Group Project 1: A Bite of Distributed Communication

CECS 327 – Intro to Networks and Distributed Computing

You should submit the required deliverable materials on Canvas by **11:55pm, October 09th (Thursday), 2025.**

---

1. Project Overview

This project builds on your Docker foundation by implementing Anycast and Multicast communication models using UDP and TCP. You will:

1. Create Dockerized server and client applications.
2. Demonstrate anycast (multiple servers, one chosen by the network).
3. Demonstrate multicast (one sender, multiple receivers).
4. Monitor traffic with tools like tcpdump or Wireshark inside containers or on the host.

2. Project Tasks

Task 1: Anycast with Docker (TCP)

Goal: Create a system where multiple servers listen on the same port, and a client connects to one server at random (simulating Anycast).

Steps:

1. **Write Server Code (`server.py`)**:
   - Create a TCP server that listens on port 5000.
   - Each server responds with a unique message (e.g., “Hello from server1”).
2. **Create a Dockerfile**:
   - Use a Python base image.
   - Copy `server.py` and install dependencies.
3. **Set Up Docker**:
   - Use `docker-compose.yml` to launch 3 server containers and 1 client container.
   - All servers listen on port 5000, and Docker’s network balances client requests.
4. **Write Client Code (`client.py`)**:
   - Connect to the Docker network hostname (resolves to one server).
   - Display the server’s response.
5. **Run and Test**:
   - Run the client multiple times to connect to different servers.
   - Use `tcpdump` to capture traffic:

   -
     ```
     docker exec -it <container_id> apt-get update && apt-get install -y tcpdump
     ```

   -
     ```
     docker exec -it <container_id> tcpdump -i eth0 udp port 5000
     ```

Deliverables:

- `server.py`: Simple TCP server code.

- `Dockerfile`: For server and client containers.
- `docker-compose.yml`: Launches 3 servers and 1 client.
- `client.py`: Client code to connect and display responses.
- Sample outputs and `tcpdump` logs.

Example Outputs:

- **Server Logs** (e.g., server1):

  ```
  Server ready on port 5000
  Accepted connection from (172.20.0.5, 48234)
  Sent: Hello from server1
  ```

- **Client Output** (3 runs):

  ```
  Received: Hello from server1
  Received: Hello from server2
  Received: Hello from server3
  ```

- **tcpdump Output**:

  ```
  IP 172.20.0.5.48234 > 172.20.0.2.5000: Flags [S], seq 12345
  ```

Task 2: Multicast with UDP

Goal: Build a system where one or more senders broadcast messages to multiple receivers in a multicast group.

Steps:

1. **Write Receiver Code (`multicast_receiver.py`)**:
   - Join a multicast group (e.g., 224.1.1.1, port 5007).
   - Add a `--duration` argument to leave the group after X seconds.
   - Display received messages.
2. **Write Sender Code (`multicast_sender.py`)**:
   - Send messages to the multicast group, including:
     - JSON (e.g., `{"sensor":"temp", "value":23.5}`).
     - Binary data (e.g., random bytes).
   - Handle both message types in the receiver.
3. **Add a Second Sender**:
   - Launch another sender container sending different data (e.g., `{"sensor":"humidity"}`).
   - Observe multiple sources in `tcpdump`.
4. **Monitor Traffic**:
   - Run `tcpdump -i eth0 udp port 5007` in a receiver container.
   - Compare captured packets to program output to spot UDP issues (e.g., dropped packets).

Deliverables:

- `multicast_sender.py`: Sends JSON and binary data.
- `multicast_receiver.py`: Supports `--duration` and handles message types.
- `docker-compose.yml`: Launches 1–2 senders and multiple receivers.
- Sample outputs and `tcpdump` logs.

Example Outputs:

- **Sender**:

  ```
  Sent: Multicast message
  Sent: {"sensor":"temp", "value":23.5}
  ```

- **Receiver1** (15 seconds):

  ```
  Joined multicast group
  Received: Multicast message from (172.20.0.4, 5007)
  Received: {"sensor":"temp", "value":23.5}
  Leaving multicast group
  ```

- **tcpdump Output**:

  ```
  IP 172.20.0.4.56123 > 224.1.1.1.5007: UDP, length 32
  ```

---

3. Required Deliverables

1. **README File:** Instructions on how to build, run, and test your Dockerized system.
2. **Source Code:** Include a Makefile (if applicable) and ensure the submission is in the correct format.
3. **Project Report (PDF or Word):**
   - Why UDP multicast is not reliable.
   - How group join/leave affects who receives packets.
   - Differences observed between single-source and multi-source multicast.
   - Clearly stated contributions of each team member.
4. **Execution Demonstration Video:**
   - Receivers joining and leaving groups.
   - Different message types being received.
   - tcpdump capture proving multiple sources.
   - 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.
- 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	20 pts
Have a **report** (pdf or word) file explains the details of your entire design	20 pts
Report contains clearly individual contributions of your group mates	5 pts
Code can be compiled and shows correct outputs	35 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! 🚀