# Unicast Protocol

## System Overview

The system is a simple distributed network that utilizes the Unicast communication protocol to enable one-to-one message exchanges between nodes. It consists of a master server and multiple client nodes (node1, node2, node3, and node4), all running within Docker containers. The server manages client registrations and routes messages between nodes.

## Components

1. **Server**
    - Functionali**ty:** The server's primary responsibilities include listening for incoming connections, registering nodes, routing messages between nodes, and logging communication details.
    - **Communication:** The server uses a TCP socket to accept connections from nodes. It maintains a dictionary of connected clients, mapping node IDs to their corresponding socket and address information.
    - **Logging:** The server logs each message's type, time, source and destination IP addresses and ports, protocol, length, and flags to a CSV file.
2. **Nodes (node1, node2, node3, node4)**
    - **Functionality:** Each node can connect to the server, send registration messages, and exchange messages with other nodes.

- **Communication:** Nodes use TCP sockets to connect to the server and send messages. They can specify the recipient node ID and the message content.

**Communication Flow**

1. **Node Registration:** Upon starting, each node connects to the server and sends a registration message in the format register:<node_id>.
2. **Message Sending:** A node can send a message to another node by specifying the recipient's node ID and the message content. The message format is <sender_id>:<recipient_id>:<message>.
3. **Message Routing:** When the server receives a message from a node, it parses the message to determine the sender and recipient IDs and the content. It then routes the message to the appropriate recipient node based on the recipient ID.
4. **Message Reception:** The recipient node receives the message and can process it accordingly.

**Docker Configuration**

- **Network:** All containers are connected to a Docker network named 'proj1-distributed-network', enabling them to communicate using their container names as hostnames (e.g., 'server', 'node1').
- **Port Mapping:** The server container exposes port 12345, which is used for communication with the nodes.

**Logging**

- The server logs all communication events to a CSV file with the following columns: Type, Time(s), Source_Ip, Destination_Ip,

Source_Port, Destination_Port, Protocol, Length (bytes), and Flags (hex).

- The log file is located at /app/UnicastProtocol/logs/communication_log.csv within the server container.

**Code Structure**

- Each node (node1.py, node2.py, node3.py, node4.py) and the server (server.py) have separate Python scripts.
- The scripts use the socket library for network communication and threading for handling multiple clients concurrently (in the case of the server).

# Broadcast Protocol

## System Overview

The system is a simple distributed network that utilizes broadcast communication protocol to enable a master server to send messages to multiple client nodes simultaneously. It consists of a master server and multiple client nodes (node1, node2, node3, and node4), all running within Docker containers. The server broadcasts messages, and all nodes listen for and receive these messages.

## Components

1. **Master Server**
    - **Functionality:** The master server's primary responsibility is to broadcast messages to all connected nodes.
    - **Communication:** The server uses a UDP socket with the broadcast option enabled. It sends messages to the special broadcast address, which ensures that all nodes on the network receive the message.
2. **Nodes (node1, node2, node3, node4)**
    - Functionality: Each node listens for broadcast messages from the master server and processes them upon receipt.
    - Communication: Nodes use UDP sockets with the broadcast option enabled. They bind to the broadcast port and listen for incoming messages.

## Communication Flow

1. **Server Broadcast:** The master server sends a broadcast message using the UDP socket. The message is sent to the broadcast address, ensuring that all nodes on the network receive it.

2. **Message Reception:** Each node listens for messages on the broadcast port. When a broadcast message is received, the node processes it accordingly.

**Docker Configuration**

- **Network:** All containers are connected to a Docker network, enabling them to communicate using their container names as hostnames.
- **Port Mapping:** The broadcast port (12345) is used for communication between the server and nodes. This port is bound to the server container and is listened on by the node containers.

**Code Structure**

- The master server (server.py) and each node (node1.py, node2.py, node3.py, node4.py) have separate Python scripts.
- The scripts use the socket library for network communication.
- The server script includes a function to broadcast messages to all nodes.
- Each node script includes a function to listen for broadcast messages and print them upon receipt.

**Example Usage**

- The master server can broadcast a message using the broadcast_message.py script, which sends a predefined message to all nodes.
- Each node will receive and print the broadcast message.