Group Project 3: A Bite of Peer-to-Peer

CECS 327 – Intro to Networks and Distributed Computing

You should submit the required deliverable materials on Canvas by **11:55pm, October 31st (Friday), 2025.**

Overview

This project introduces students to distributed systems by developing a peer-to-peer (P2P) network using Docker containers. The system will consist of 50-100 nodes, each acting as both a client and a server, enabling peer discovery, registration, and communication.

Learning Outcomes

- Gain hands-on experience with Docker for containerized distributed systems.
- Implement a P2P network architecture with a bootstrap node.
- Develop a peer registration and messaging system.

Project Phases & Steps (**Provided codes are just examples and free to change**)

Phase 1: Setting Up the Environment

Objective:

Prepare the system with required tools and dependencies to develop a containerized P2P system.

Steps:

Step 1: Install Required Software

Ensure the following tools are installed:

- Docker: For containerizing and running multiple P2P nodes.
- Docker Compose: To manage multi-container setups.
- Python 3.9+ (or alternative language such as Go/Node.js).

Step 2: Create a Project Directory

```
mkdir p2p-system && cd p2p-system
```

Step 3: Write a Basic P2P Node Application

Each node should:

- Assign itself a unique identifier using `uuid.`
- Start a minimal HTTP server using `Flask` to receive requests.

Step 4: Create a Dockerfile to build a container image.

Step 5: Build and Run a Single Node

```
docker build -t p2p-node .
```

```
docker run -d -p 5000:5000 --name node1 p2p-node
```

Expected Output:

- Running docker `ps` should **show a single container running.**
- Visiting `http://localhost:5000/` in a browser should return:

  ```
  {"message": "Node <UUID> is running!"}
  ```

Phase 2: Developing a Basic P2P Node

Objective:

Modify each node to:

- Store a list of known peers.
- Send and receive messages between nodes.

Steps:

Step 1: Implement Peer Registration (Modify node.py to allow peers to register using a POST request)

Step 2: Enable Peer-to-Peer Messaging (Use a `/message` endpoint to send and receive messages)

Step 3: Start Multiple Nodes

```
docker run -d --name node1 -p 5001:5000 p2p-node
docker run -d --name node2 -p 5002:5000 p2p-node
```

Step 4: Send a Message Between Nodes

```
curl -X POST http://localhost:5002/message -H "Content-Type:
application/json" -d '{"sender": "Node1", "msg": "Hello Node2!"}'
```

Expected Output:

- Nodes register new peers dynamically.
- Logs in Node will display the received message:

```
Received message from Node1: Hello Node2!
```

- The receiving node should print the message in logs and return:

```
{"status": "received"}
```

Phase 3: Bootstrapping the P2P Network and Communication

Objective:

Enable automatic peer discovery using a bootstrap node, then explore P2P communication without bootstrap node.

Step 1: Implement a Bootstrap Node (Create `bootstrap.py` to serve as the central registry for peer nodes)

Step 2: Start the Bootstrap Node in Docker

```
docker build -t bootstrap-node -f bootstrap.Dockerfile .
docker run -d --name bootstrap -p 5000:5000 bootstrap-node
```

Step 3: Start Nodes and Verify Bootstrapping

```
docker run -d --name node1 -p 5001:5000 p2p-node
```

```
docker run -d --name node2 -p 5002:5000 p2p-node
```

Step 4: Check Peer Registration

```
curl http://localhost:5000/peers
```

Expected Output:

- Nodes register with the bootstrap node.
- Running `curl http://localhost:5000/peers` should return:
  `{"peers": ["http://node1:5000", "http://node2:5000"]}`

Step 5: Update Node for Peer Discovery and Management

• After starting, each node will request the peer list from the bootstrap node using a /peers API.
• The node will store this list and periodically update it by communicating with peers directly instead of relying on the bootstrap node.

```
Sample code:
import requests
import threading
import time
from flask import Flask, request, jsonify
import uuid

app = Flask(__name__)
node_id = str(uuid.uuid4())
peers = set()
bootstrap_url = "http://localhost:5000"

# Register with bootstrap node
# Your code here

# Discover peers directly
# Your code here

# Receive and send messages
# Your code here

# Provide peer list when requested
# Your code here
```

Step 6: Update Bootstrap Node for Peer Registration

• The bootstrap node only provides peer registration and the initial peer list.

• Once nodes are connected, it is no longer necessary for further communication.

Step 7: Test Peer Communication Without Bootstrap (only show small examples)
1. Start Bootstrap Node

```
docker build -t bootstrap-node -f bootstrap.Dockerfile .
docker run -d --name bootstrap -p 5000:5000 bootstrap-node
```

2. Start Nodes

```
docker run -d --name node1 -p 5001:5000 p2p-node
docker run -d --name node2 -p 5002:5000 p2p-node
```

3. Send Messages Between Nodes
Once nodes discover each other, use curl to send messages directly without using the bootstrap:

```
curl -X POST http://localhost:5001/message -H "Content-Type:
application/json" \
-d '{"sender": "Node2", "msg": "Hello Node1!"}'
```

```
curl -X POST http://localhost:5002/message -H "Content-Type:
application/json" \
-d '{"sender": "Node1", "msg": "Hey Node2, how are you?"}'
```

#Your test case needs to show the communicated among dozens of nodes within the P2P network instead of two nodes.

---

3. Required Deliverables

1. README File: Instructions on how to build, run, and test your system.
2. Source Code: Include a Makefile (if applicable) and ensure the submission is in the correct format (`node.py`, `bootstrap.py`, and `Dockerfile`, etc.).
3. Project Report (PDF):
   - Explanation of the system design.
   - Screenshots of your design and outputs.
4. Execution Demonstration Video:
   - Record a video showing your code execution and outputs.
   - The video should display your name and date as identification.
   - Upload to **YouTube** (or another platform) and provide a link in your report.

4. Submission Guidelines

- Submit a single .zip/.rar file containing all required files (zip named by your Name).
- Only one submission per group is required.
- Ensure your code compiles and runs; otherwise, a zero grade will be assigned.
- If your code is incomplete, specify missing parts in your report for partial credit consideration.
- Provide sufficient comments in your code to explain the logic.

5. Grading Criteria

Details	Points
Have a README file shows how to compile and test your submission	5 pts
Submitted code has proper comments to show the design	15 pts
Screen a ***video*** to record code execution and outputs	15 pts
Have a **report** (pdf or word) file explains the details of your entire design	20 pts
Report contains clearly individual contributions of your groupmates	5 pts
Code can be compiled and shows correct outputs	40 pts

6. Policies

1. Late Submissions: Will be penalized as per course syllabus.
2. Plagiarism: Code-level discussions are prohibited. Anti-plagiarism tools will be used.

3. Use of AI Tools: ChatGPT, GPT-4, and similar AI tools are prohibited for both code and written content.

Final Notes:

- This project requires independent research and problem-solving skills.
- Properly cite any resources you reference.
- Have fun experimenting with distributed systems and networking!

Good luck! 🚀