Section	01
Name	Yash Vardhan Sharma
CWID	A20575676
Adv Operating Systems	PA2

INDEX

SR No.	Content	Page No.
1	Overview	2
2	Requirements	2
3	Tools Used	2
4	Structure of the Code	3
5	Structure of the Program	3
6	The Process Description	3-4
7	cURL of the Given APIs	4
8	Testing Result - 1 Peer and 1 Indexing Server	5
9	Testing Result - Deploying at Least 3 Peers and 1 Indexing Server	5
10	Testing Result - Measuring Average Response Time	6
11	Graph Result - Measuring Average Response Time	6
12	Testing Result - Benchmark the Latency and Throughput of Each API	7
13	Graph Result - Benchmark the Latency and Throughput of Each API	8-10
14	Conclusion	11

Overview:

This assignment implements a **Peer-to-Peer (P2P) Publisher-Subscriber System** in Python. The system allows multiple peer nodes to communicate through a central indexing server, facilitating message publishing, topic management, and subscription functionalities.

Requirement:

Functional Requirements:

1. **Peer Nodes**:
    - Each peer node should register with the indexing server.
    - Peer nodes must create topics.
    - Each peer should publish messages to specific topics.
    - Peers must subscribe to topics and retrieve messages.
    - Each peer should delete topics.
2. **Indexing Server**:
    - The server must maintain a list of active peer nodes and their topics.
    - It should provide operations for peers to register and unregister.
    - It should allow peers to add or delete topics.
    - The server must handle messages from peer nodes and facilitate message retrieval.
3. **Concurrency**:
    - The system should handle multiple concurrent requests from peer nodes.
    - Peers must publish and subscribe to topics while interacting with others simultaneously.

Non-Functional Requirements:

1. **Performance**:
    - The system should demonstrate acceptable response times under load.
    - The indexing server should manage a significant number of topics and messages efficiently.
2. **Logging**:
    - The application should maintain logs for both the indexing server and peer nodes.
3. **Ease of Use**:
    - Provide scripts for easy deployment and setup of the system.

Tools Used:

Python 3.6+: The programming language used for development.
Flask: Web framework for creating the indexing server and peer nodes.
Requests: Library for making HTTP requests between peers and the server.
Make: Tool for automating tasks related to setup and management.
Matplotlib: Library for plotting graphs and visualizing results.
TQDM: Library for displaying progress bars in the terminal.

Structure of the code:

```
PA2
├── Code
│   ├── deploy.sh                # Deployment script to start the indexing server and peers
│   ├── indexing_server.py       # Central indexing server script
│   ├── peer_node.py             # Peer node script
│   ├── api_benchmark.py         # Script for benchmarking API performance
│   ├── response_time_test.py    # Script for measuring response times under load
│   ├── test_p2p.py              # Script for testing API functionalities
│   ├── requirements.txt         # List of required dependencies
│   └── Makefile                 # Makefile for setting up environment and installing dependencies
├── Files
│   ├── Report                   # Report
│   ├── Design Doc               # Design documentation
│   └── ReadMe                   # README file
└── Out
    ├── Logs                     # Directory for storing log files
    └── Images                   # Directory for storing generated images and graphs
```

Structure of the program:

Components:

1. **Indexing Server**:
    - Handles peer registrations, topic creation, and message handling.
    - Provides API endpoints for peers to interact with the system.
2. **Peer Nodes**:
    - Register with the indexing server, create topics, publish messages, and pull messages.
3. **Benchmarking and Testing Scripts**:
    - **api_benchmark.py**: Measures latency and throughput of API calls.
    - **response_time_test.py**: Tests the average response time when multiple peer nodes query topics.
    - **test_p2p.py**: Tests the overall functionality of the P2P system by performing various API requests.

The Process Description:

The functionality concludes by deploying the APIs in the below order:

1. /register_peer_node: Registers a new peer node with the indexing server, allowing it to participate in the system.

2. /unregister_peer_node: Unregisters a peer node from the indexing server, removing it from active participants.
3. /create_topic: Creates a new topic on the indexing server, allowing peers to publish and subscribe to it.

4. /delete_topic: Deletes an existing topic from the indexing server, removing it and notifying all subscribed peers.

5. /publish: Publishes a message to a specified topic, making it available to all subscribed peers.

6. /subscribe: Subscribes a peer to a specified topic, allowing it to receive messages published to that topic.

7. /pull_messages: Retrieves messages from a specified topic for a subscribed peer, providing it with any published messages.

8. /query_peers: Returns a list of peers that are subscribed to a specified topic, enabling peer discovery based on topic interest.

cURL of the Given APIs

Register Peer Nodes

curl -X POST http://localhost:5000/register_peer_node -H "Content-Type: application/json" -d '{"peer_id": "peer1"}'

Create a Topic

curl -X POST http://localhost:5000/create_topic -H "Content-Type: application/json" -d '{"topic": "news"}'

Publish a Message

curl -X POST http://localhost:5000/publish -H "Content-Type: application/json" -d '{"peer_id": "peer1", "topic": "news", "message": "Breaking news: AI is transforming the world!"}'

Subscribe to a Topic

curl -X POST http://localhost:5000/subscribe -H "Content-Type: application/json" -d '{"peer_id": "peer1", "topic": "news"}'

Pull Messages

curl -X POST http://localhost:5000/pull_messages -H "Content-Type: application/json" -d '{"peer_id": "peer1", "topic": "news"}'

Query Peers

curl -X GET "http://localhost:5000/query_peers?topic=news"

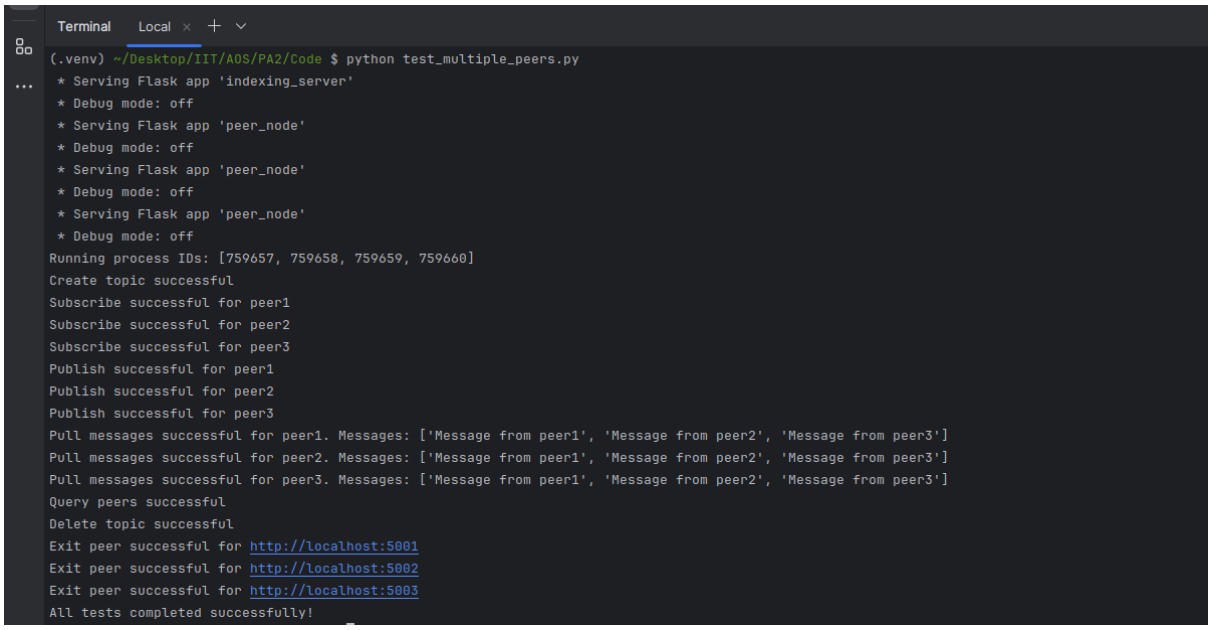
Delete a Topic

curl -X DELETE http://localhost:5000/delete_topic -H "Content-Type: application/json" -d '{"topic": "news"}'

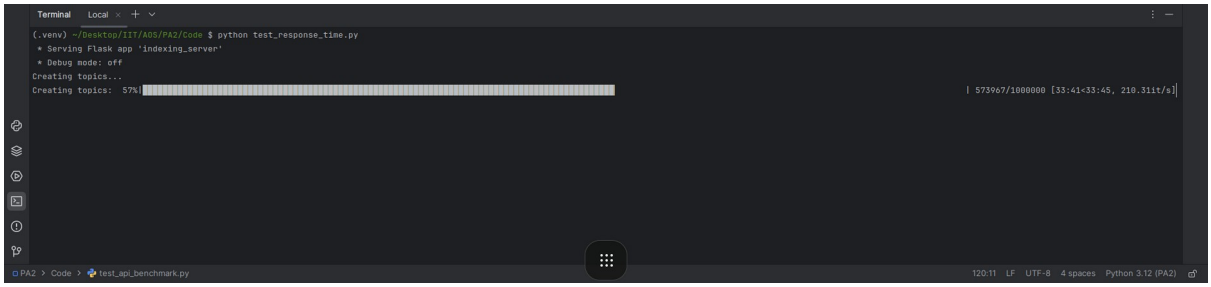
Testing Results:

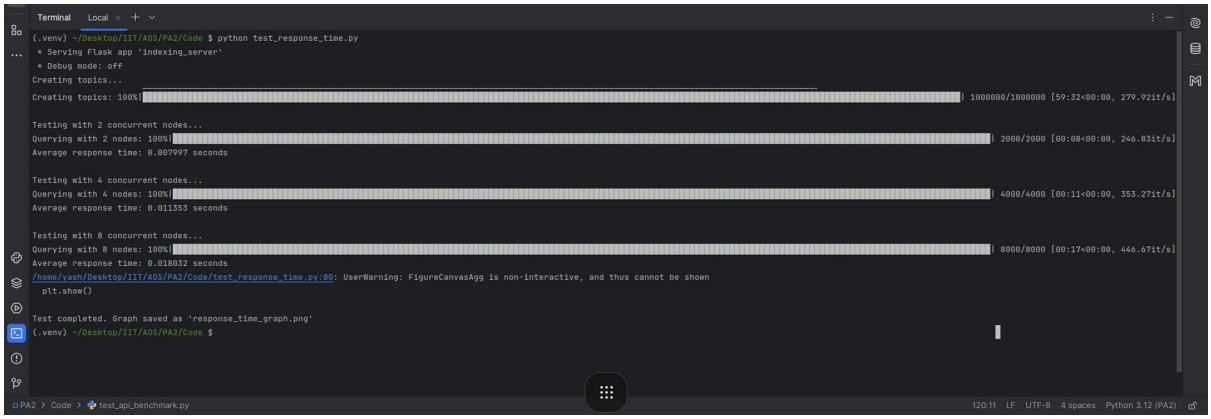
1 Peer and 1 Indexing Server

Deploying at Least 3 Peers and 1 Indexing Server

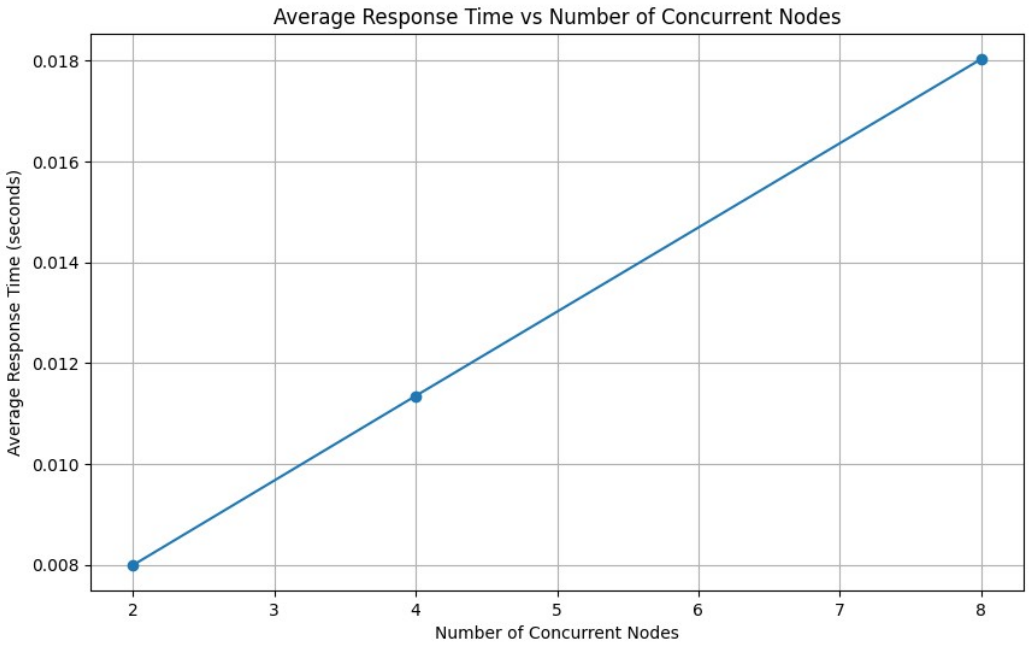


Measuring Average Response Time

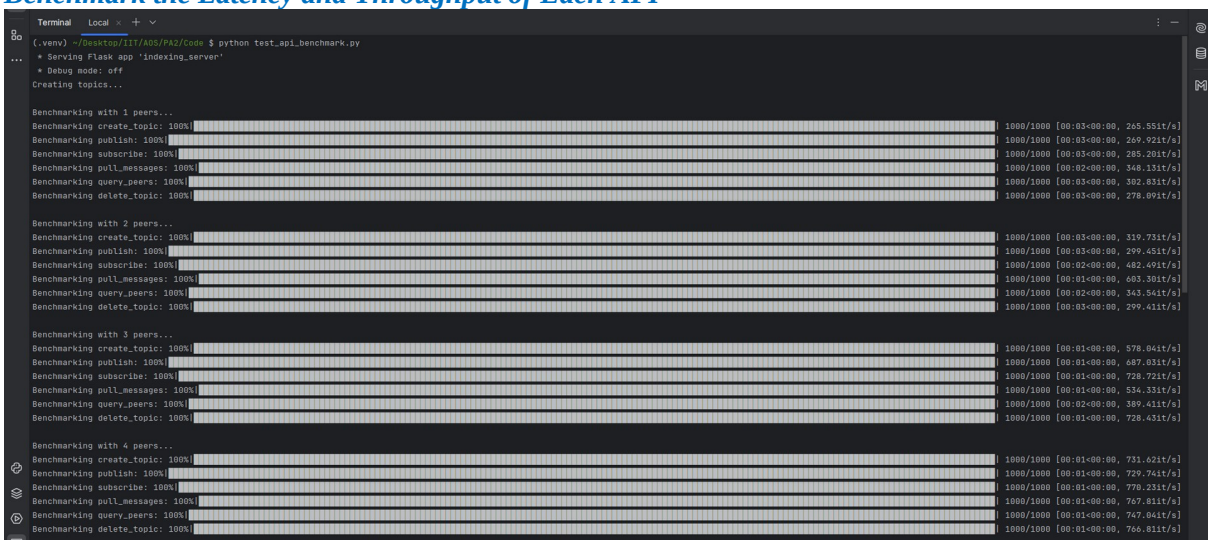


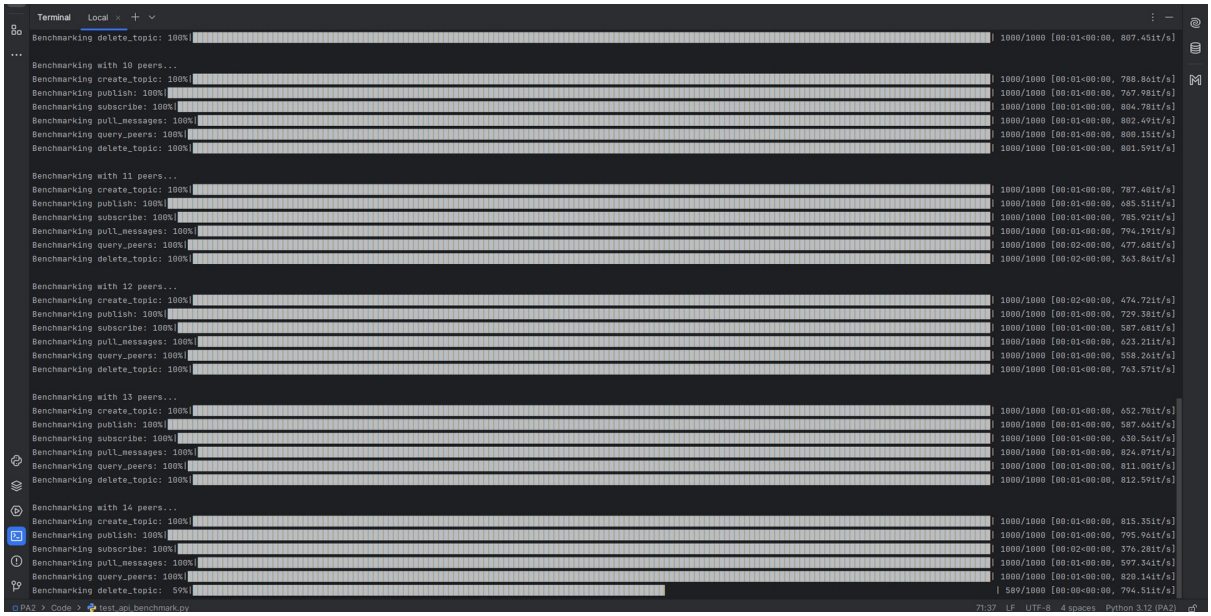


Graph Results

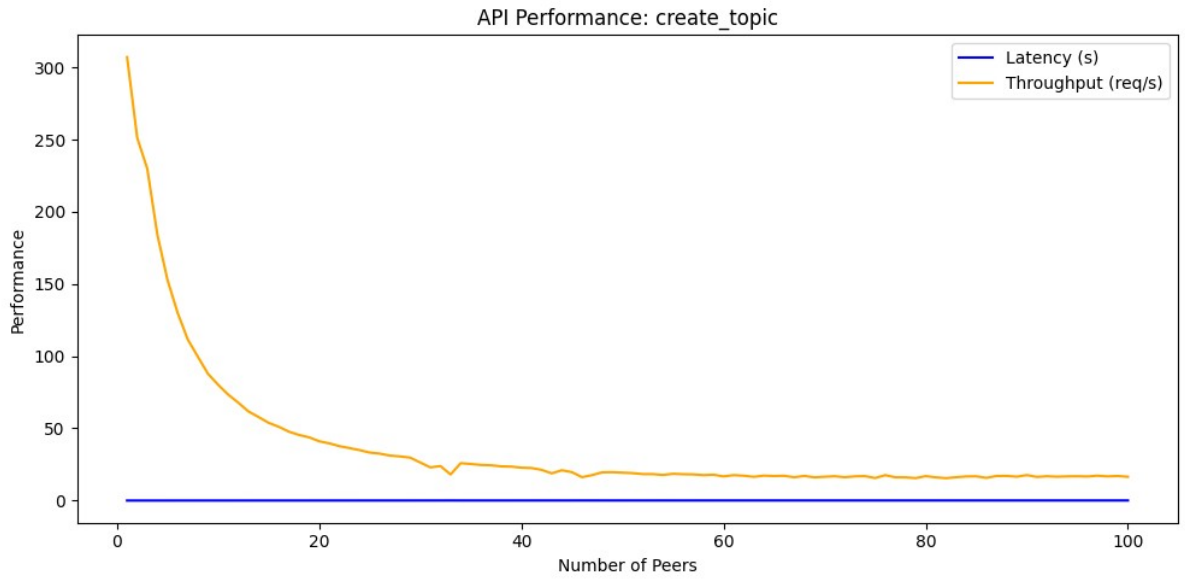


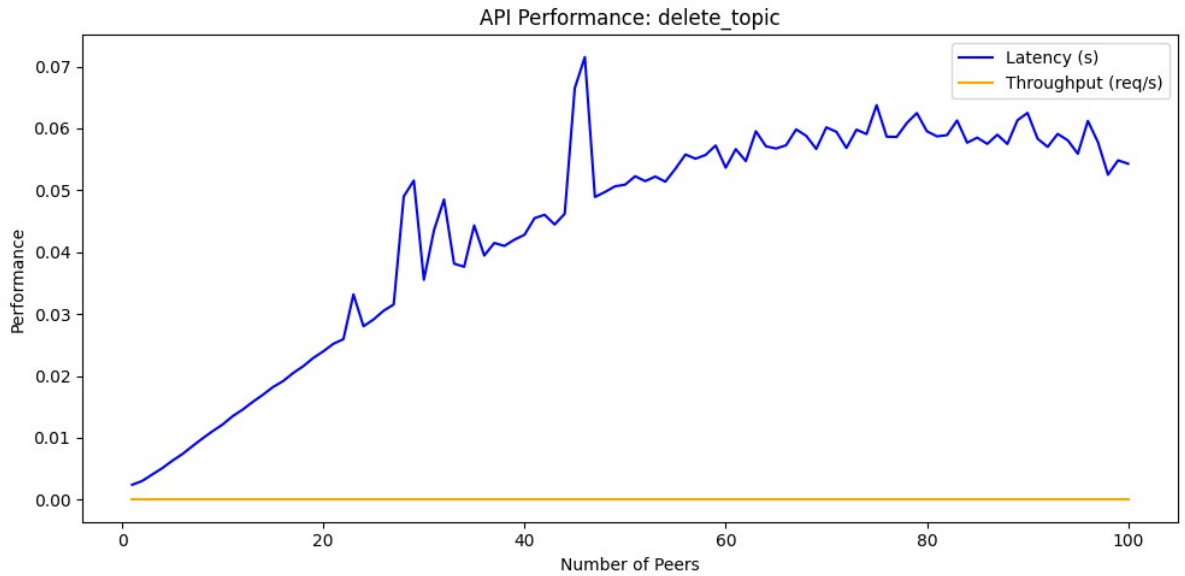
Benchmark the Latency and Throughput of Each API

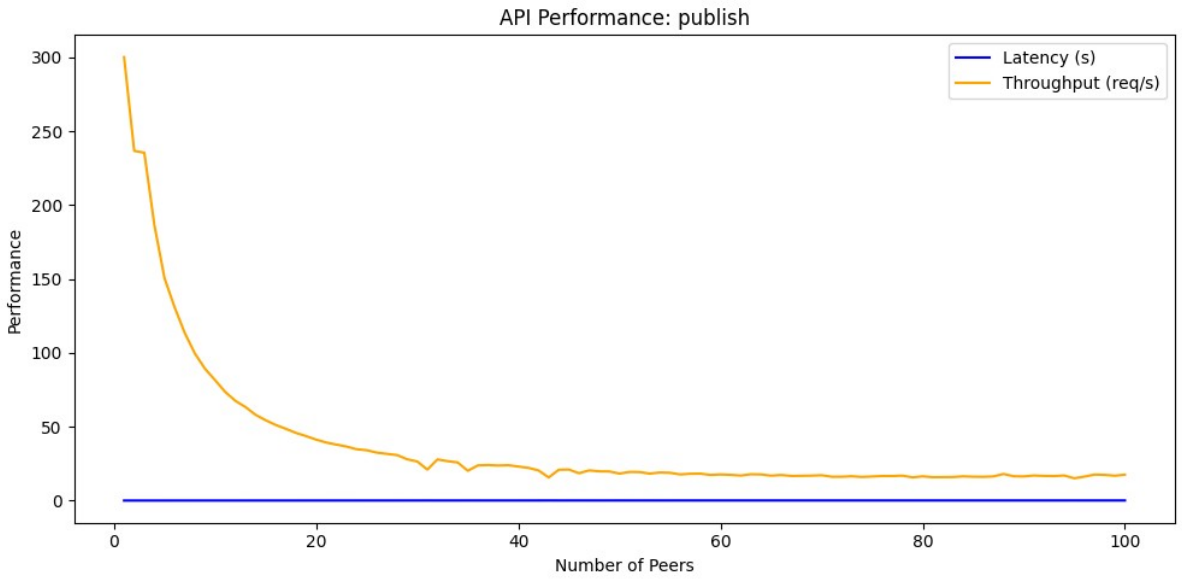


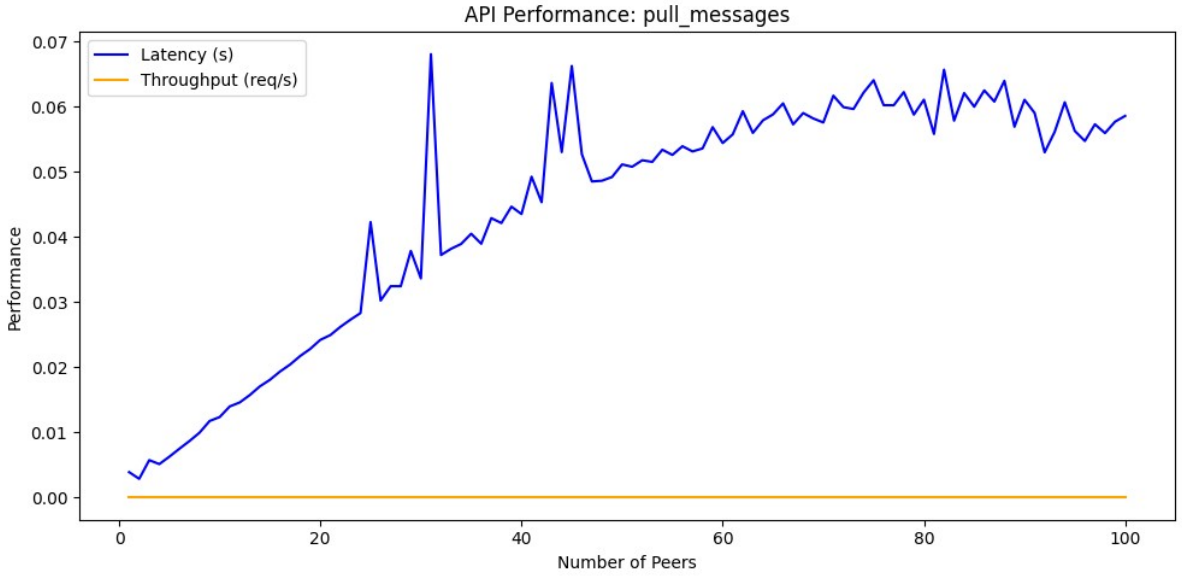


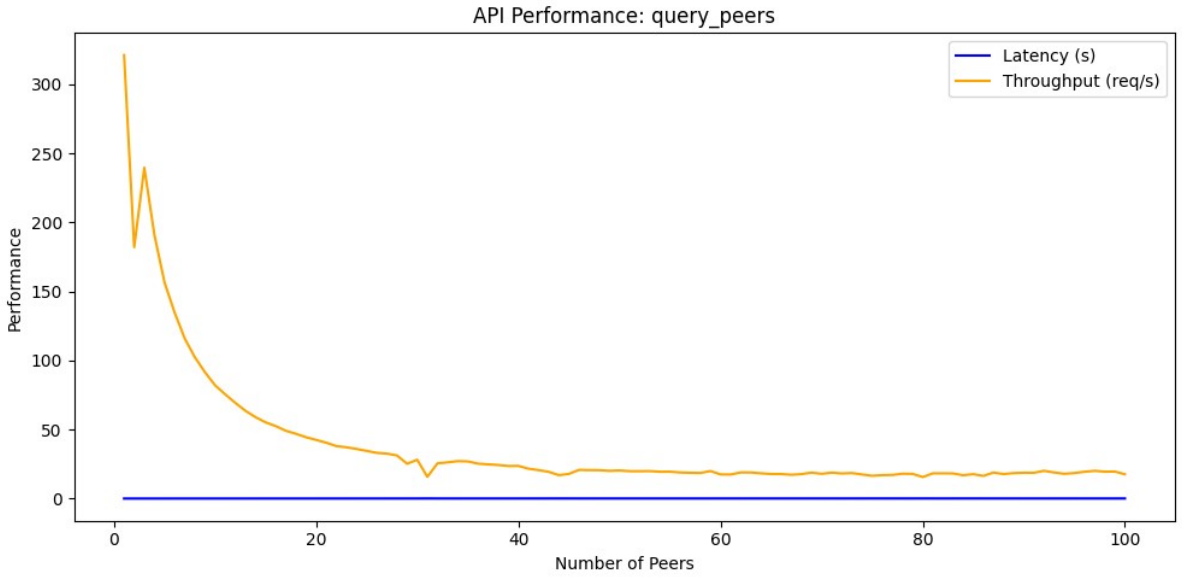
Graph Results

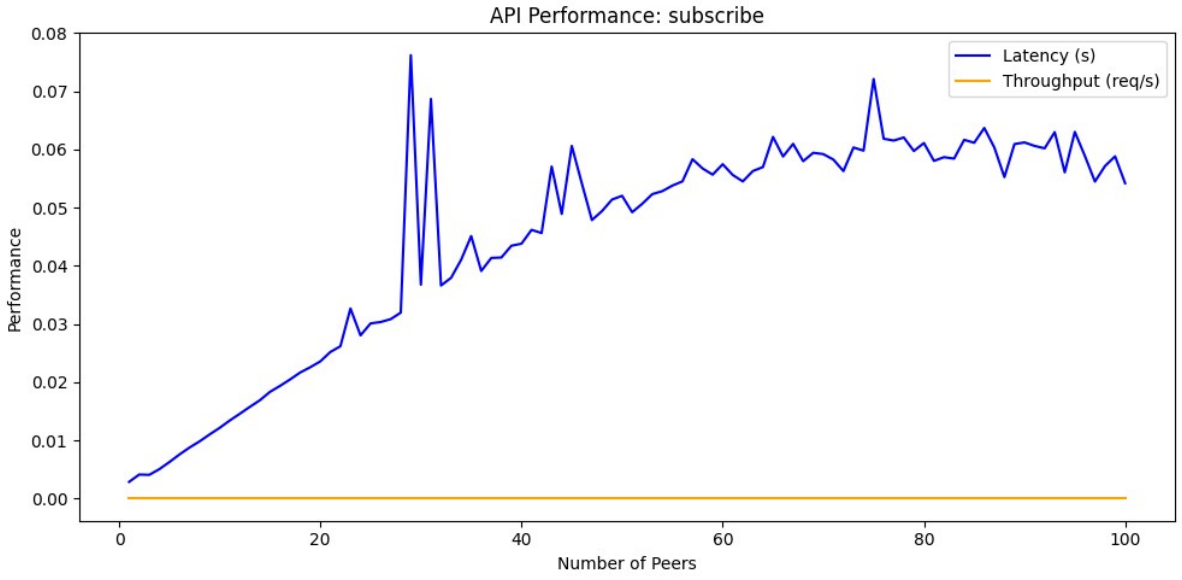





API Performance: publish
Latency (s)
Throughput (req/s)
Performance
Number of Peers
0
50
100
150
200
250
300
0
20
40
60
80
100


API Performance: pull_messages
Latency (s)
Throughput (req/s)
Performance
Number of Peers
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0
20
40
60
80
100


API Performance: query_peers
Latency (s)
Throughput (req/s)
Performance
Number of Peers
0
50
100
150
200
250
300
0
20
40
60
80
100


API Performance: subscribe
Latency (s)
Throughput (req/s)
Performance
Number of Peers
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0
20
40
60
80
100

Note:

During benchmark testing, the server remained stable and was not overwhelmed even with 100 peers operating simultaneously. However, the test was time-consuming, so the benchmark testing was concluded with a maximum of 100 peers.

CONCLUSION:

The **P2P Publisher-Subscriber System** was successfully implemented, allowing multiple peer nodes to communicate through a central indexing server. Testing confirmed that the system handles registrations, topic management, and message publishing efficiently. The performance testing showed scalability and responsiveness, making it suitable for larger deployments.

Future enhancements could include improving error handling, optimizing performance under high load, and expanding the API functionalities.