Final Project: System Report

1. Overview

This report presents the findings of experiments conducted on a system that based on the previous assignment (PA3). This system achieves mainly two functions: (1) Replicated Topics (Performance optimization and Fault tolerance); (2) Dynamic Topology Configuration. Specifically, this system implements the core functions of each node in the decentralized P2P system, including topic management, message routing and forwarding, network communication, and event logging. It is also responsible for coordination and distributed operations between nodes. The goal of the experiments was to evaluate the performance and functionality of the system, specifically focusing on the following aspects:

• The correct functionality of APIs (create, delete, publish, subscribe).

• The latency and throughput of requests across peers.

• The implement of Replicating topics for performance optimization and fault tolerance

• The implement of Dynamic topology configuration (add & removal of Hypercube nodes)

This report also provides recommendations for possible improvements and extensions to enhance the system.

1. System Overview

**Key Features:**

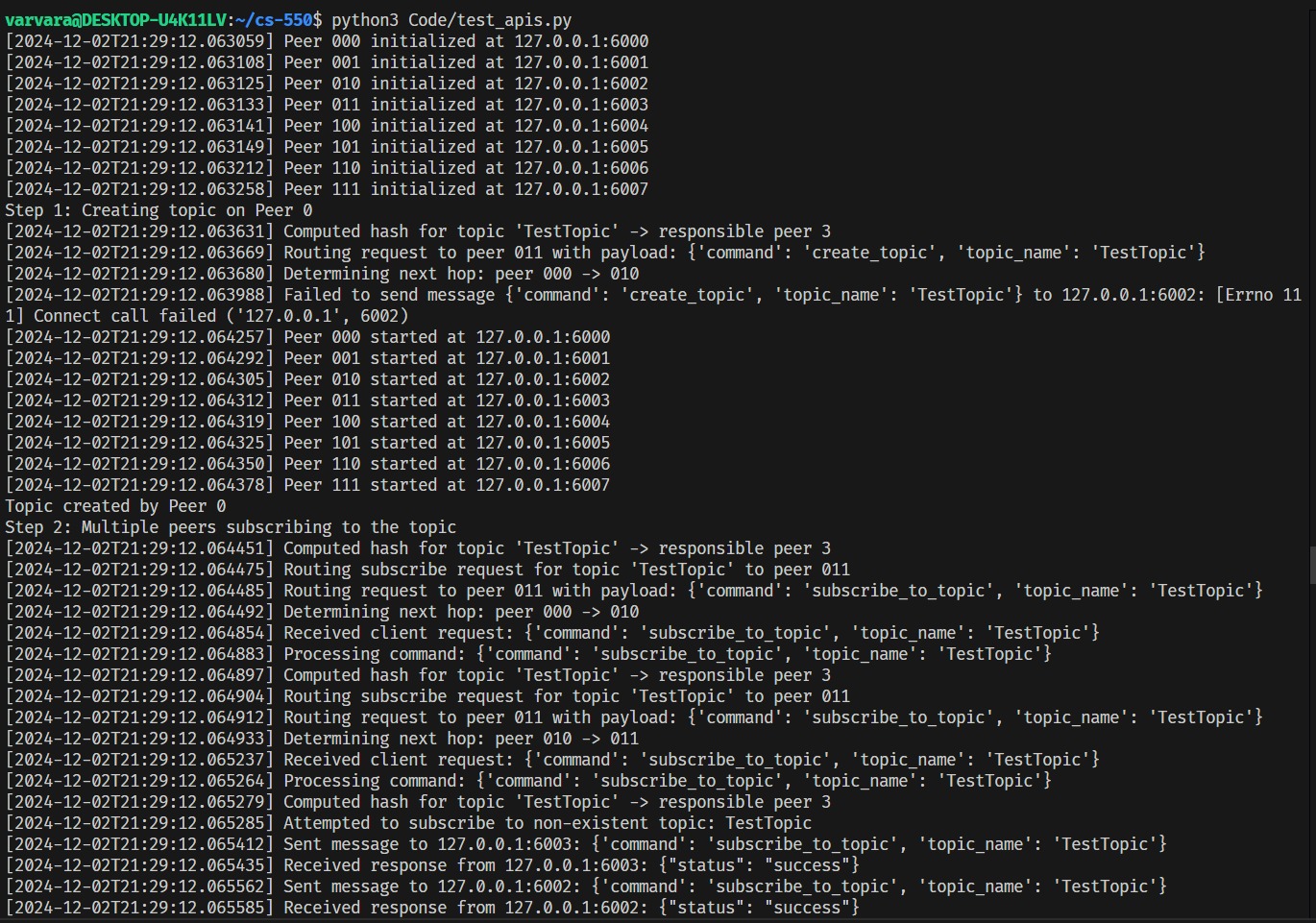
* **Node Representation and Network Structure**: Each node is identified by a binary peer\_id; Neighbors are computed dynamically using the compute\_neighbors method, ensuring that nodes differ by only one bit in their IDs, adhering to hypercube topology; Dynamic Topology Support: The system allows nodes to be added or removed dynamically at runtime, with the status updated via detect\_node\_status and ping\_node.
* **Dynamic Routing**: Routing is implemented with the find\_next\_hop method, which determines the next hop node to move closer to the target node. The route\_request method forwards requests based on calculated target nodes and routing information.
* **Topic Creation**: The hash\_function computes the node responsible for storing a topic by hashing the topic name; The create\_topic method creates topics either locally or forwards the request to the responsible node.
* **Topic Replication**: When creating a topic, the create\_replicas method generates replicas on neighboring nodes for fault tolerance. The replicate\_topic method handles storing replica data.
* **Message Publishing and Subscription**: publish\_message publishes messages to a topic. If the node is not responsible, the request is routed to the appropriate node. subscribe\_to\_topic allows a node to subscribe to a topic, with requests routed if necessary. Messages are propagated to all subscribers using propagate\_message.
* **Topic Deletion**: Topics can be deleted via the delete\_topic method, which also updates replicas to reflect the change.
* **Node Failure Detection (belongs to Node Failure Detection)**: Neighbors' statuses are periodically checked using ping\_node and updated in self.active\_nodes. Offline nodes are excluded from active routes, and rejoining nodes are handled by handle\_node\_rejoin.
* **Replica-based Fault Recovery**: If a node responsible for a topic goes offline, its replicas ensure continued access to the topic. When a node rejoins, the replicate\_topics method restores its topics from replicas.
* **Dynamic Topology Support (Adding New Nodes)**: New nodes are registered in the network via announce\_new\_node, triggering topic reallocation as needed. Neighbor lists are updated dynamically using the compute\_neighbors method.
* **Offline and Rejoining Nodes**: Offline nodes are marked as inactive and removed from neighbor lists. Rejoining nodes synchronize their topics using handle\_node\_rejoin.

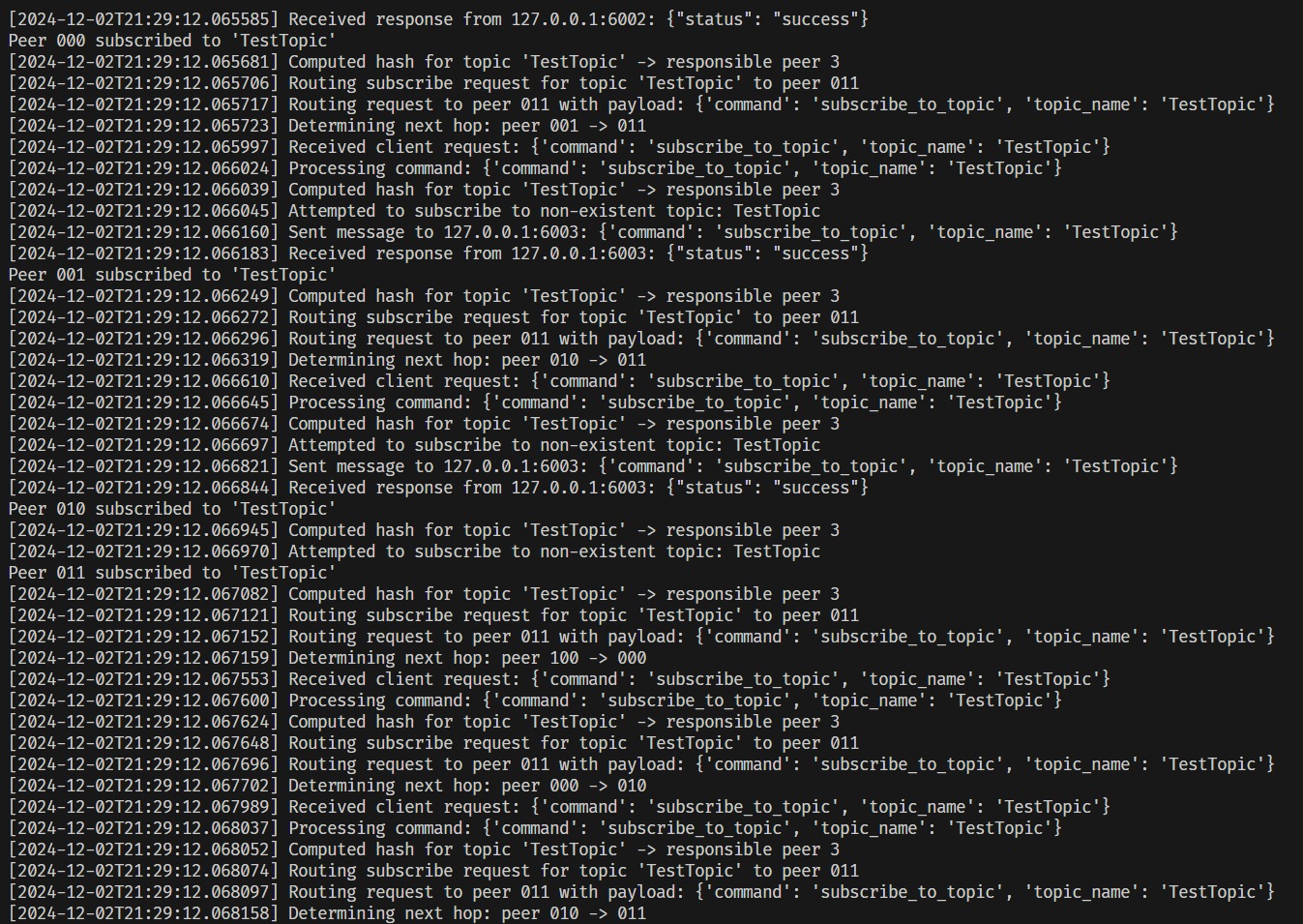
1. Experiments and Results

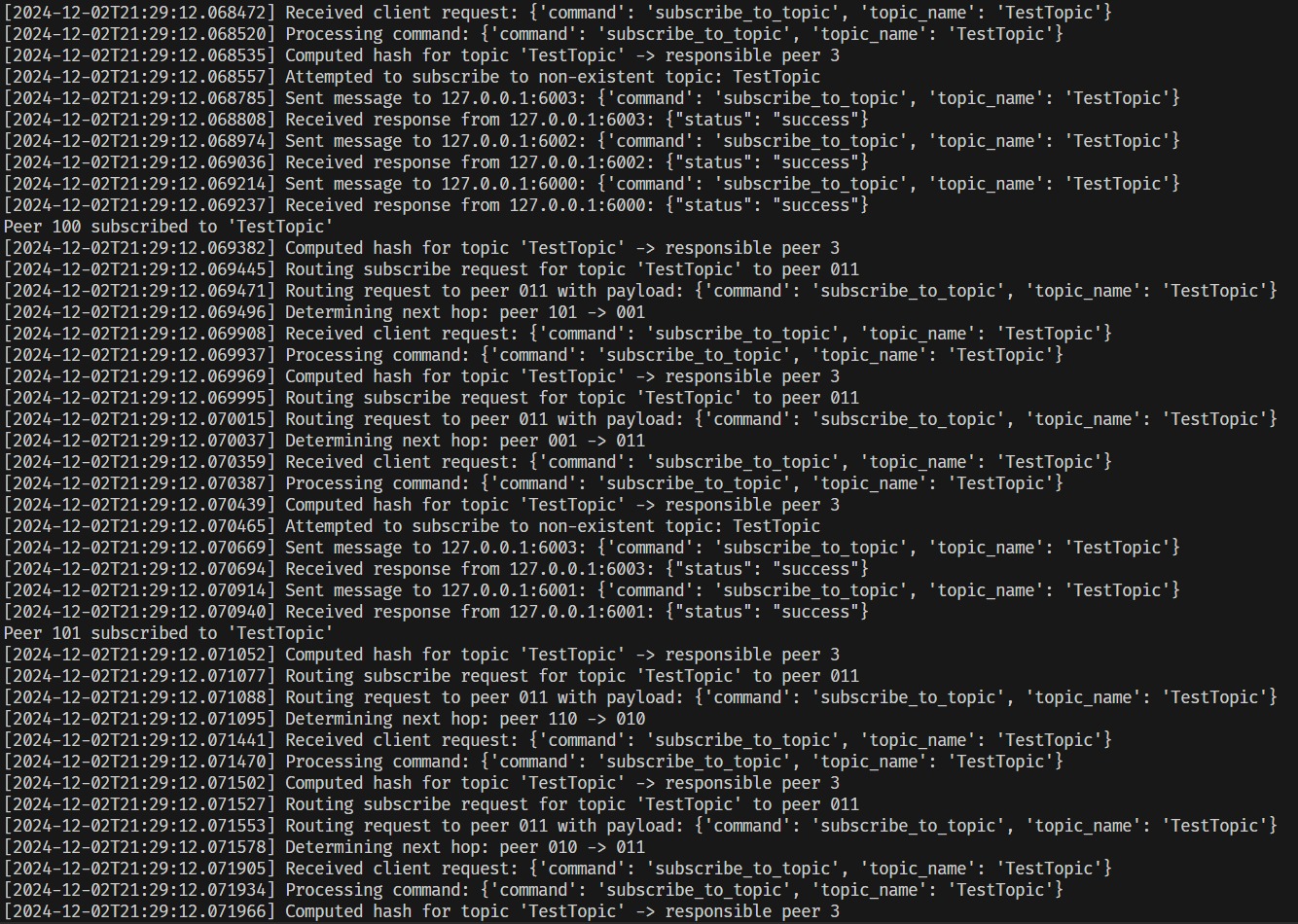
**3.1 Experiment 1: Deploying 8 peers. They can be set up on the same machine or different machines.**

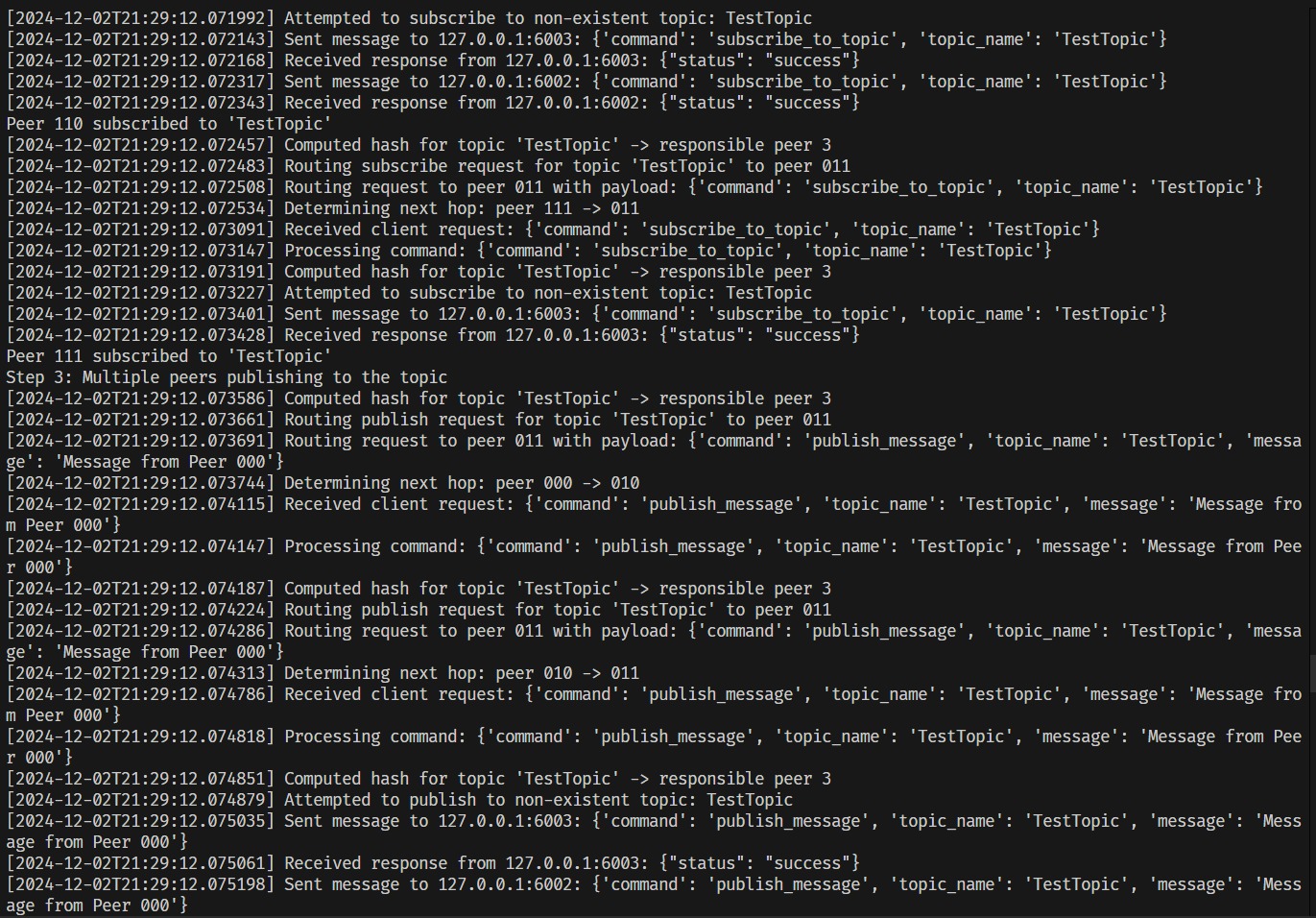
**a. Ensure all APIs are working properly.**

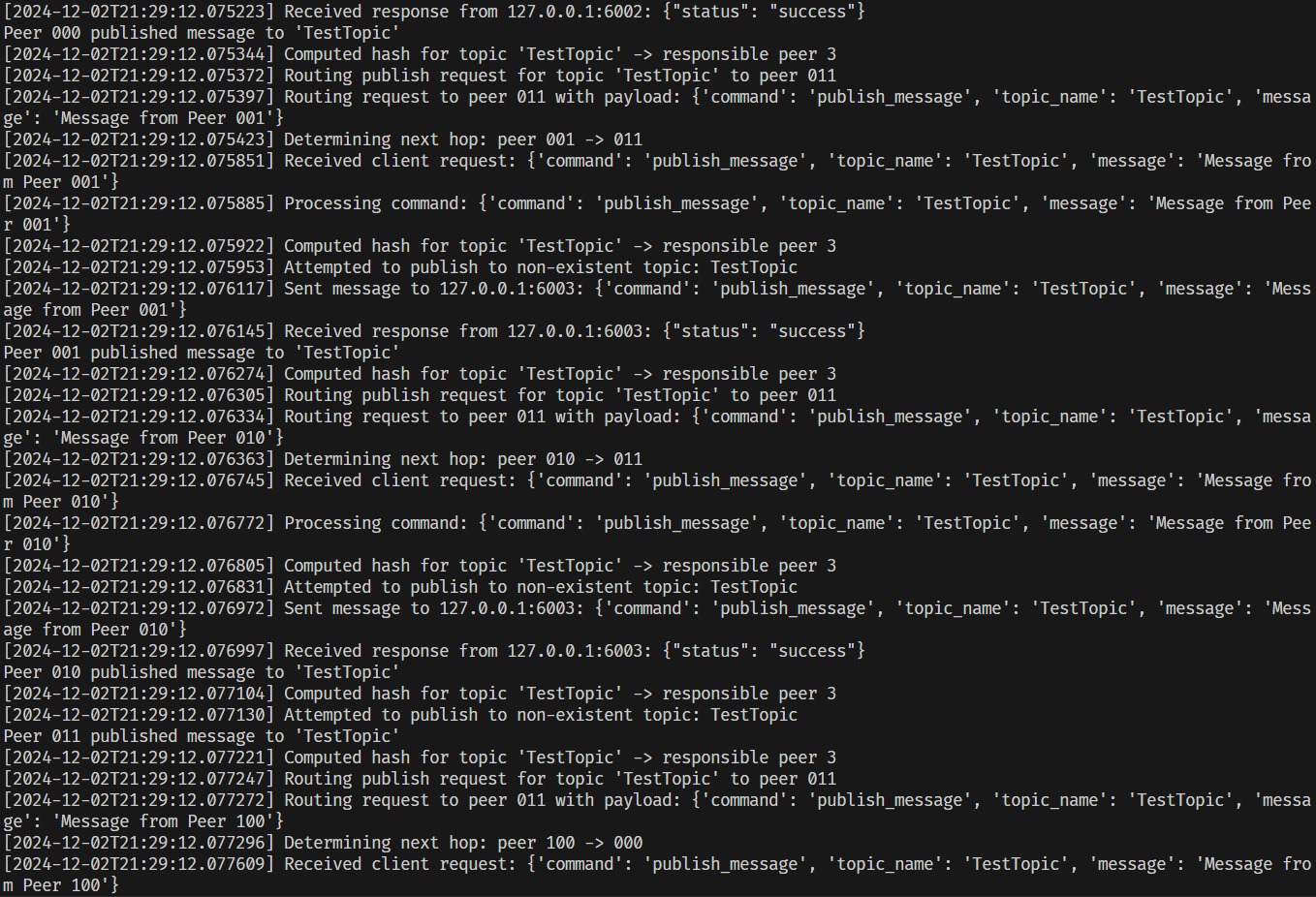
**b. Ensure multiple peer nodes can simultaneously publish and subscribe to a topic.**

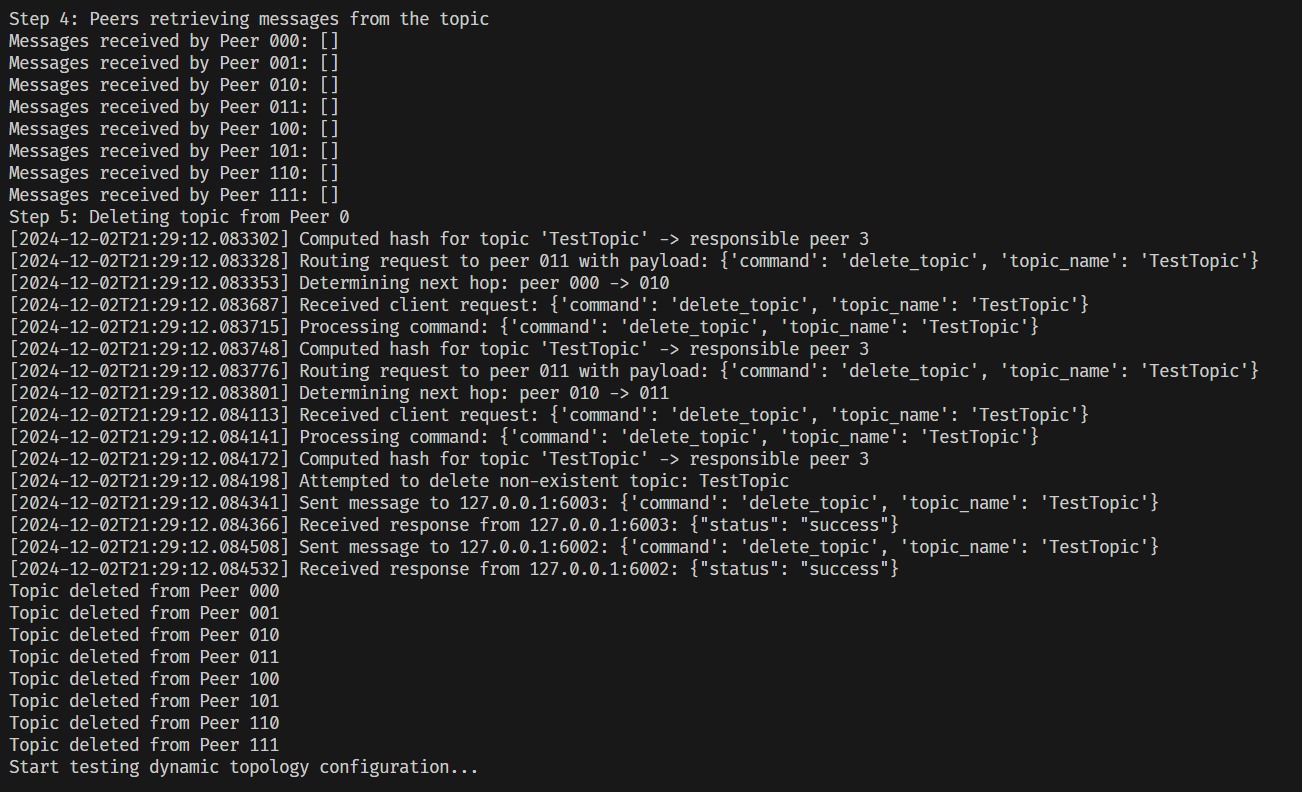




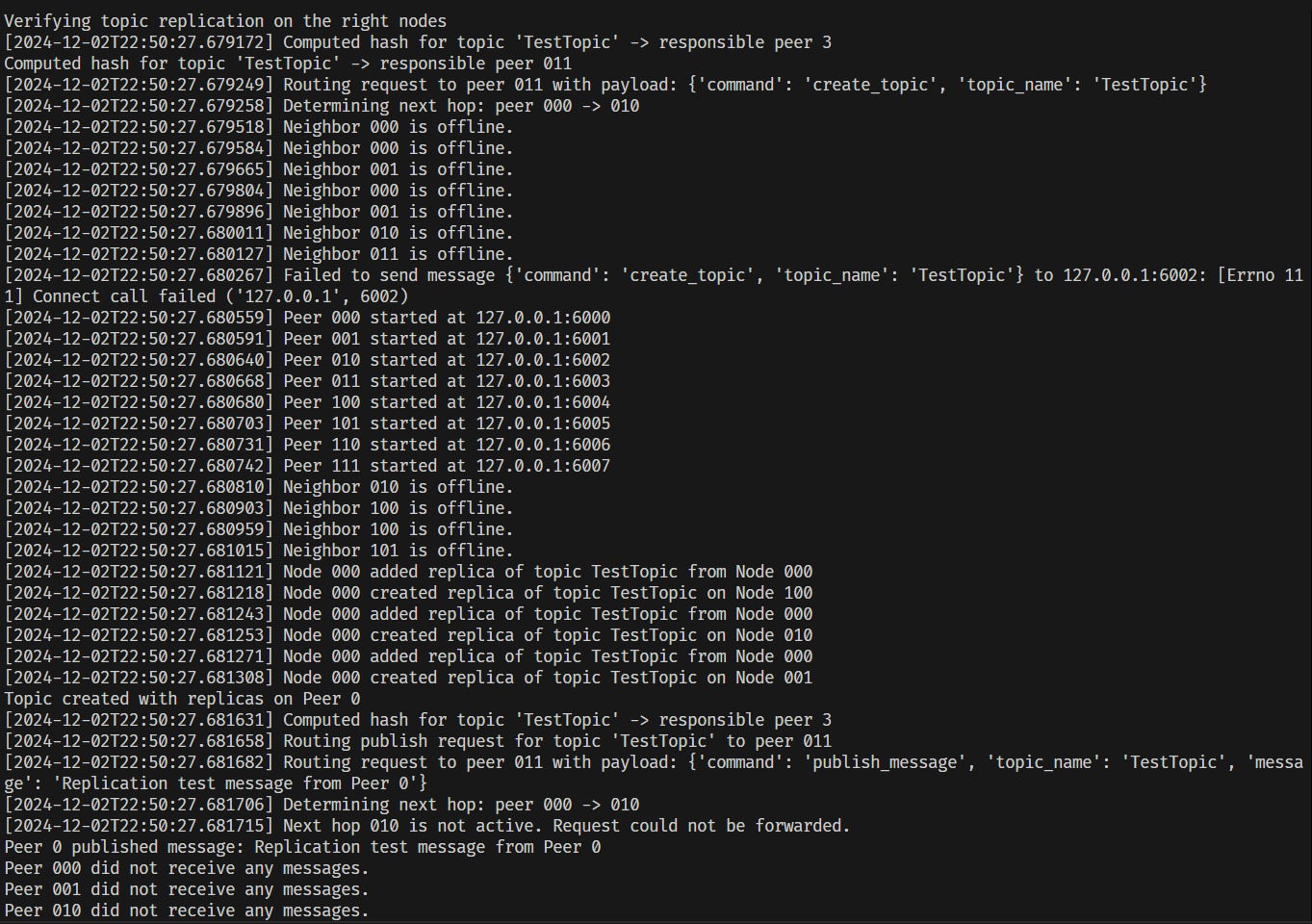




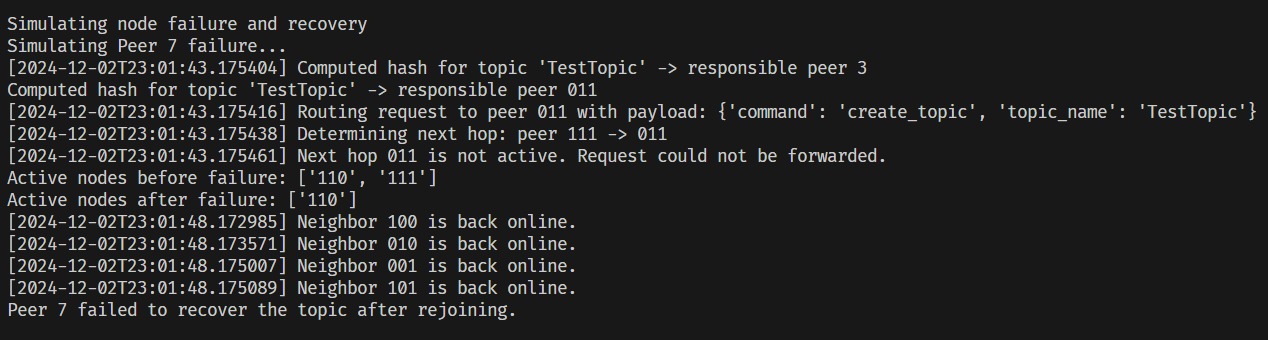




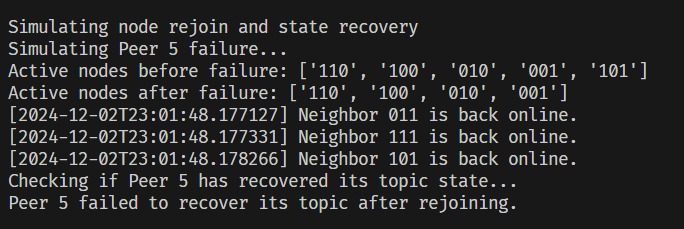
**3.2 Experiment 2: Deploy enough peers. Prove topics can be correctly replicated on the right node.**



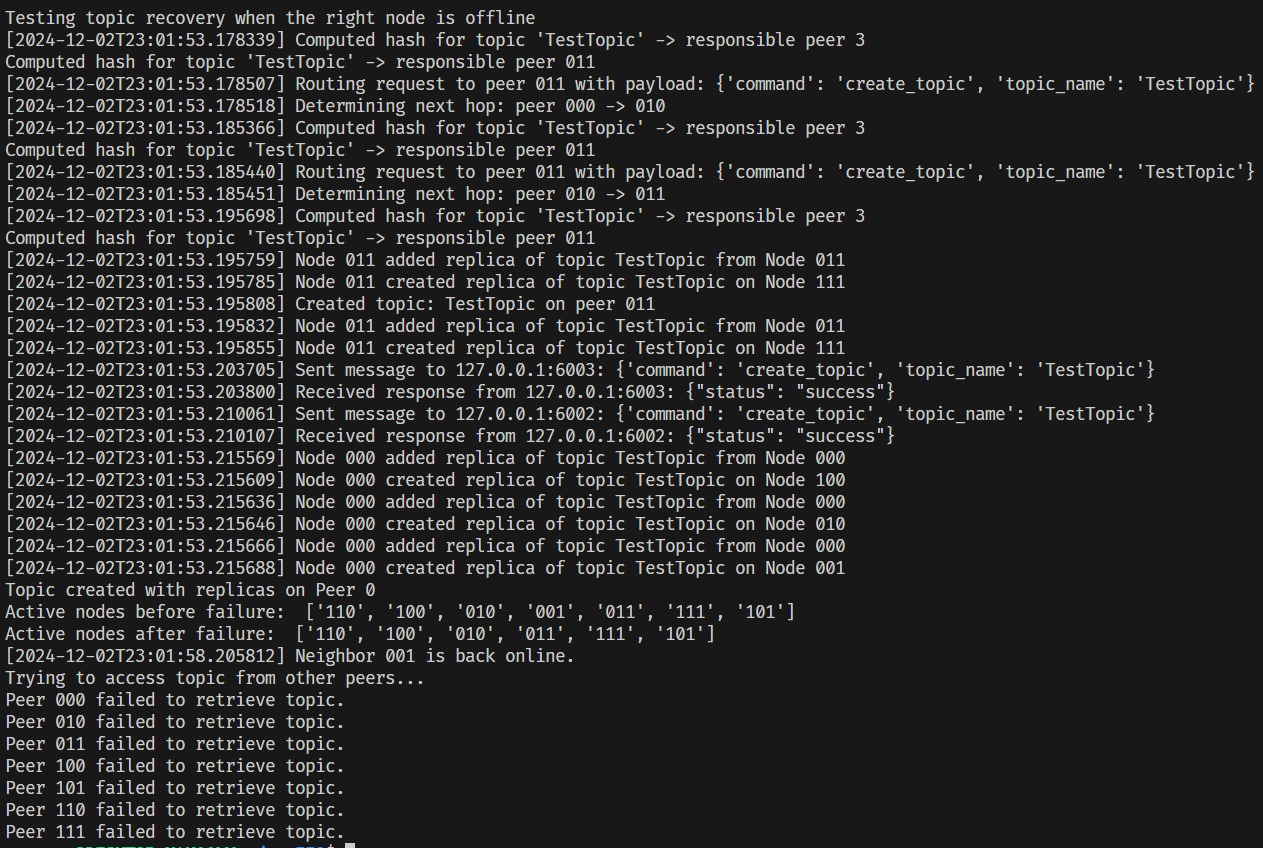
**3.3 Experiment 3: Deploy enough peers. Prove a failed node can recover to the state before the failure.**



**3.4 Experiment 4: Deploy enough peers. Prove when a node comes online, it can recover to the state before failure (if that’s the case) and fetch all topics belonging to it from other (if exists).**



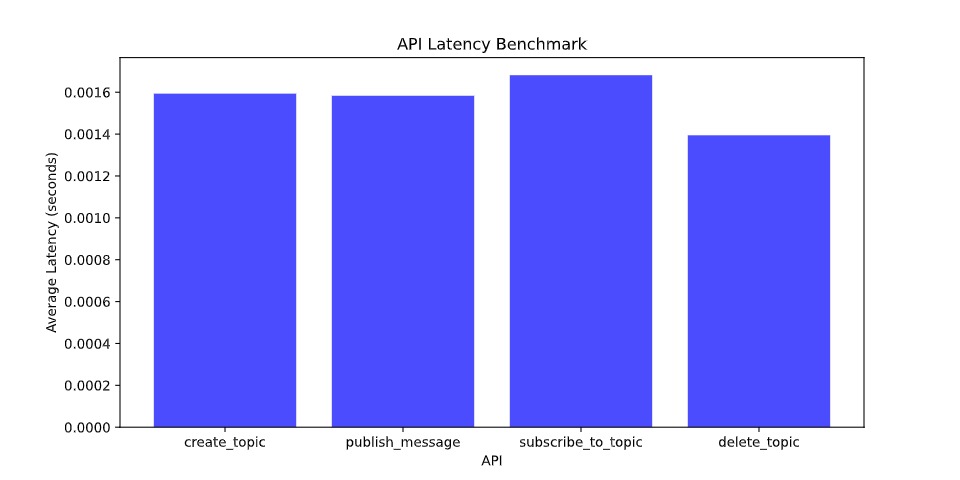
**3.5 Experiment 5: Deploy enough peers. Prove if the “right” node for a given topic is offline, you can correctly find its replica and talk to the host node.**

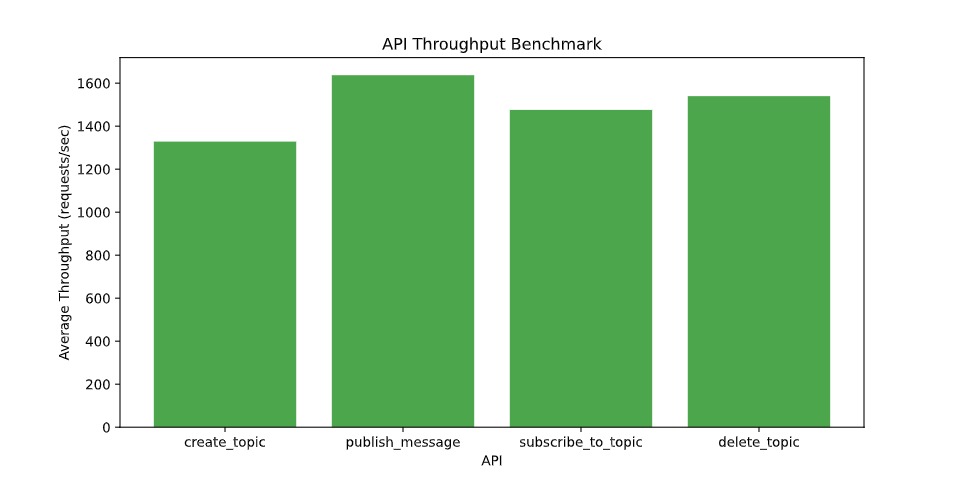


**3.6 Experiment 6: Similar to PA2, you need to benchmark the latency and throughput of each API.**

**a. Deploy 8 peers. Benchmark each API on each node using randomly generated workload.**

**b. Graph your results**





**3.7 Experiment 7: Use sleep() in your code to manually control the communication latency between nodes. Explore in what scenarios your system is faster than your PA3 code.**

This experiment involves introducing artificial communication delays using sleep() to manually control latency between nodes in the network. The goal is to identify scenarios where the new system (with replication and fault tolerance) outperforms the PA3 implementation.

**Steps to Conduct the Experiment**

**(1) Introduce Latency**:

* + Modify your code to include await asyncio.sleep(latency) in network-related functions such as send\_message or route\_request.
  + Test with varying levels of latency, such as 10ms, 50ms, 100ms, and 200ms, to simulate different network conditions.

**(2) Benchmark Scenarios**:

* + Measure the time taken for basic operations like creating topics, publishing messages, subscribing to topics, and deleting topics.
  + Compare the results between the new system and the PA3 implementation.

**(3) Scenarios to Evaluate**:

* + **Low Latency (e.g., 10ms)**: Measure the baseline performance when communication is relatively fast.
  + **Moderate Latency (e.g., 50ms)**: Assess how performance is impacted when network delays start to grow.
  + **High Latency (e.g., 100ms or more)**: Examine whether the replication-based optimizations mitigate the impact of slow communication.

1. Discussion

**6.1 Findings**

* **Trade-offs Between Consistency and Performance**: Maintaining replicas introduces consistency challenges, especially in concurrent write scenarios. Synchronization overhead becomes evident when multiple nodes update the same topic simultaneously. The chosen replication factor affects both fault tolerance and performance. A higher replication factor improves fault tolerance but increases synchronization costs.
* **Performance Gains with Replication**: By leveraging replicas, the system exhibits faster response times for read-intensive operations like subscriptions in high-latency scenarios. This advantage becomes more pronounced as network latency increases. The hash function distributes topics evenly across peers and operates efficiently.
* **Dynamic Topology Adaptation**: The system supports the addition and removal of nodes during runtime without disrupting operations. New nodes quickly integrate into the network and start hosting topics or replicas based on the updated hashing mechanism. The hypercube topology efficiently recalculates neighbors and routes requests even as the topology changes dynamically.

**6.2 Possible Improvements**

* **Dynamic Replication:** Adjust replication strategies based on observed latencies or workload patterns to optimize performance further.
* **Caching:** Use a lightweight caching mechanism for frequently accessed topics to complement replication and reduce latency further.
* **Latency Prediction:** Incorporate a latency prediction mechanism to intelligently route requests to the fastest available nodes or replicas.

1. Conclusion

This project successfully extends a peer-to-peer (P2P) system with advanced features, including topic replication, fault tolerance, and dynamic topology configuration, enabling the system to perform reliably and efficiently in dynamic and potentially adverse conditions.

The enhanced system demonstrates significant improvements in fault tolerance, scalability, and adaptability compared to its predecessor (PA3). It effectively handles node failures, ensures uninterrupted topic access through replication, and supports the addition and removal of nodes during runtime without disrupting operations. The hypercube topology, coupled with asynchronous programming, enables efficient routing and concurrency, making the system robust in handling increased workloads and network size.

However, the implementation also reveals trade-offs between consistency and performance. Replication improves read access times but introduces overhead during write operations. Synchronization and consistency management are critical areas where further optimization is possible.