Use `NetworkX` with `matplotlib`.
- **C++ users**: Output graph structure (e.g., edge list) to a file; use a Python script with `NetworkX` to visualize from this file.
- Save the image (e.g., `network.png`) and include it in your submission.

Submission

- Push all source code, visualization image, and a `README.md` to your GitHub repo.
- `README.md` should include: your name and rollno, brief run instructions, and any critical dependencies.

Quick Tips

- Represent connections efficiently (e.g., adjacency list).
- Ensure random connections are undirected and degrees are updated correctly.
- Test connectivity algorithms on simple cases.

Questions? Ask during our online sessions! Good luck! 👍