COMPUTER NETWORKS
ASSIGNMENT 1

Gossip-Based Peer-to-Peer Network with Two-Level Consensus

Vadlamudi Jyothsna (B23CS1076)
Jandhyam Sai Sriya(B23CS1021)

1. Introduction

This project implements a **Gossip-based Peer-to-Peer (P2P) network** supporting:

- Reliable message dissemination
- Power-law overlay topology
- Peer-level liveness detection
- Seed-level consensus-based membership management
- Protection against unilateral and malicious node decisions

Unlike a traditional P2P system, this implementation enforces:

- Majority-based seed registration
- Majority-based dead-node removal
- Two-level consensus (peer-level + seed-level)

2. System Architecture

The system consists of two types of nodes:

2.1 Seed Nodes

- Maintain Peer List (PL)
- Participate in consensus for:
  - Peer addition
  - Peer removal
- Do NOT participate in gossip

Responsibilities:

- Receive peer registration proposals
- Exchange votes with other seeds
- Commit membership only after quorum
- Log all consensus events

2.2 Peer Nodes

- Read seed list from config file
- Register with $\lfloor n/2 \rfloor + 1$ seeds
- Form overlay with power-law degree distribution
- Disseminate gossip
- Detect neighbor failures
- Perform peer-level suspicion consensus

3. Network Setup

Configuration File Example

```
127.0.0.1:6000
127.0.0.1:6001
127.0.0.1:6002
```

If n = 3 seeds:

Quorum = ⌊3/2⌋ + 1 = 2

A peer must register with at least 2 seeds.

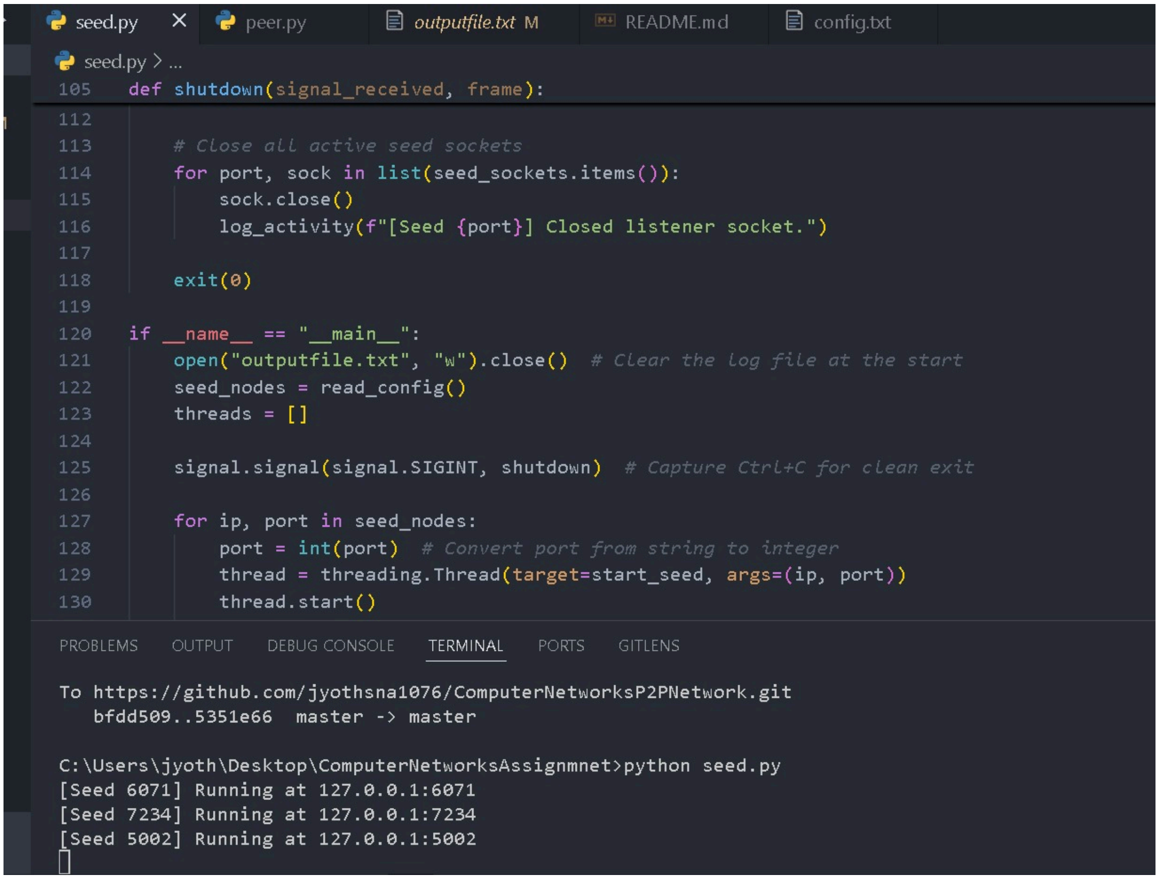
4. Overlay Network Formation (Power-Law Topology)

To achieve power-law degree distribution:

- Peer obtains union of Peer Lists
- Uses **preferential attachment**
- Higher-degree nodes have higher probability of selection

Why Power Law?

- Real-world P2P networks follow power-law
- Improves robustness against random failures
- Maintains small-world properties



```python
def shutdown(signal_received, frame):

    # Close all active seed sockets
    for port, sock in list(seed_sockets.items()):
        sock.close()
        log_activity(f"[Seed {port}] Closed listener socket.")

    exit(0)

if __name__ == "__main__":
    open("outputfile.txt", "w").close()  # Clear the log file at the start
    seed_nodes = read_config()
    threads = []

    signal.signal(signal.SIGINT, shutdown)  # Capture Ctrl+C for clean exit

    for ip, port in seed_nodes:
        port = int(port)  # Convert port from string to integer
        thread = threading.Thread(target=start_seed, args=(ip, port))
        thread.start()
```

```
To https://github.com/jyothsna1076/ComputerNetworksP2PNetwork.git
   bfdd509..5351e66  master -> master

C:\Users\jyoth\Desktop\ComputerNetworksAssignmnet>python seed.py
[Seed 6071] Running at 127.0.0.1:6071
[Seed 7234] Running at 127.0.0.1:7234
[Seed 5002] Running at 127.0.0.1:5002
```

Seed Nodes Running

- 3 seed terminals
- Console logs
- Registration proposals
- Consensus messages

Example output:

```
Seed running on port 6000
PROPOSE_ADD: 127.0.0.1:7001
ADD CONSENSUS ACHIEVED: 127.0.0.1:7001
```

5. Gossip Protocol

Message Format

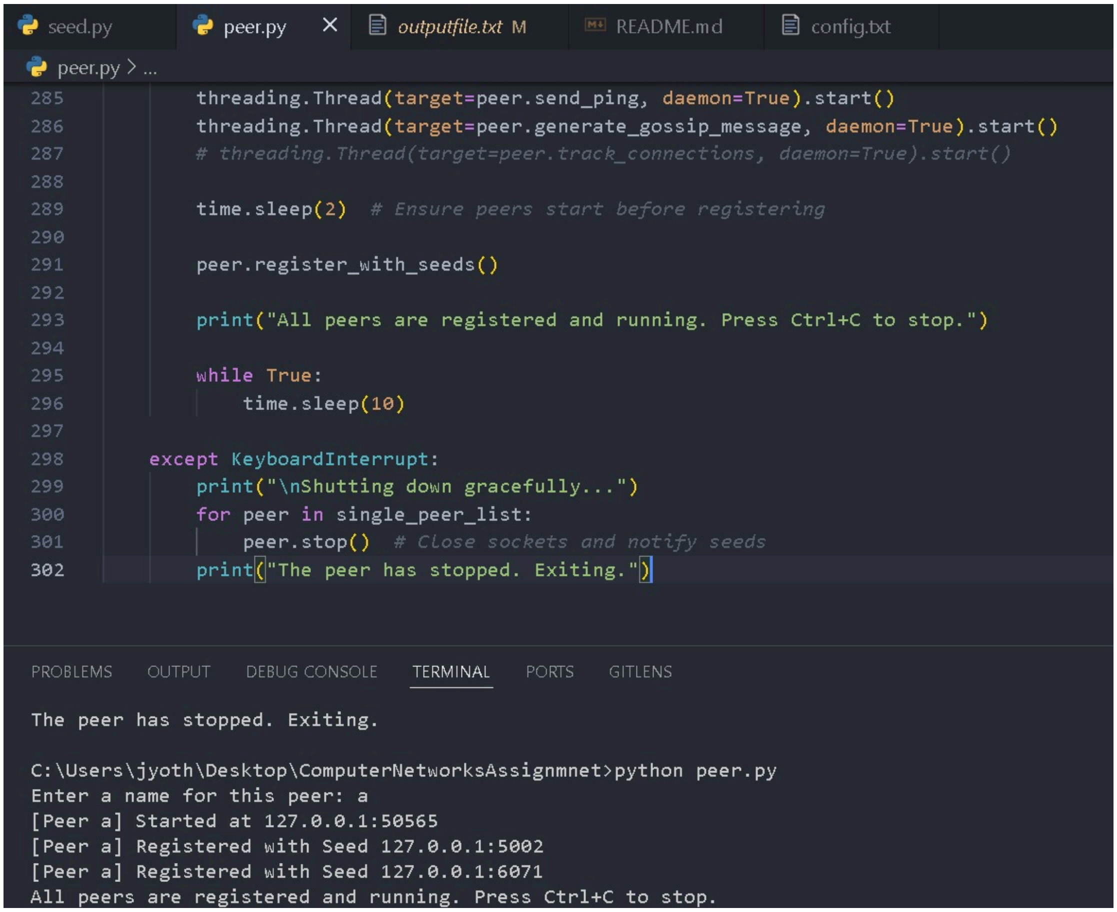
<timestamp>:<IP>:<Msg#>

Example:

```
1717689000:127.0.0.1:3
```

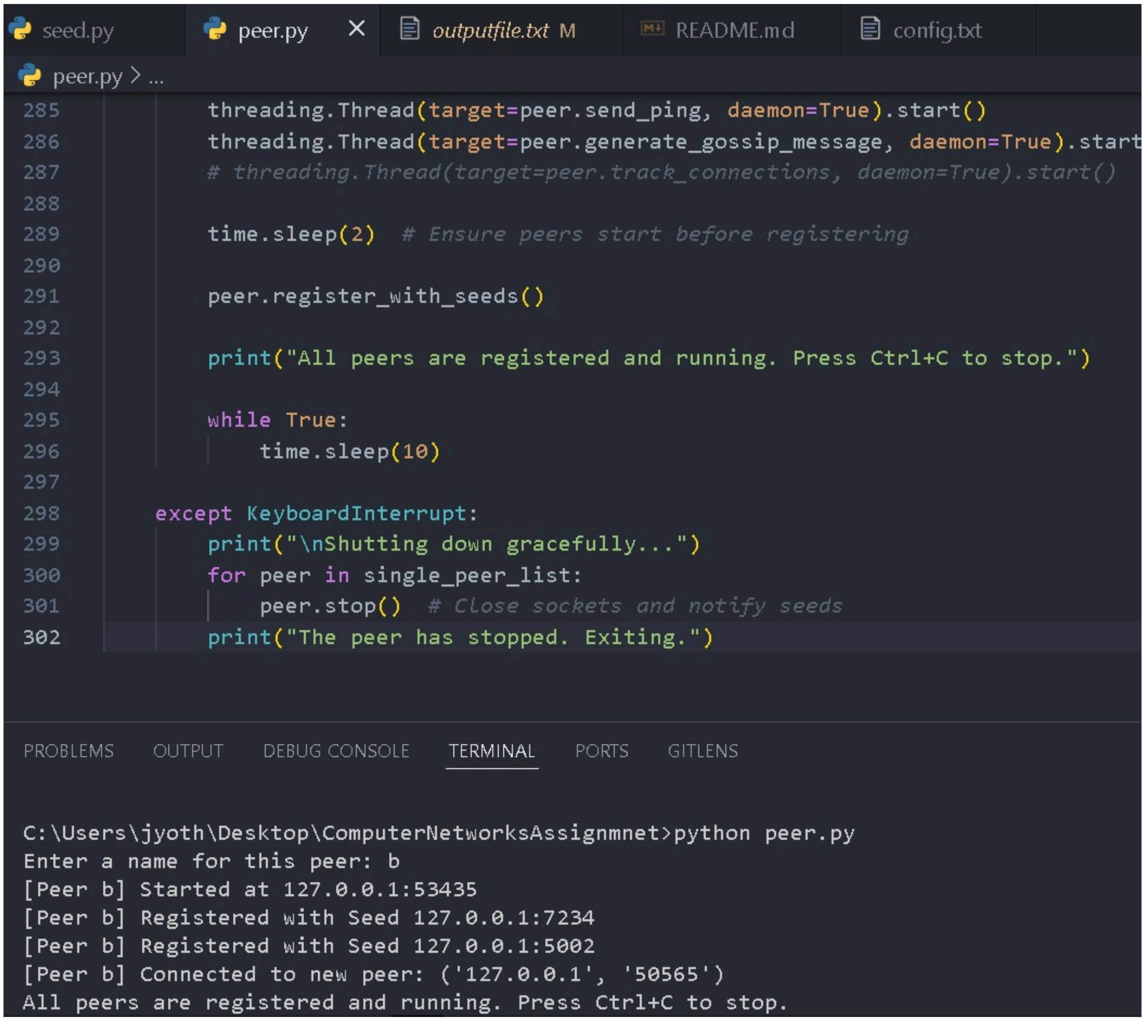
Gossip Rules

- Generated every 5 seconds
- Maximum 10 messages per peer
- Stored in Message List (ML)
- Forwarded to all neighbors except sender
- Duplicate messages ignored



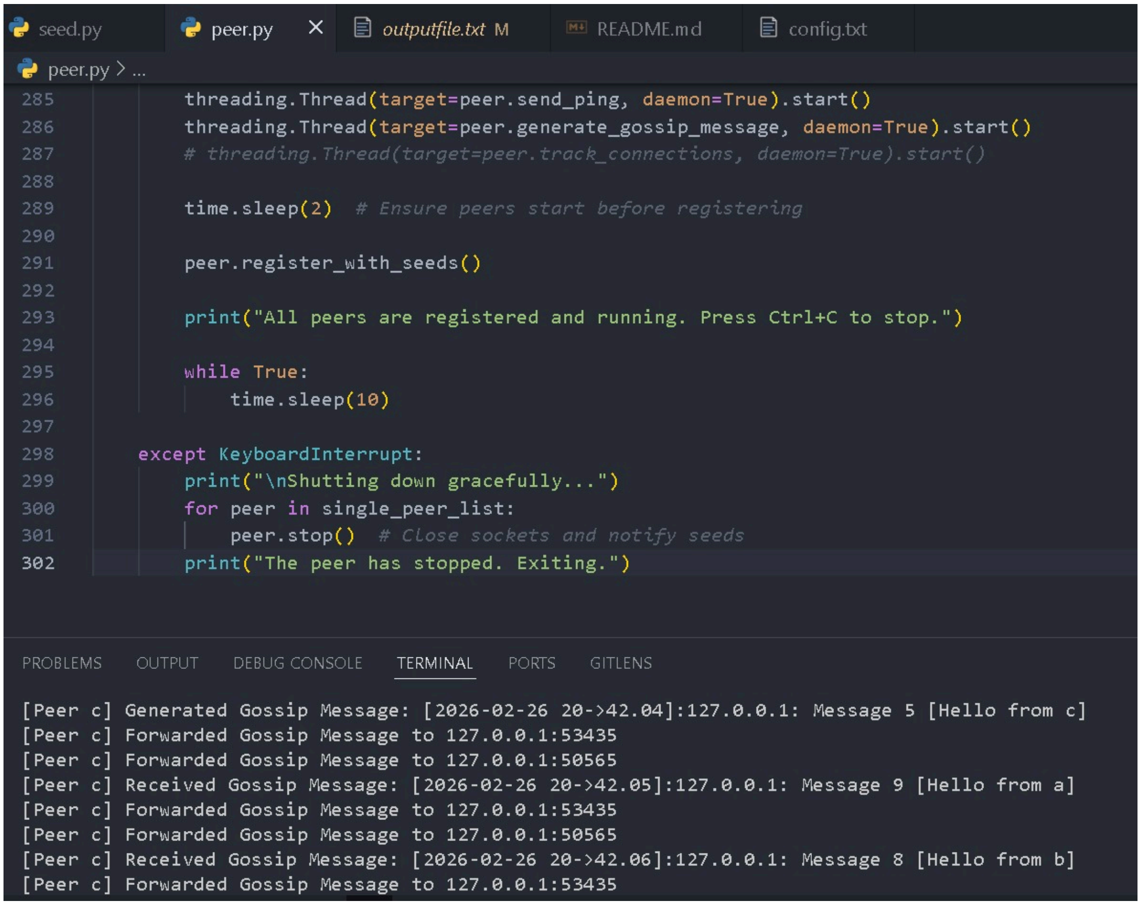
```python
        threading.Thread(target=peer.send_ping, daemon=True).start()
        threading.Thread(target=peer.generate_gossip_message, daemon=True).start()
        # threading.Thread(target=peer.track_connections, daemon=True).start()

        time.sleep(2)  # Ensure peers start before registering

        peer.register_with_seeds()

        print("All peers are registered and running. Press Ctrl+C to stop.")

        while True:
            time.sleep(10)

    except KeyboardInterrupt:
        print("\nShutting down gracefully...")
        for peer in single_peer_list:
            peer.stop()  # Close sockets and notify seeds
        print("The peer has stopped. Exiting.")
```

```
The peer has stopped. Exiting.

C:\Users\jyoth\Desktop\ComputerNetworksAssignmnet>python peer.py
Enter a name for this peer: a
[Peer a] Started at 127.0.0.1:50565
[Peer a] Registered with Seed 127.0.0.1:5002
[Peer a] Registered with Seed 127.0.0.1:6071
All peers are registered and running. Press Ctrl+C to stop.
```



```python
        threading.Thread(target=peer.send_ping, daemon=True).start()
        threading.Thread(target=peer.generate_gossip_message, daemon=True).start
        # threading.Thread(target=peer.track_connections, daemon=True).start()

        time.sleep(2)  # Ensure peers start before registering

        peer.register_with_seeds()

        print("All peers are registered and running. Press Ctrl+C to stop.")

        while True:
            time.sleep(10)

    except KeyboardInterrupt:
        print("\nShutting down gracefully...")
        for peer in single_peer_list:
            peer.stop()  # Close sockets and notify seeds
        print("The peer has stopped. Exiting.")
```

```
C:\Users\jyoth\Desktop\ComputerNetworksAssignmnet>python peer.py
Enter a name for this peer: b
[Peer b] Started at 127.0.0.1:53435
[Peer b] Registered with Seed 127.0.0.1:7234
[Peer b] Registered with Seed 127.0.0.1:5002
[Peer b] Connected to new peer: ('127.0.0.1', '50565')
All peers are registered and running. Press Ctrl+C to stop.
```



```python
        threading.Thread(target=peer.send_ping, daemon=True).start()
        threading.Thread(target=peer.generate_gossip_message, daemon=True).start()
        # threading.Thread(target=peer.track_connections, daemon=True).start()

        time.sleep(2)  # Ensure peers start before registering

        peer.register_with_seeds()

        print("All peers are registered and running. Press Ctrl+C to stop.")

        while True:
            time.sleep(10)

    except KeyboardInterrupt:
        print("\nShutting down gracefully...")
        for peer in single_peer_list:
            peer.stop()  # Close sockets and notify seeds
        print("The peer has stopped. Exiting.")
```

```
[Peer c] Generated Gossip Message: [2026-02-26 20->42.04]:127.0.0.1: Message 5 [Hello from c]
[Peer c] Forwarded Gossip Message to 127.0.0.1:53435
[Peer c] Forwarded Gossip Message to 127.0.0.1:50565
[Peer c] Received Gossip Message: [2026-02-26 20->42.05]:127.0.0.1: Message 9 [Hello from a]
[Peer c] Forwarded Gossip Message to 127.0.0.1:53435
[Peer c] Forwarded Gossip Message to 127.0.0.1:50565
[Peer c] Received Gossip Message: [2026-02-26 20->42.06]:127.0.0.1: Message 8 [Hello from b]
[Peer c] Forwarded Gossip Message to 127.0.0.1:53435
```

Gossip Dissemination Across Peers

- Peer a generating message
- Peer b receiving it
- Peer c receiving it

Console example:

Generated Gossip: 1717689000:127.0.0.1:1
First time received: 1717689000:127.0.0.1:1

6. Peer-Level Liveness Detection

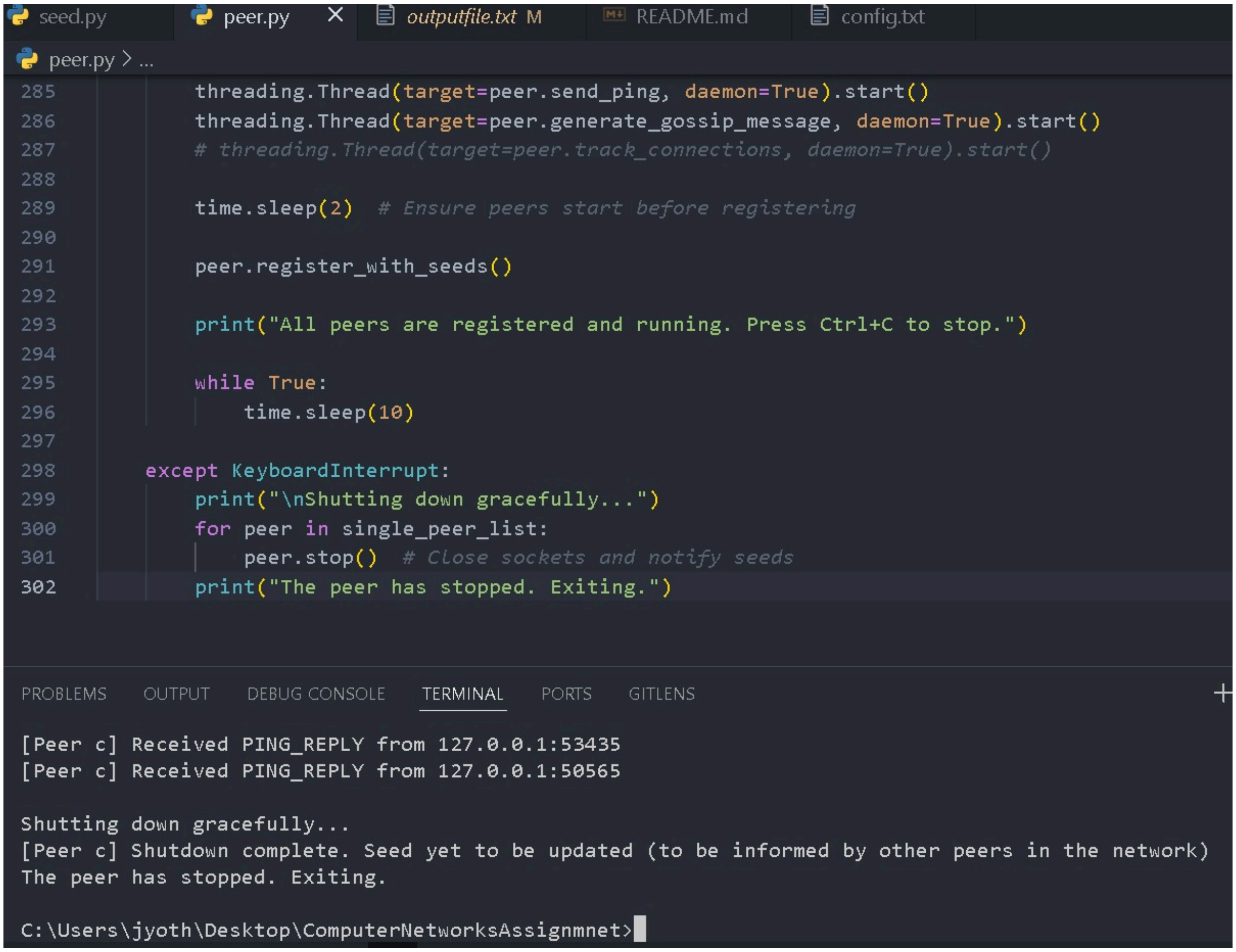
Peers periodically:

- Send PING every 13 seconds
- If no response for 2 intervals → mark suspected

- Share suspicion with neighbors

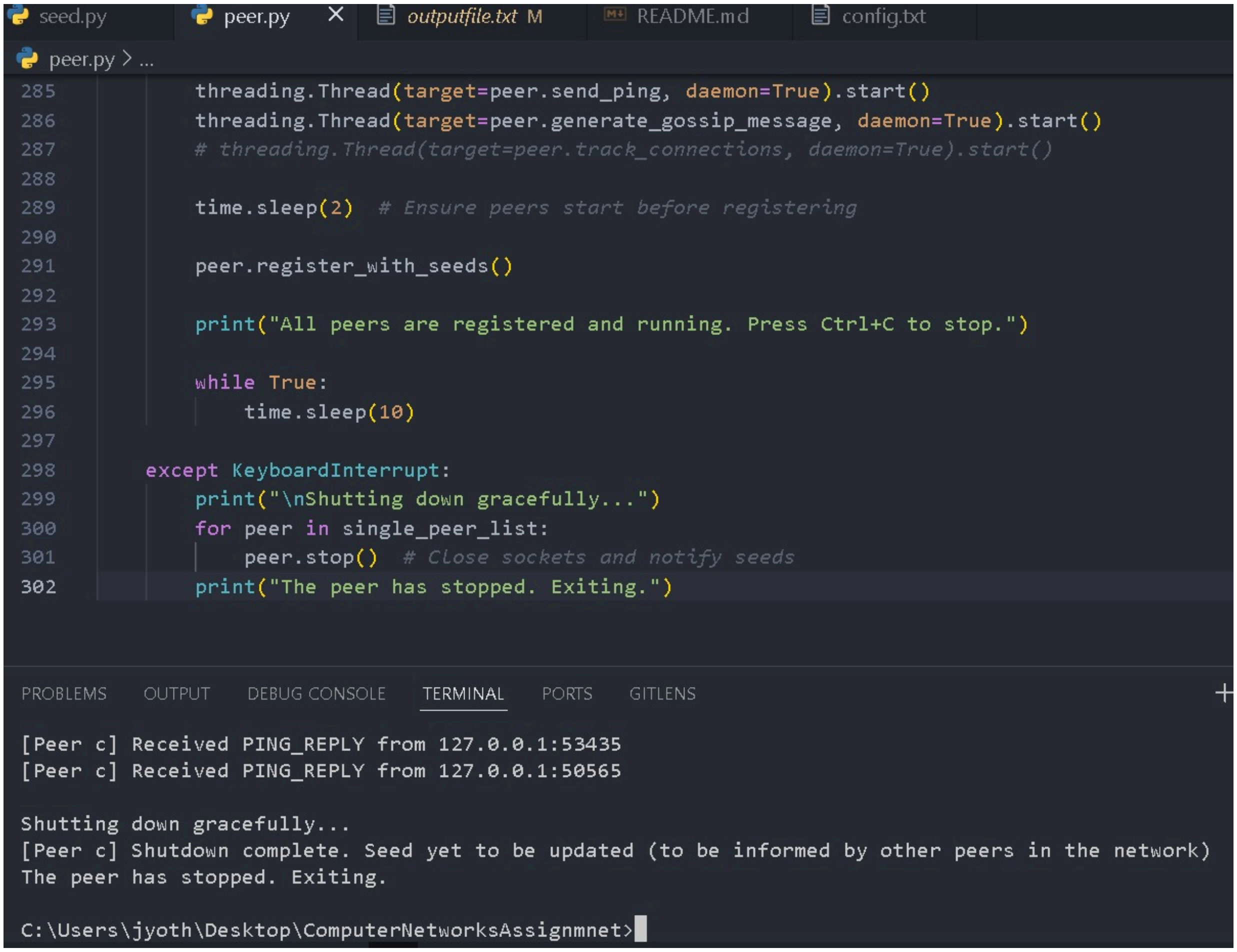
A dead-node report is generated only if:

Majority of neighbors confirm suspicion.



```python
        threading.Thread(target=peer.send_ping, daemon=True).start()
        threading.Thread(target=peer.generate_gossip_message, daemon=True).start()
        # threading.Thread(target=peer.track_connections, daemon=True).start()

        time.sleep(2)  # Ensure peers start before registering

        peer.register_with_seeds()

        print("All peers are registered and running. Press Ctrl+C to stop.")

        while True:
            time.sleep(10)

    except KeyboardInterrupt:
        print("\nShutting down gracefully...")
        for peer in single_peer_list:
            peer.stop()  # Close sockets and notify seeds
        print("The peer has stopped. Exiting.")
```

```
[Peer c] Received PING_REPLY from 127.0.0.1:53435
[Peer c] Received PING_REPLY from 127.0.0.1:50565

Shutting down gracefully...
[Peer c] Shutdown complete. Seed yet to be updated (to be informed by other peers in the network)
The peer has stopped. Exiting.

C:\Users\jyoth\Desktop\ComputerNetworksAssignmnet>
```



```python
        threading.Thread(target=peer.send_ping, daemon=True).start()
        threading.Thread(target=peer.generate_gossip_message, daemon=True).start()
        # threading.Thread(target=peer.track_connections, daemon=True).start()

        time.sleep(2)  # Ensure peers start before registering

        peer.register_with_seeds()

        print("All peers are registered and running. Press Ctrl+C to stop.")

        while True:
            time.sleep(10)

    except KeyboardInterrupt:
        print("\nShutting down gracefully...")
        for peer in single_peer_list:
            peer.stop()  # Close sockets and notify seeds
        print("The peer has stopped. Exiting.")
```

```
[Peer c] Received PING_REPLY from 127.0.0.1:53435
[Peer c] Received PING_REPLY from 127.0.0.1:50565

Shutting down gracefully...
[Peer c] Shutdown complete. Seed yet to be updated (to be informed by other peers in the network)
The peer has stopped. Exiting.

C:\Users\jyoth\Desktop\ComputerNetworksAssignmnet>
```

Peer-Level Suspicion

- Kill one peer
- Other peers detect failure
- Show suspicion message

Example:

Suspecting 127.0.0.1:7003
Neighbor consensus achieved
Reporting to seeds...

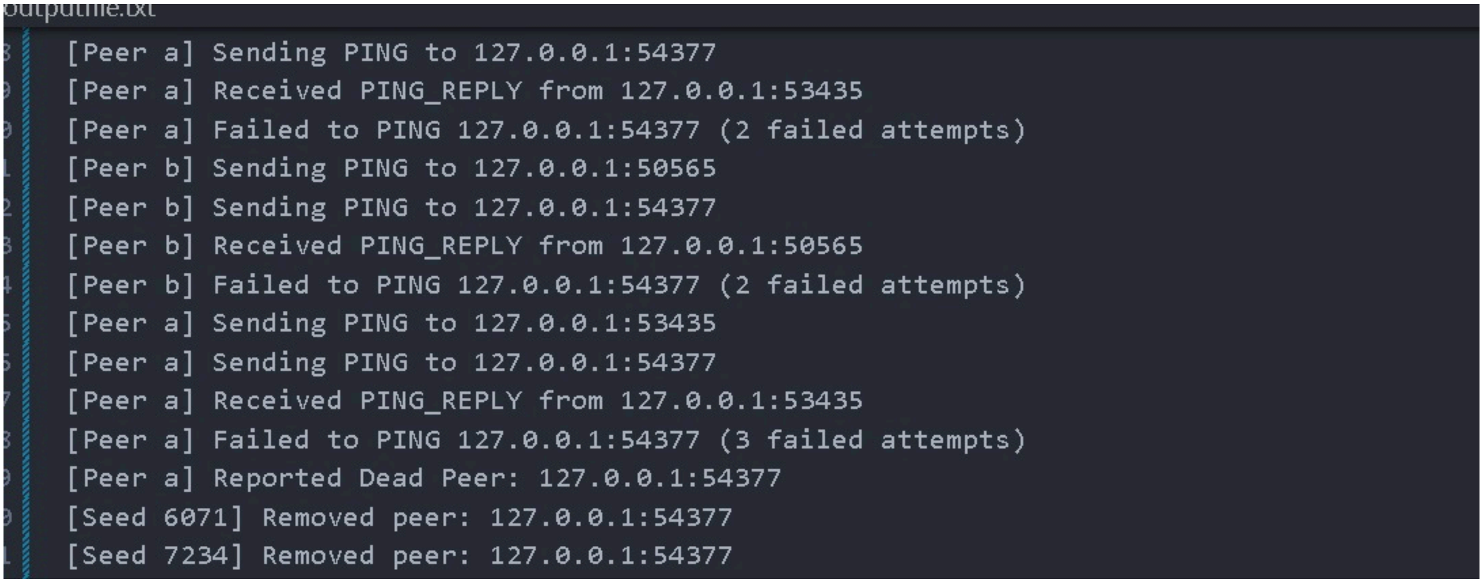
7. Seed-Level Consensus

When seeds receive:

Dead Node:<IP>:<Port>:<timestamp>:<ReporterIP>

Seeds:

1. Exchange removal proposals
2. Vote
3. Remove only if quorum achieved



```
[Peer a] Sending PING to 127.0.0.1:54377
[Peer a] Received PING_REPLY from 127.0.0.1:53435
[Peer a] Failed to PING 127.0.0.1:54377 (2 failed attempts)
[Peer b] Sending PING to 127.0.0.1:50565
[Peer b] Sending PING to 127.0.0.1:54377
[Peer b] Received PING_REPLY from 127.0.0.1:50565
[Peer b] Failed to PING 127.0.0.1:54377 (2 failed attempts)
[Peer a] Sending PING to 127.0.0.1:53435
[Peer a] Sending PING to 127.0.0.1:54377
[Peer a] Received PING_REPLY from 127.0.0.1:53435
[Peer a] Failed to PING 127.0.0.1:54377 (3 failed attempts)
[Peer a] Reported Dead Peer: 127.0.0.1:54377
[Seed 6071] Removed peer: 127.0.0.1:54377
[Seed 7234] Removed peer: 127.0.0.1:54377
```

Seed Consensus for Dead Node Removal

Example output:

PROPOSE_REMOVE: 127.0.0.1:7003
REMOVE CONSENSUS ACHIEVED
Peer removed from PL

8. Two-Level Consensus Model

This system prevents:

- False accusations
- Malicious node removal
- Sybil-style attacks

Level 1 – Peer Consensus

Multiple neighbors must confirm suspicion.

Level 2 – Seed Consensus

Majority seeds must agree before removal.

9. Security Analysis

9.1 Attack: Malicious Peer Reports Fake Death

Mitigation:

- Requires neighbor majority
- Requires seed majority
- Single node cannot remove another

9.2 Attack: Colluding Peers

Mitigation:

- Requires seed consensus
- Seeds operate independently

9.3 Sybil Attack

Mitigation:

- Requires quorum registration
- Seeds verify majority agreement

10. Experimental Evaluation

Test Environment:

- OS: Windows / Linux
- Implementation: Python (friend’s version)
- Number of Seeds: 3
- Number of Peers: 5

Results:

- Gossip reaches all peers within seconds
- No duplicate flooding
- Dead-node removal consistent across all seeds
- Overlay remains connected

11. Conclusion

This implementation successfully demonstrates:

- Reliable gossip dissemination
- Power-law overlay construction
- Robust liveness detection
- Two-level consensus for secure membership management

The system is resilient against:

- False failure reports
- Malicious membership manipulation
- Unilateral node deletion

12. Future Improvements

- Digital signatures for reports
- Byzantine fault tolerance
- Cryptographic identity verification
- Dynamic overlay restructuring