Raft Testing

Overview

The flow of client requests to create a report to the raft cluster is as follows:
Client → API Gateway → Business Logic Layer → Data Access Layer (NGINX → Raft Cluster)

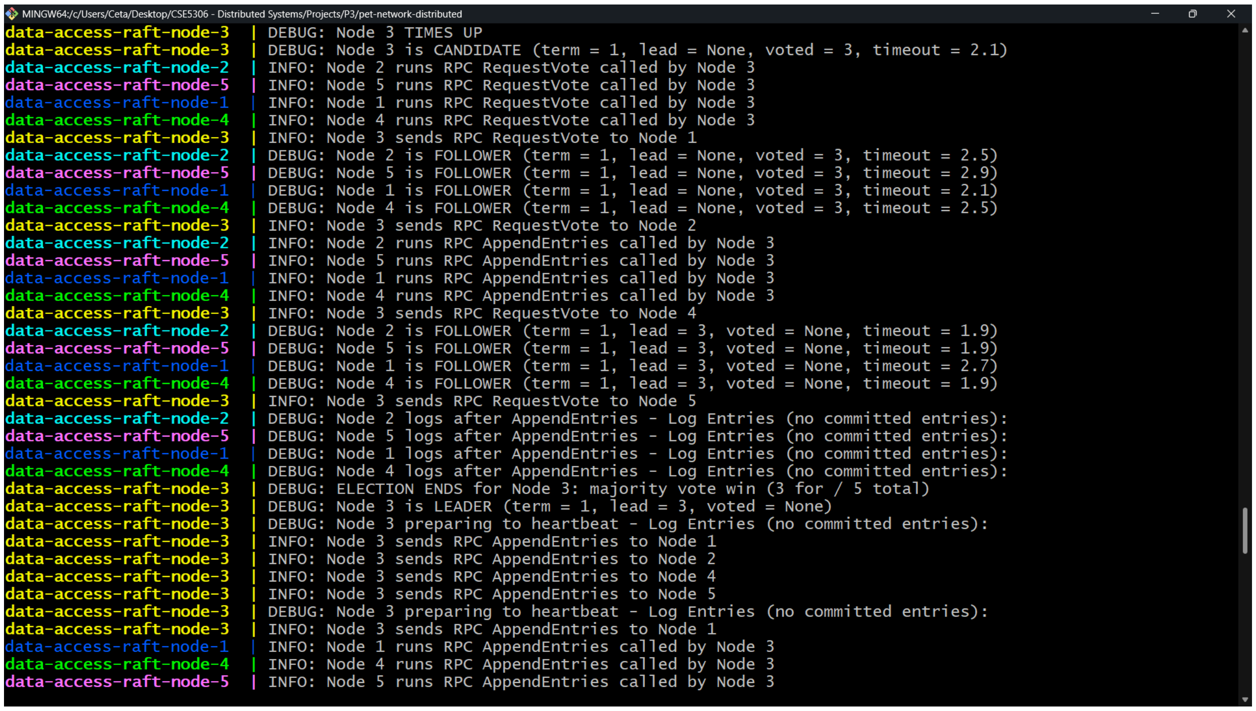
Raft testing is done with the following configurations and assumptions:

- The raft node logger level is set to DEBUG, so line 14 of the `pet-network-distributed/layered/data-access/raft/simple_raft.py` file is `logger.setLevel(logging.DEBUG)`
    - The default value is `logging.INFO`, so be sure to change this if you wish to see similar results.
- `docker-compose -f docker-compose-layered-raft.yml up --build` is ran in the parent directory `pet-network-distributed/`
- As mentioned previously in the overview, only the leader node will execute client commands
- Each test case is tested using different instances of the raft cluster
- A mixture of logs from Docker Desktop along with terminal output is used to analyze and evaluate behavior

Note that it is very unlikely for logs to be similar between raft cluster executions due to the following reasons:

- A randomized timeout is used for each node (so the node that begins candidacy first is random).
- The election_timeout is set to a value between the interval [1.5, 3] every time it is set and refreshed. So, a single node will have different election_timeouts during its lifetime.
    - The logs truncate the timeout values to the tenths place, so if a log shows 2.3 twice, it's likely that they are different values in terms of ms or a more precise unit of measurement.
- NGINX is used to forward HTTP requests from the business layer to the raft cluster. NGINX uses a round-robin forward scheme, but there is no guarantee to know which raft node will receive the HTTP request. However, the request is expected to be served if any raft node receives it and there is a leader.

The image below shows what you can expect from the raft cluster when the `docker-compose` command is executed and raft nodes are running (with `logging.DEBUG`):

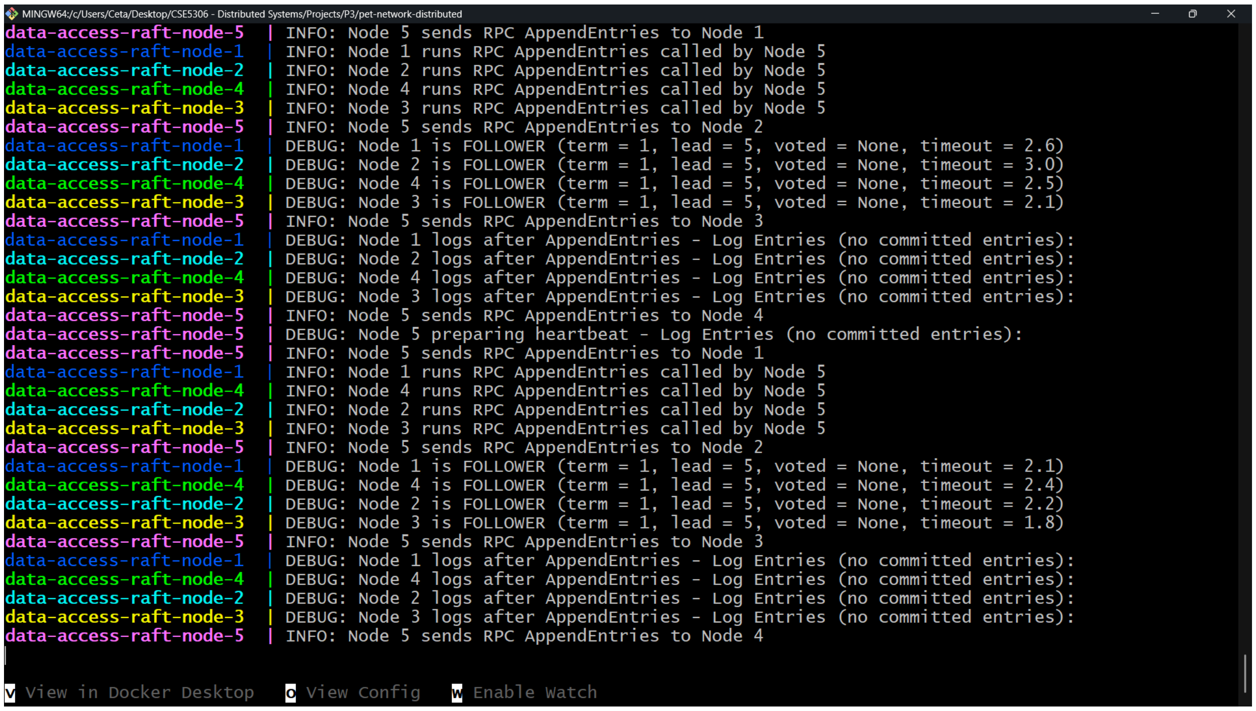


```
MINGW64:/c/Users/Ceta/Desktop/CSE5306 - Distributed Systems/Projects/P3/pet-network-distributed
data-access-raft-node-3  | DEBUG: Node 3 TIMES UP
data-access-raft-node-3  | DEBUG: Node 3 is CANDIDATE (term = 1, lead = None, voted = 3, timeout = 2.1)
data-access-raft-node-2  | INFO: Node 2 runs RPC RequestVote called by Node 3
data-access-raft-node-5  | INFO: Node 5 runs RPC RequestVote called by Node 3
data-access-raft-node-1  | INFO: Node 1 runs RPC RequestVote called by Node 3
data-access-raft-node-4  | INFO: Node 4 runs RPC RequestVote called by Node 3
data-access-raft-node-3  | INFO: Node 3 sends RPC RequestVote to Node 1
data-access-raft-node-2  | DEBUG: Node 2 is FOLLOWER (term = 1, lead = None, voted = 3, timeout = 2.5)
data-access-raft-node-5  | DEBUG: Node 5 is FOLLOWER (term = 1, lead = None, voted = 3, timeout = 2.9)
data-access-raft-node-1  | DEBUG: Node 1 is FOLLOWER (term = 1, lead = None, voted = 3, timeout = 2.1)
data-access-raft-node-4  | DEBUG: Node 4 is FOLLOWER (term = 1, lead = None, voted = 3, timeout = 2.5)
data-access-raft-node-3  | INFO: Node 3 sends RPC RequestVote to Node 2
data-access-raft-node-2  | INFO: Node 2 runs RPC AppendEntries called by Node 3
data-access-raft-node-5  | INFO: Node 5 runs RPC AppendEntries called by Node 3
data-access-raft-node-1  | INFO: Node 1 runs RPC AppendEntries called by Node 3
data-access-raft-node-4  | INFO: Node 4 runs RPC AppendEntries called by Node 3
data-access-raft-node-3  | INFO: Node 3 sends RPC RequestVote to Node 4
data-access-raft-node-2  | DEBUG: Node 2 is FOLLOWER (term = 1, lead = 3, voted = None, timeout = 1.9)
data-access-raft-node-5  | DEBUG: Node 5 is FOLLOWER (term = 1, lead = 3, voted = None, timeout = 1.9)
data-access-raft-node-1  | DEBUG: Node 1 is FOLLOWER (term = 1, lead = 3, voted = None, timeout = 2.7)
data-access-raft-node-4  | DEBUG: Node 4 is FOLLOWER (term = 1, lead = 3, voted = None, timeout = 1.9)
data-access-raft-node-3  | INFO: Node 3 sends RPC RequestVote to Node 5
data-access-raft-node-2  | DEBUG: Node 2 logs after AppendEntries - Log Entries (no committed entries):
data-access-raft-node-5  | DEBUG: Node 5 logs after AppendEntries - Log Entries (no committed entries):
data-access-raft-node-1  | DEBUG: Node 1 logs after AppendEntries - Log Entries (no committed entries):
data-access-raft-node-4  | DEBUG: Node 4 logs after AppendEntries - Log Entries (no committed entries):
data-access-raft-node-3  | DEBUG: ELECTION ENDS for Node 3: majority vote win (3 for / 5 total)
data-access-raft-node-3  | DEBUG: Node 3 is LEADER (term = 1, lead = 3, voted = None)
data-access-raft-node-3  | DEBUG: Node 3 preparing to heartbeat - Log Entries (no committed entries):
data-access-raft-node-3  | INFO: Node 3 sends RPC AppendEntries to Node 1
data-access-raft-node-3  | INFO: Node 3 sends RPC AppendEntries to Node 2
data-access-raft-node-3  | INFO: Node 3 sends RPC AppendEntries to Node 4
data-access-raft-node-3  | INFO: Node 3 sends RPC AppendEntries to Node 5
data-access-raft-node-3  | DEBUG: Node 3 preparing to heartbeat - Log Entries (no committed entries):
data-access-raft-node-3  | INFO: Node 3 sends RPC AppendEntries to Node 1
data-access-raft-node-1  | INFO: Node 1 runs RPC AppendEntries called by Node 3
data-access-raft-node-4  | INFO: Node 4 runs RPC AppendEntries called by Node 3
data-access-raft-node-5  | INFO: Node 5 runs RPC AppendEntries called by Node 3
```

This output is very difficult to read and process, however this can still be used to see some useful information. For instance, node 3 became a candidate node for term 1, received a majority of votes and became the leader node for term 1.
If more precise and readable logs are required, then there will also be screenshots of the node logs in Docker Desktop. The docker command `docker logs -t <container>` can also be used, but the log format in Docker Desktop is less cluttered.

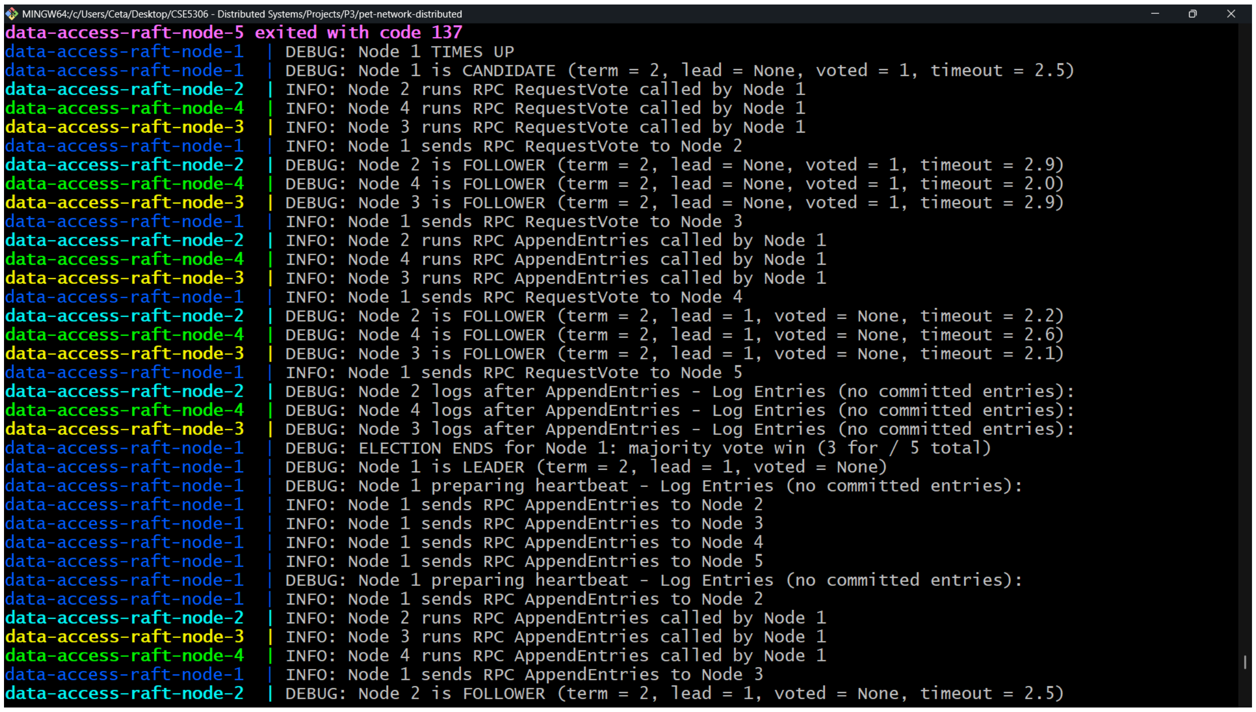
Test Case 1: Current Leader Node Crashes, A New Leader Gets Elected

- Scenario: The current leader node crashes.
- Expected Behavior:
  - One of the follower nodes will become a candidate, increment the term, and send RequestVote RPC to the other nodes in the cluster
  - Other nodes run the RequestVote RPC and vote for the node
  - Candidate node receives the majority and becomes the leader of the cluster
  - Leader node sends AppendEntries RPC to other nodes every second
  - Follower nodes run AppendEntries RPC and recognize the new Leader node for the next term
- Evaluation:
  - Run `docker-compose -f docker-compose-layered-raft.yml up --build` and wait until AppendEntries RPCs are being sent
  - Identify the leader node based on who is sending AppendEntries RPC or observing follower nodes and who they are running the AppendEntries RPC for.
  - Below shows Node 5 as the leader for term 1


MINGW64:/c/Users/Ceta/Desktop/CSE5306 - Distributed Systems/Projects/P3/pet-network-distributed

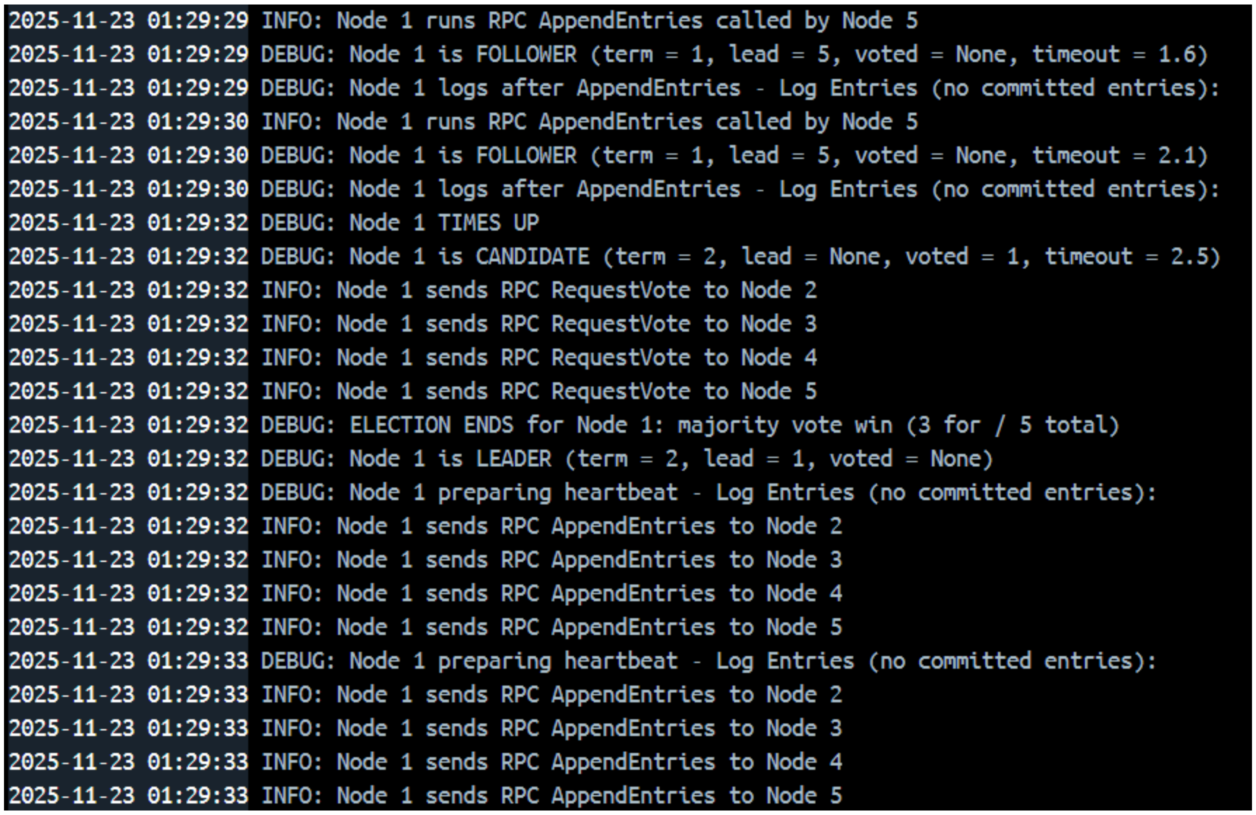
```
data-access-raft-node-5  | INFO: Node 5 sends RPC AppendEntries to Node 1
data-access-raft-node-1  | INFO: Node 1 runs RPC AppendEntries called by Node 5
data-access-raft-node-2  | INFO: Node 2 runs RPC AppendEntries called by Node 5
data-access-raft-node-4  | INFO: Node 4 runs RPC AppendEntries called by Node 5
data-access-raft-node-3  | INFO: Node 3 runs RPC AppendEntries called by Node 5
data-access-raft-node-5  | INFO: Node 5 sends RPC AppendEntries to Node 2
data-access-raft-node-1  | DEBUG: Node 1 is FOLLOWER (term = 1, lead = 5, voted = None, timeout = 2.6)
data-access-raft-node-2  | DEBUG: Node 2 is FOLLOWER (term = 1, lead = 5, voted = None, timeout = 3.0)
data-access-raft-node-4  | DEBUG: Node 4 is FOLLOWER (term = 1, lead = 5, voted = None, timeout = 2.5)
data-access-raft-node-3  | DEBUG: Node 3 is FOLLOWER (term = 1, lead = 5, voted = None, timeout = 2.1)
data-access-raft-node-5  | INFO: Node 5 sends RPC AppendEntries to Node 3
data-access-raft-node-1  | DEBUG: Node 1 logs after AppendEntries - Log Entries (no committed entries):
data-access-raft-node-2  | DEBUG: Node 2 logs after AppendEntries - Log Entries (no committed entries):
data-access-raft-node-4  | DEBUG: Node 4 logs after AppendEntries - Log Entries (no committed entries):
data-access-raft-node-3  | DEBUG: Node 3 logs after AppendEntries - Log Entries (no committed entries):
data-access-raft-node-5  | INFO: Node 5 sends RPC AppendEntries to Node 4
data-access-raft-node-5  | DEBUG: Node 5 preparing heartbeat - Log Entries (no committed entries):
data-access-raft-node-5  | INFO: Node 5 sends RPC AppendEntries to Node 1
data-access-raft-node-1  | INFO: Node 1 runs RPC AppendEntries called by Node 5
data-access-raft-node-4  | INFO: Node 4 runs RPC AppendEntries called by Node 5
data-access-raft-node-2  | INFO: Node 2 runs RPC AppendEntries called by Node 5
data-access-raft-node-3  | INFO: Node 3 runs RPC AppendEntries called by Node 5
data-access-raft-node-5  | INFO: Node 5 sends RPC AppendEntries to Node 2
data-access-raft-node-1  | DEBUG: Node 1 is FOLLOWER (term = 1, lead = 5, voted = None, timeout = 2.1)
data-access-raft-node-4  | DEBUG: Node 4 is FOLLOWER (term = 1, lead = 5, voted = None, timeout = 2.4)
data-access-raft-node-2  | DEBUG: Node 2 is FOLLOWER (term = 1, lead = 5, voted = None, timeout = 2.2)
data-access-raft-node-3  | DEBUG: Node 3 is FOLLOWER (term = 1, lead = 5, voted = None, timeout = 1.8)
data-access-raft-node-5  | INFO: Node 5 sends RPC AppendEntries to Node 3
data-access-raft-node-1  | DEBUG: Node 1 logs after AppendEntries - Log Entries (no committed entries):
data-access-raft-node-4  | DEBUG: Node 4 logs after AppendEntries - Log Entries (no committed entries):
data-access-raft-node-2  | DEBUG: Node 2 logs after AppendEntries - Log Entries (no committed entries):
data-access-raft-node-3  | DEBUG: Node 3 logs after AppendEntries - Log Entries (no committed entries):
data-access-raft-node-5  | INFO: Node 5 sends RPC AppendEntries to Node 4

v View in Docker Desktop   o View Config   w Enable Watch
```

- Stop the container running the corresponding leader node

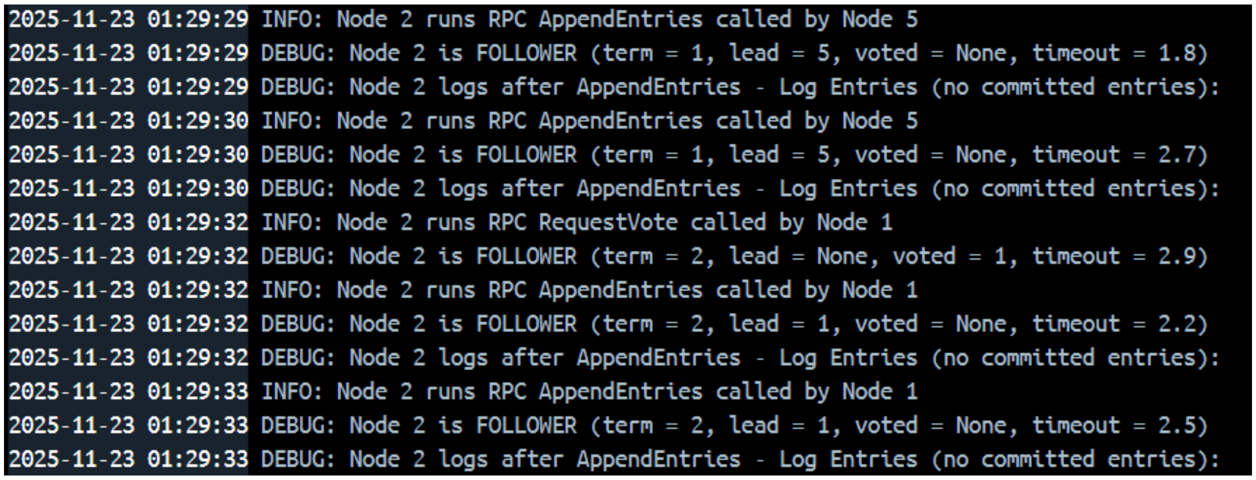

MINGW64:/c/Users/Ceta/Desktop/CSE5306 - Distributed Systems/Projects/P3/pet-network-distributed

```
data-access-raft-node-5 exited with code 137
data-access-raft-node-1  | DEBUG: Node 1 TIMES UP
data-access-raft-node-1  | DEBUG: Node 1 is CANDIDATE (term = 2, lead = None, voted = 1, timeout = 2.5)
data-access-raft-node-2  | INFO: Node 2 runs RPC RequestVote called by Node 1
data-access-raft-node-4  | INFO: Node 4 runs RPC RequestVote called by Node 1
data-access-raft-node-3  | INFO: Node 3 runs RPC RequestVote called by Node 1
data-access-raft-node-1  | INFO: Node 1 sends RPC RequestVote to Node 2
data-access-raft-node-2  | DEBUG: Node 2 is FOLLOWER (term = 2, lead = None, voted = 1, timeout = 2.9)
data-access-raft-node-4  | DEBUG: Node 4 is FOLLOWER (term = 2, lead = None, voted = 1, timeout = 2.0)
data-access-raft-node-3  | DEBUG: Node 3 is FOLLOWER (term = 2, lead = None, voted = 1, timeout = 2.9)
data-access-raft-node-1  | INFO: Node 1 sends RPC RequestVote to Node 3
data-access-raft-node-2  | INFO: Node 2 runs RPC AppendEntries called by Node 1
data-access-raft-node-4  | INFO: Node 4 runs RPC AppendEntries called by Node 1
data-access-raft-node-3  | INFO: Node 3 runs RPC AppendEntries called by Node 1
data-access-raft-node-1  | INFO: Node 1 sends RPC RequestVote to Node 4
data-access-raft-node-2  | DEBUG: Node 2 is FOLLOWER (term = 2, lead = 1, voted = None, timeout = 2.2)
data-access-raft-node-4  | DEBUG: Node 4 is FOLLOWER (term = 2, lead = 1, voted = None, timeout = 2.6)
data-access-raft-node-3  | DEBUG: Node 3 is FOLLOWER (term = 2, lead = 1, voted = None, timeout = 2.1)
data-access-raft-node-1  | INFO: Node 1 sends RPC RequestVote to Node 5
data-access-raft-node-2  | DEBUG: Node 2 logs after AppendEntries - Log Entries (no committed entries):
data-access-raft-node-4  | DEBUG: Node 4 logs after AppendEntries - Log Entries (no committed entries):
data-access-raft-node-3  | DEBUG: Node 3 logs after AppendEntries - Log Entries (no committed entries):
data-access-raft-node-1  | DEBUG: ELECTION ENDS for Node 1: majority vote win (3 for / 5 total)
data-access-raft-node-1  | DEBUG: Node 1 is LEADER (term = 2, lead = 1, voted = None)
data-access-raft-node-1  | DEBUG: Node 1 preparing heartbeat - Log Entries (no committed entries):
data-access-raft-node-1  | INFO: Node 1 sends RPC AppendEntries to Node 2
data-access-raft-node-1  | INFO: Node 1 sends RPC AppendEntries to Node 3
data-access-raft-node-1  | INFO: Node 1 sends RPC AppendEntries to Node 4
data-access-raft-node-1  | INFO: Node 1 sends RPC AppendEntries to Node 5
data-access-raft-node-1  | DEBUG: Node 1 preparing heartbeat - Log Entries (no committed entries):
data-access-raft-node-1  | INFO: Node 1 sends RPC AppendEntries to Node 2
data-access-raft-node-2  | INFO: Node 2 runs RPC AppendEntries called by Node 1
data-access-raft-node-3  | INFO: Node 3 runs RPC AppendEntries called by Node 1
data-access-raft-node-4  | INFO: Node 4 runs RPC AppendEntries called by Node 1
data-access-raft-node-1  | INFO: Node 1 sends RPC AppendEntries to Node 3
data-access-raft-node-2  | DEBUG: Node 2 is FOLLOWER (term = 2, lead = 1, voted = None, timeout = 2.5)
```

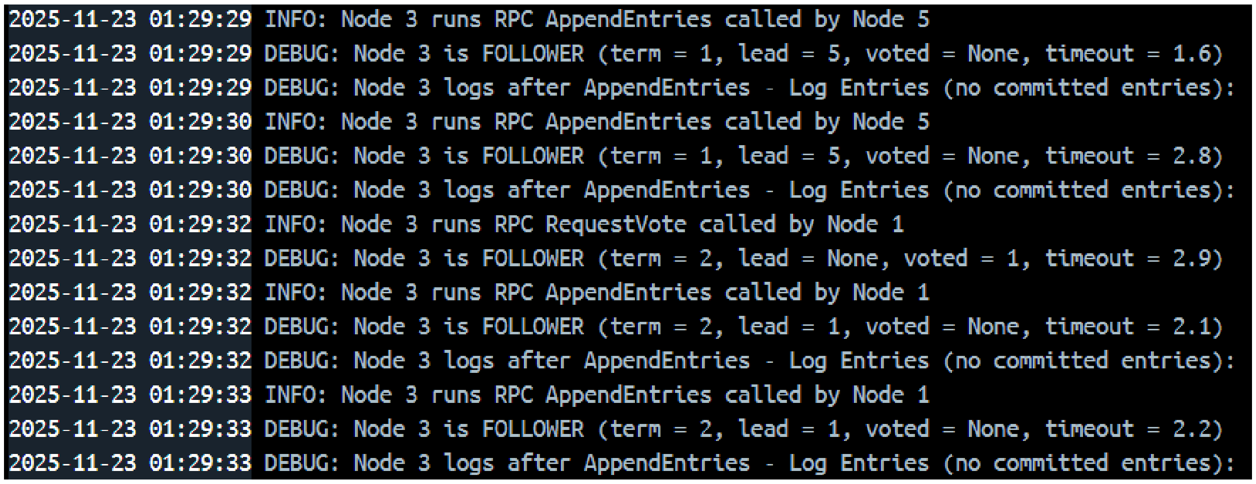
- The screenshot above shows that Node 5 crashes. As a result Node 1's election_timeout expires, becoming a candidate node for term 2, and sends votes out and becomes the new leader. Below shows a snippet of Node 1's logs in Docker Desktop, it also shows that Node 5 stops sending AppendEntries RPCs to followers after 2025-11-23 01:29:30


```
2025-11-23 01:29:29 INFO: Node 1 runs RPC AppendEntries called by Node 5
2025-11-23 01:29:29 DEBUG: Node 1 is FOLLOWER (term = 1, lead = 5, voted = None, timeout = 1.6)
2025-11-23 01:29:29 DEBUG: Node 1 logs after AppendEntries - Log Entries (no committed entries):
2025-11-23 01:29:30 INFO: Node 1 runs RPC AppendEntries called by Node 5
2025-11-23 01:29:30 DEBUG: Node 1 is FOLLOWER (term = 1, lead = 5, voted = None, timeout = 2.1)
2025-11-23 01:29:30 DEBUG: Node 1 logs after AppendEntries - Log Entries (no committed entries):
2025-11-23 01:29:32 DEBUG: Node 1 TIMES UP
2025-11-23 01:29:32 DEBUG: Node 1 is CANDIDATE (term = 2, lead = None, voted = 1, timeout = 2.5)
2025-11-23 01:29:32 INFO: Node 1 sends RPC RequestVote to Node 2
2025-11-23 01:29:32 INFO: Node 1 sends RPC RequestVote to Node 3
2025-11-23 01:29:32 INFO: Node 1 sends RPC RequestVote to Node 4
2025-11-23 01:29:32 INFO: Node 1 sends RPC RequestVote to Node 5
2025-11-23 01:29:32 DEBUG: ELECTION ENDS for Node 1: majority vote win (3 for / 5 total)
2025-11-23 01:29:32 DEBUG: Node 1 is LEADER (term = 2, lead = 1, voted = None)
2025-11-23 01:29:32 DEBUG: Node 1 preparing heartbeat - Log Entries (no committed entries):
2025-11-23 01:29:32 INFO: Node 1 sends RPC AppendEntries to Node 2
2025-11-23 01:29:32 INFO: Node 1 sends RPC AppendEntries to Node 3
2025-11-23 01:29:32 INFO: Node 1 sends RPC AppendEntries to Node 4
2025-11-23 01:29:32 INFO: Node 1 sends RPC AppendEntries to Node 5
2025-11-23 01:29:33 DEBUG: Node 1 preparing heartbeat - Log Entries (no committed entries):
2025-11-23 01:29:33 INFO: Node 1 sends RPC AppendEntries to Node 2
2025-11-23 01:29:33 INFO: Node 1 sends RPC AppendEntries to Node 3
2025-11-23 01:29:33 INFO: Node 1 sends RPC AppendEntries to Node 4
2025-11-23 01:29:33 INFO: Node 1 sends RPC AppendEntries to Node 5
```

- Below shows a snippet of follower nodes (Node 2 and Node 3) from the same timeframe



```
2025-11-23 01:29:29 INFO: Node 2 runs RPC AppendEntries called by Node 5
2025-11-23 01:29:29 DEBUG: Node 2 is FOLLOWER (term = 1, lead = 5, voted = None, timeout = 1.8)
2025-11-23 01:29:29 DEBUG: Node 2 logs after AppendEntries - Log Entries (no committed entries):
2025-11-23 01:29:30 INFO: Node 2 runs RPC AppendEntries called by Node 5
2025-11-23 01:29:30 DEBUG: Node 2 is FOLLOWER (term = 1, lead = 5, voted = None, timeout = 2.7)
2025-11-23 01:29:30 DEBUG: Node 2 logs after AppendEntries - Log Entries (no committed entries):
2025-11-23 01:29:32 INFO: Node 2 runs RPC RequestVote called by Node 1
2025-11-23 01:29:32 DEBUG: Node 2 is FOLLOWER (term = 2, lead = None, voted = 1, timeout = 2.9)
2025-11-23 01:29:32 INFO: Node 2 runs RPC AppendEntries called by Node 1
2025-11-23 01:29:32 DEBUG: Node 2 is FOLLOWER (term = 2, lead = 1, voted = None, timeout = 2.2)
2025-11-23 01:29:32 DEBUG: Node 2 logs after AppendEntries - Log Entries (no committed entries):
2025-11-23 01:29:33 INFO: Node 2 runs RPC AppendEntries called by Node 1
2025-11-23 01:29:33 DEBUG: Node 2 is FOLLOWER (term = 2, lead = 1, voted = None, timeout = 2.5)
2025-11-23 01:29:33 DEBUG: Node 2 logs after AppendEntries - Log Entries (no committed entries):
```



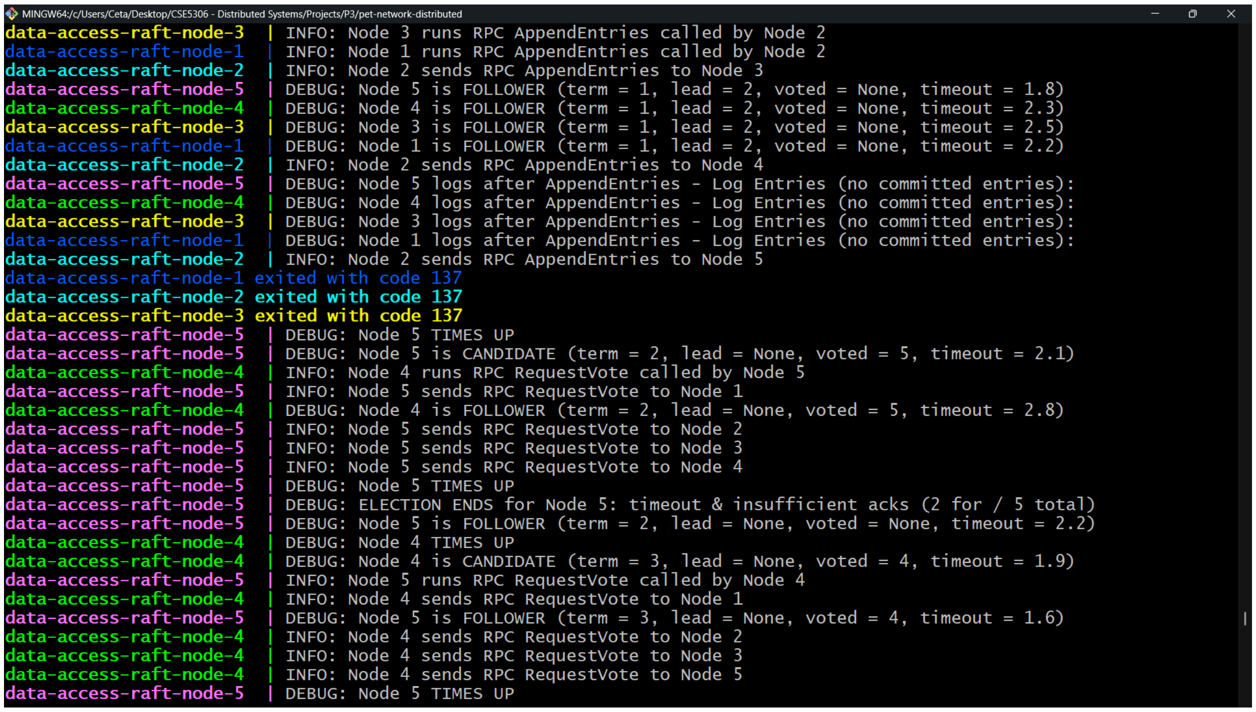
```
2025-11-23 01:29:29 INFO: Node 3 runs RPC AppendEntries called by Node 5
2025-11-23 01:29:29 DEBUG: Node 3 is FOLLOWER (term = 1, lead = 5, voted = None, timeout = 1.6)
2025-11-23 01:29:29 DEBUG: Node 3 logs after AppendEntries - Log Entries (no committed entries):
2025-11-23 01:29:30 INFO: Node 3 runs RPC AppendEntries called by Node 5
2025-11-23 01:29:30 DEBUG: Node 3 is FOLLOWER (term = 1, lead = 5, voted = None, timeout = 2.8)
2025-11-23 01:29:30 DEBUG: Node 3 logs after AppendEntries - Log Entries (no committed entries):
2025-11-23 01:29:32 INFO: Node 3 runs RPC RequestVote called by Node 1
2025-11-23 01:29:32 DEBUG: Node 3 is FOLLOWER (term = 2, lead = None, voted = 1, timeout = 2.9)
2025-11-23 01:29:32 INFO: Node 3 runs RPC AppendEntries called by Node 1
2025-11-23 01:29:32 DEBUG: Node 3 is FOLLOWER (term = 2, lead = 1, voted = None, timeout = 2.1)
2025-11-23 01:29:32 DEBUG: Node 3 logs after AppendEntries - Log Entries (no committed entries):
2025-11-23 01:29:33 INFO: Node 3 runs RPC AppendEntries called by Node 1
2025-11-23 01:29:33 DEBUG: Node 3 is FOLLOWER (term = 2, lead = 1, voted = None, timeout = 2.2)
2025-11-23 01:29:33 DEBUG: Node 3 logs after AppendEntries - Log Entries (no committed entries):
```

    - Node 4's log is mostly the same, to reduce the report length, that screenshot will be omitted (the terminal output above with all of the node's logs show the same log output for node 4 after node 5 crashes)
    - As such, this test case is successful. Node 5 (leader for term 1) crashes. One of the followers election_timeout expires (Node 1), and becomes a candidate for the next term (term 2) and sends RequestVote RPC to others. Other nodes (2, 3, and 4) run the RPC updating their term and Node 1 receives a majority vote becoming the new leader node. Node 1 begins sending AppendEntry RPCs to followers.

Test Case 2: Nodes Do Not Promote From Candidate State to Leader State Without Majority Votes

- Scenario: The leader node and 2 follower nodes crash, leaving only 2 nodes running.
- Expected Behavior:
    - One of the nodes election timeout expires and becomes a candidate for the next term (t+1) and send RequestVote RPC to nodes in the cluster
    - The other node runs that RequestVote RPC and becomes a follower who votes for the original node

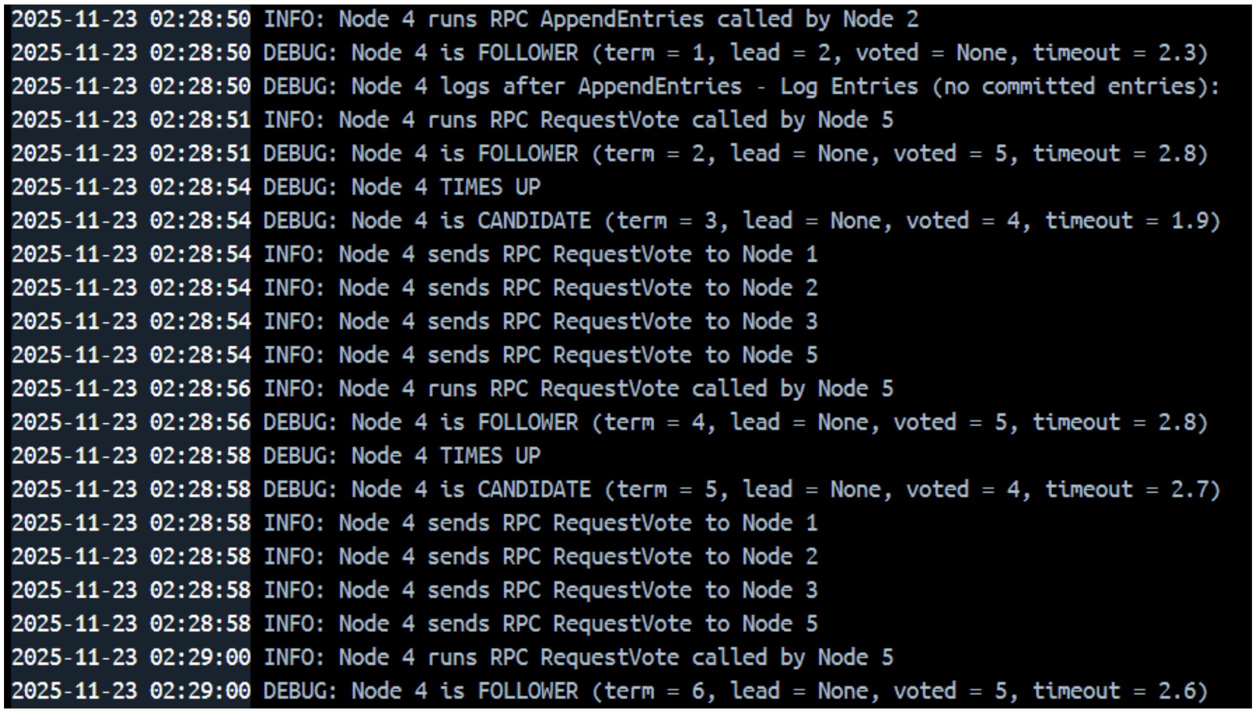
        - The original node's election timeout expires and reverts back to a follower node OR the other node's election timeout expires becomes a candidate for the next term after (t+2) and sends RequestVote RPC to nodes in the cluster (causing the original node to become a follower)
        - The original node runs that RequestVote RPC and becomes a follower who votes for the other node
        - The other node's election timeout expires and reverts back to a follower node OR the original node's election timeout and becomes a candidate for the next term (t+3) and sends RequestVote RPC…
        - This continues until enough nodes recover to where a majority can be decided
- Evaluation:
    - Run `docker-compose -f docker-compose-layered-raft.yml up --build` and wait until AppendEntries RPCs are being sent
    - Identify the leader node and stop its container and two other nodes containers. In the terminal output below, Node 2 is the leader, so the containers for Node 2, Node 1 and Node 3 were stopped.



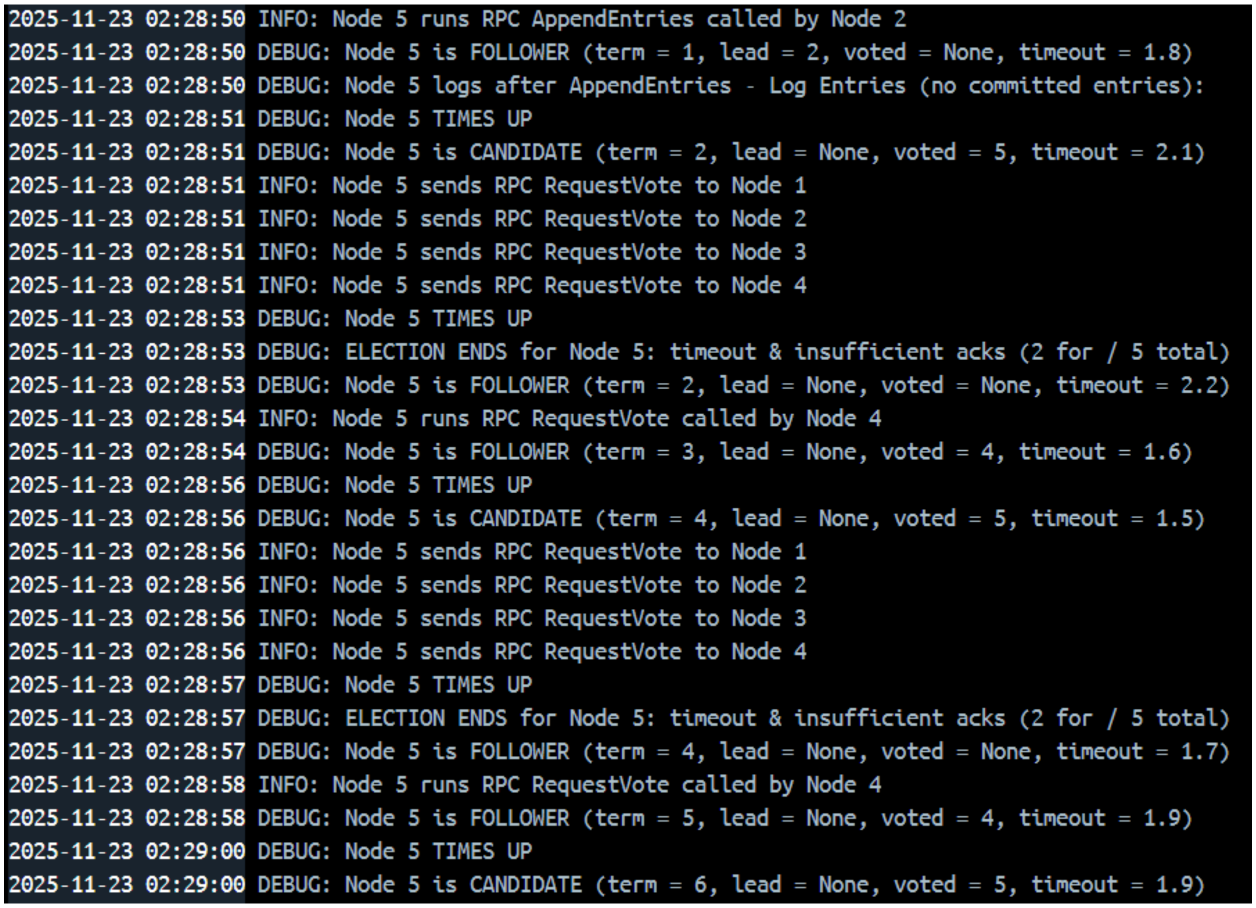
MINGW64:/c/Users/Ceta/Desktop/CSE5306 - Distributed Systems/Projects/P3/pet-network-distributed

```
data-access-raft-node-3  | INFO: Node 3 runs RPC AppendEntries called by Node 2
data-access-raft-node-1  | INFO: Node 1 runs RPC AppendEntries called by Node 2
data-access-raft-node-2  | INFO: Node 2 sends RPC AppendEntries to Node 3
data-access-raft-node-5  | DEBUG: Node 5 is FOLLOWER (term = 1, lead = 2, voted = None, timeout = 1.8)
data-access-raft-node-4  | DEBUG: Node 4 is FOLLOWER (term = 1, lead = 2, voted = None, timeout = 2.3)
data-access-raft-node-3  | DEBUG: Node 3 is FOLLOWER (term = 1, lead = 2, voted = None, timeout = 2.5)
data-access-raft-node-1  | DEBUG: Node 1 is FOLLOWER (term = 1, lead = 2, voted = None, timeout = 2.2)
data-access-raft-node-2  | INFO: Node 2 sends RPC AppendEntries to Node 4
data-access-raft-node-5  | DEBUG: Node 5 logs after AppendEntries - Log Entries (no committed entries):
data-access-raft-node-4  | DEBUG: Node 4 logs after AppendEntries - Log Entries (no committed entries):
data-access-raft-node-3  | DEBUG: Node 3 logs after AppendEntries - Log Entries (no committed entries):
data-access-raft-node-1  | DEBUG: Node 1 logs after AppendEntries - Log Entries (no committed entries):
data-access-raft-node-2  | INFO: Node 2 sends RPC AppendEntries to Node 5
data-access-raft-node-1 exited with code 137
data-access-raft-node-2 exited with code 137
data-access-raft-node-3 exited with code 137
data-access-raft-node-5  | DEBUG: Node 5 TIMES UP
data-access-raft-node-5  | DEBUG: Node 5 is CANDIDATE (term = 2, lead = None, voted = 5, timeout = 2.1)
data-access-raft-node-4  | INFO: Node 4 runs RPC RequestVote called by Node 5
data-access-raft-node-5  | INFO: Node 5 sends RPC RequestVote to Node 1
data-access-raft-node-4  | DEBUG: Node 4 is FOLLOWER (term = 2, lead = None, voted = 5, timeout = 2.8)
data-access-raft-node-5  | INFO: Node 5 sends RPC RequestVote to Node 2
data-access-raft-node-5  | INFO: Node 5 sends RPC RequestVote to Node 3
data-access-raft-node-5  | INFO: Node 5 sends RPC RequestVote to Node 4
data-access-raft-node-5  | DEBUG: Node 5 TIMES UP
data-access-raft-node-5  | DEBUG: ELECTION ENDS for Node 5: timeout & insufficient acks (2 for / 5 total)
data-access-raft-node-5  | DEBUG: Node 5 is FOLLOWER (term = 2, lead = None, voted = None, timeout = 2.2)
data-access-raft-node-4  | DEBUG: Node 4 TIMES UP
data-access-raft-node-4  | DEBUG: Node 4 is CANDIDATE (term = 3, lead = None, voted = 4, timeout = 1.9)
data-access-raft-node-5  | INFO: Node 5 runs RPC RequestVote called by Node 4
data-access-raft-node-4  | INFO: Node 4 sends RPC RequestVote to Node 1
data-access-raft-node-5  | DEBUG: Node 5 is FOLLOWER (term = 3, lead = None, voted = 4, timeout = 1.6)
data-access-raft-node-4  | INFO: Node 4 sends RPC RequestVote to Node 2
data-access-raft-node-4  | INFO: Node 4 sends RPC RequestVote to Node 3
data-access-raft-node-4  | INFO: Node 4 sends RPC RequestVote to Node 5
data-access-raft-node-5  | DEBUG: Node 5 TIMES UP
```

    - This leaves Node 4 and Node 5 as running, below are the logs for them around the time the other nodes crash



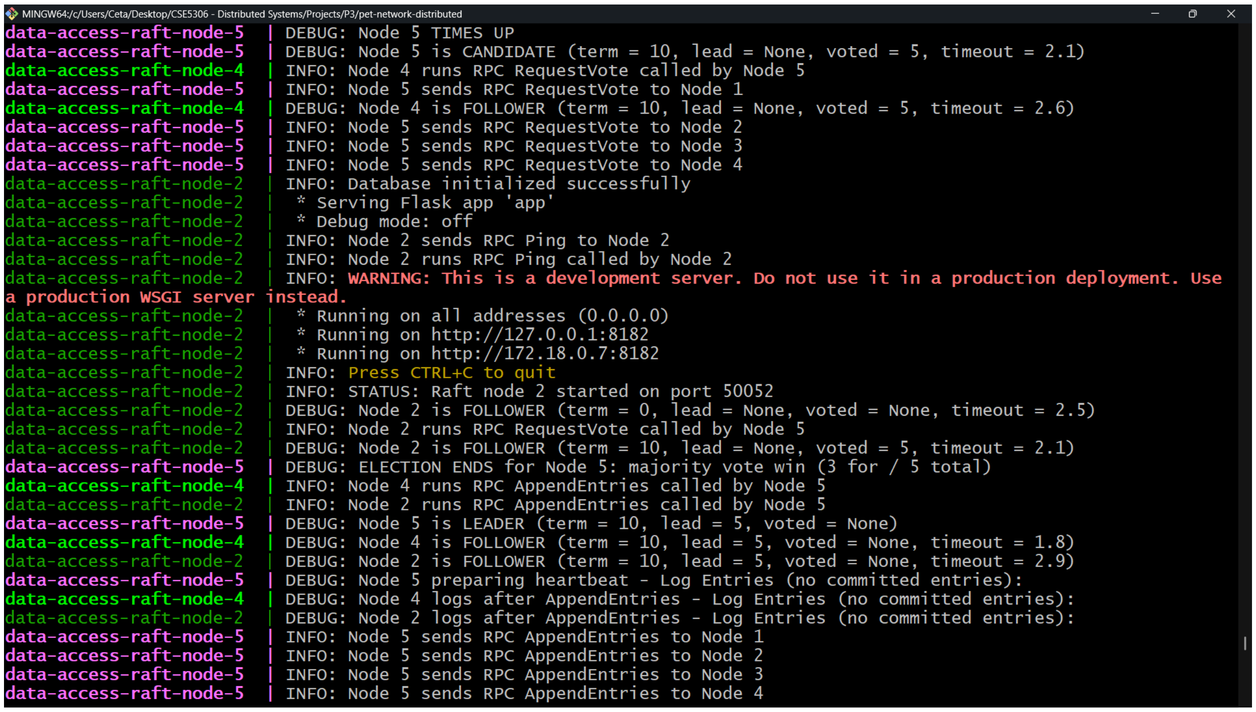
```
2025-11-23 02:28:50 INFO: Node 4 runs RPC AppendEntries called by Node 2
2025-11-23 02:28:50 DEBUG: Node 4 is FOLLOWER (term = 1, lead = 2, voted = None, timeout = 2.3)
2025-11-23 02:28:50 DEBUG: Node 4 logs after AppendEntries - Log Entries (no committed entries):
2025-11-23 02:28:51 INFO: Node 4 runs RPC RequestVote called by Node 5
2025-11-23 02:28:51 DEBUG: Node 4 is FOLLOWER (term = 2, lead = None, voted = 5, timeout = 2.8)
2025-11-23 02:28:54 DEBUG: Node 4 TIMES UP
2025-11-23 02:28:54 DEBUG: Node 4 is CANDIDATE (term = 3, lead = None, voted = 4, timeout = 1.9)
2025-11-23 02:28:54 INFO: Node 4 sends RPC RequestVote to Node 1
2025-11-23 02:28:54 INFO: Node 4 sends RPC RequestVote to Node 2
2025-11-23 02:28:54 INFO: Node 4 sends RPC RequestVote to Node 3
2025-11-23 02:28:54 INFO: Node 4 sends RPC RequestVote to Node 5
2025-11-23 02:28:56 INFO: Node 4 runs RPC RequestVote called by Node 5
2025-11-23 02:28:56 DEBUG: Node 4 is FOLLOWER (term = 4, lead = None, voted = 5, timeout = 2.8)
2025-11-23 02:28:58 DEBUG: Node 4 TIMES UP
2025-11-23 02:28:58 DEBUG: Node 4 is CANDIDATE (term = 5, lead = None, voted = 4, timeout = 2.7)
2025-11-23 02:28:58 INFO: Node 4 sends RPC RequestVote to Node 1
2025-11-23 02:28:58 INFO: Node 4 sends RPC RequestVote to Node 2
2025-11-23 02:28:58 INFO: Node 4 sends RPC RequestVote to Node 3
2025-11-23 02:28:58 INFO: Node 4 sends RPC RequestVote to Node 5
2025-11-23 02:29:00 INFO: Node 4 runs RPC RequestVote called by Node 5
2025-11-23 02:29:00 DEBUG: Node 4 is FOLLOWER (term = 6, lead = None, voted = 5, timeout = 2.6)
```



```
2025-11-23 02:28:50 INFO: Node 5 runs RPC AppendEntries called by Node 2
2025-11-23 02:28:50 DEBUG: Node 5 is FOLLOWER (term = 1, lead = 2, voted = None, timeout = 1.8)
2025-11-23 02:28:50 DEBUG: Node 5 logs after AppendEntries - Log Entries (no committed entries):
2025-11-23 02:28:51 DEBUG: Node 5 TIMES UP
2025-11-23 02:28:51 DEBUG: Node 5 is CANDIDATE (term = 2, lead = None, voted = 5, timeout = 2.1)
2025-11-23 02:28:51 INFO: Node 5 sends RPC RequestVote to Node 1
2025-11-23 02:28:51 INFO: Node 5 sends RPC RequestVote to Node 2
2025-11-23 02:28:51 INFO: Node 5 sends RPC RequestVote to Node 3
2025-11-23 02:28:51 INFO: Node 5 sends RPC RequestVote to Node 4
2025-11-23 02:28:53 DEBUG: Node 5 TIMES UP
2025-11-23 02:28:53 DEBUG: ELECTION ENDS for Node 5: timeout & insufficient acks (2 for / 5 total)
2025-11-23 02:28:53 DEBUG: Node 5 is FOLLOWER (term = 2, lead = None, voted = None, timeout = 2.2)
2025-11-23 02:28:54 INFO: Node 5 runs RPC RequestVote called by Node 4
2025-11-23 02:28:54 DEBUG: Node 5 is FOLLOWER (term = 3, lead = None, voted = 4, timeout = 1.6)
2025-11-23 02:28:56 DEBUG: Node 5 TIMES UP
2025-11-23 02:28:56 DEBUG: Node 5 is CANDIDATE (term = 4, lead = None, voted = 5, timeout = 1.5)
2025-11-23 02:28:56 INFO: Node 5 sends RPC RequestVote to Node 1
2025-11-23 02:28:56 INFO: Node 5 sends RPC RequestVote to Node 2
2025-11-23 02:28:56 INFO: Node 5 sends RPC RequestVote to Node 3
2025-11-23 02:28:56 INFO: Node 5 sends RPC RequestVote to Node 4
2025-11-23 02:28:57 DEBUG: Node 5 TIMES UP
2025-11-23 02:28:57 DEBUG: ELECTION ENDS for Node 5: timeout & insufficient acks (2 for / 5 total)
2025-11-23 02:28:57 DEBUG: Node 5 is FOLLOWER (term = 4, lead = None, voted = None, timeout = 1.7)
2025-11-23 02:28:58 INFO: Node 5 runs RPC RequestVote called by Node 4
2025-11-23 02:28:58 DEBUG: Node 5 is FOLLOWER (term = 5, lead = None, voted = 4, timeout = 1.9)
2025-11-23 02:29:00 DEBUG: Node 5 TIMES UP
2025-11-23 02:29:00 DEBUG: Node 5 is CANDIDATE (term = 6, lead = None, voted = 5, timeout = 1.9)
```

- From both of the Node's logs, they are flip-floping between follower and candidate states. Node 4 doesn't receive an ELECTION ENDS due to timeout because what ended up happening was that Node 5 sends the RequestVote RPC to Node 4 before its voting period was over. As such, Node 4 saw Node 5 with a higher term and became a follower that voted for Node 5. Node 5's voting period on the other hand timed out, but was unable to become a leader due to insufficient votes/acks

    ○ The following screenshot shows the clusters behavior once Node 2 recovers



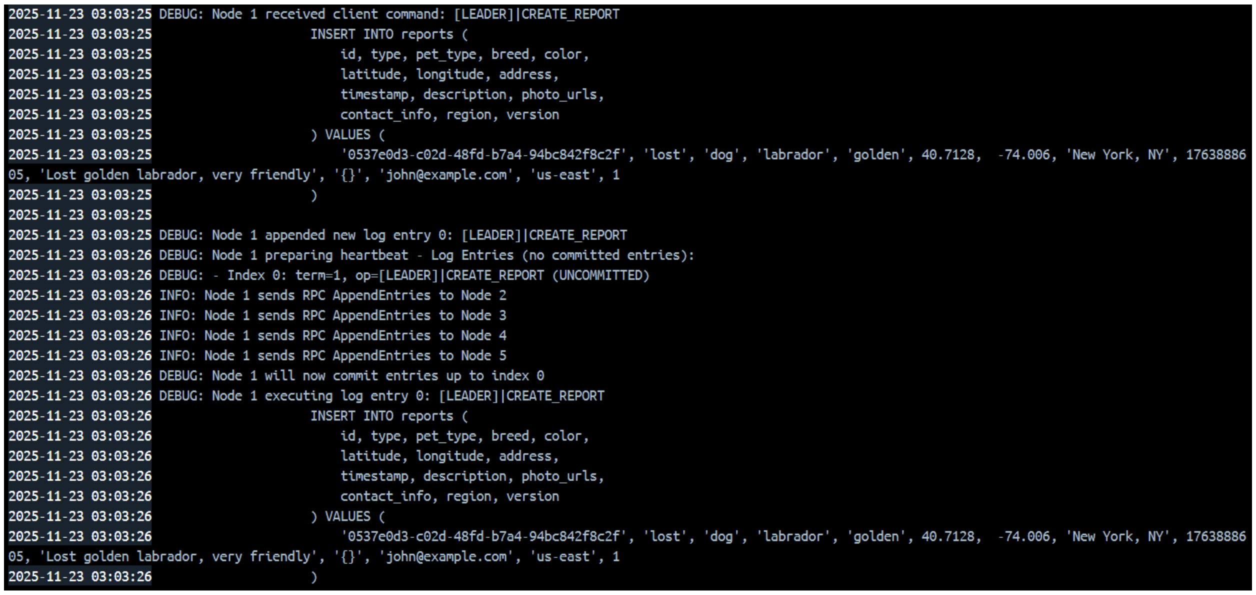
```
MINGW64:/c/Users/Ceta/Desktop/CSE5306 - Distributed Systems/Projects/P3/pet-network-distributed
data-access-raft-node-5  | DEBUG: Node 5 TIMES UP
data-access-raft-node-5  | DEBUG: Node 5 is CANDIDATE (term = 10, lead = None, voted = 5, timeout = 2.1)
data-access-raft-node-4  | INFO: Node 4 runs RPC RequestVote called by Node 5
data-access-raft-node-5  | INFO: Node 5 sends RPC RequestVote to Node 1
data-access-raft-node-4  | DEBUG: Node 4 is FOLLOWER (term = 10, lead = None, voted = 5, timeout = 2.6)
data-access-raft-node-5  | INFO: Node 5 sends RPC RequestVote to Node 2
data-access-raft-node-5  | INFO: Node 5 sends RPC RequestVote to Node 3
data-access-raft-node-5  | INFO: Node 5 sends RPC RequestVote to Node 4
data-access-raft-node-2  | INFO: Database initialized successfully
data-access-raft-node-2  |  * Serving Flask app 'app'
data-access-raft-node-2  |  * Debug mode: off
data-access-raft-node-2  | INFO: Node 2 sends RPC Ping to Node 2
data-access-raft-node-2  | INFO: Node 2 runs RPC Ping called by Node 2
data-access-raft-node-2  | INFO: WARNING: This is a development server. Do not use it in a production deployment. Use
a production WSGI server instead.
data-access-raft-node-2  |  * Running on all addresses (0.0.0.0)
data-access-raft-node-2  |  * Running on http://127.0.0.1:8182
data-access-raft-node-2  |  * Running on http://172.18.0.7:8182
data-access-raft-node-2  | INFO: Press CTRL+C to quit
data-access-raft-node-2  | INFO: STATUS: Raft node 2 started on port 50052
data-access-raft-node-2  | DEBUG: Node 2 is FOLLOWER (term = 0, lead = None, voted = None, timeout = 2.5)
data-access-raft-node-2  | INFO: Node 2 runs RPC RequestVote called by Node 5
data-access-raft-node-2  | DEBUG: Node 2 is FOLLOWER (term = 10, lead = None, voted = 5, timeout = 2.1)
data-access-raft-node-5  | DEBUG: ELECTION ENDS for Node 5: majority vote win (3 for / 5 total)
data-access-raft-node-4  | INFO: Node 4 runs RPC AppendEntries called by Node 5
data-access-raft-node-2  | INFO: Node 2 runs RPC AppendEntries called by Node 5
data-access-raft-node-5  | DEBUG: Node 5 is LEADER (term = 10, lead = 5, voted = None)
data-access-raft-node-4  | DEBUG: Node 4 is FOLLOWER (term = 10, lead = 5, voted = None, timeout = 1.8)
data-access-raft-node-2  | DEBUG: Node 2 is FOLLOWER (term = 10, lead = 5, voted = None, timeout = 2.9)
data-access-raft-node-5  | DEBUG: Node 5 preparing heartbeat - Log Entries (no committed entries):
data-access-raft-node-4  | DEBUG: Node 4 logs after AppendEntries - Log Entries (no committed entries):
data-access-raft-node-2  | DEBUG: Node 2 logs after AppendEntries - Log Entries (no committed entries):
data-access-raft-node-5  | INFO: Node 5 sends RPC AppendEntries to Node 1
data-access-raft-node-5  | INFO: Node 5 sends RPC AppendEntries to Node 2
data-access-raft-node-5  | INFO: Node 5 sends RPC AppendEntries to Node 3
data-access-raft-node-5  | INFO: Node 5 sends RPC AppendEntries to Node 4
```

    ○ Here we can see Node 2 boots back up. Node 5 recently became a candidate and sends its RequestVote RPCs. Both Node 4 and Node 2 run the RequestVote RPC from Node 5 and grant their votes. Now Node 5 has a majority and can become the leader and start sending heartbeats.
    ○ As such, this test case is successful. When only two nodes remained in the cluster (Node 4 & Node 5), no leader could be elected as there were not enough nodes for consensus. It wasn't until another Node (Node 2) recovered and was able to process RPCs again. The three Nodes together make a majority out of five total nodes, and thus a new leader was able to be elected.

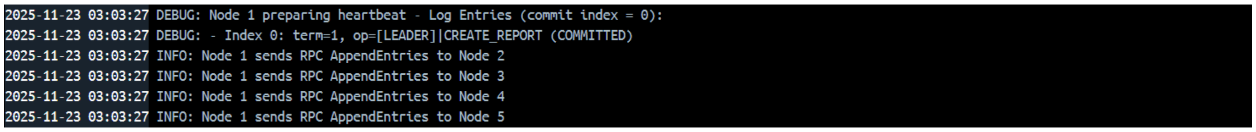
Test Case 3: Logs and Commit Indices Are Replicated In the Cluster

- Scenario: A client submits a POST request to create a report via the API gateway and the leader receives it.
- Expected Behavior:
    - Leader node adds the request to the log
    - Leader node sends heartbeat with the log to followers via AppendEntries RPC
    - Follower nodes run AppendEntries RPC copying the log and responding with an ACK
    - Leader node receives majority ACKs and commits and executes the client request

    - Leader node sends heartbeat with the log and commit index to followers via AppendEntries RPC
    - Follower nodes run AppendEntries RPC copying the log and updating commit index
- Evaluation:
    - Run `docker-compose -f docker-compose-layered-raft.yml up --build` and wait until AppendEntries RPCs are being sent
    - In a separate terminal, execute the following command:
```
curl -X POST http://localhost:8080/api/reports \
  -H "Content-Type: application/json" \
  -d '{
    "type": "lost",
    "pet_type": "dog",
    "breed": "labrador",
    "color": "golden",
    "location": {
      "latitude": 40.7128,
      "longitude": -74.0060,
      "address": "New York, NY"
    },
    "description": "Lost golden labrador, very friendly",
    "contact_info": "john@example.com"
  }'
```
    - Observe the node's logs and check to see that the leader received the client command.
    - Once the leader receives the client's command, it will append to its log a new entry. On the next heartbeat (AppendEntries RPCs), the leader's log will be sent over along with the commit index. The commit index should be before the index of the newly entered log to signify that it is uncommitted.
    - Followers will run the AppendEntries RPC and copy the log sent. As such, the Follower's log will also contain that same entry by the leader. Followers will then send an ACK back as a response
    - When the leader receives a majority ACK, the leader will then commit and execute the log entry. The commit index will be increased and on the next heartbeat with follows, the followers will proceed to commit entries up to the new commit index shared by the leader.
    - Below are the logs of different Nodes with Node 1 being the leader and receiving the client request within the same time interval
    - Node 1's log below

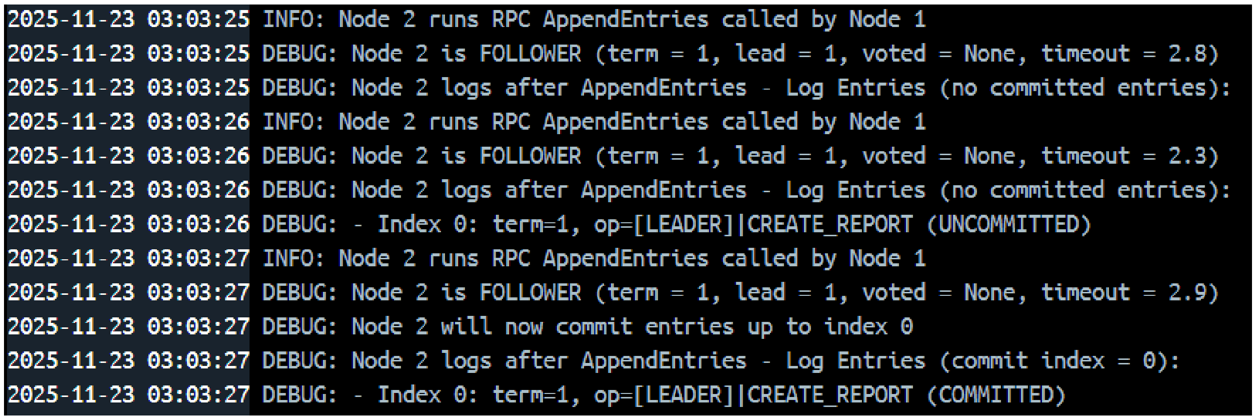


```
2025-11-23 03:03:25 DEBUG: Node 1 received client command: [LEADER]|CREATE_REPORT
2025-11-23 03:03:25                     INSERT INTO reports (
2025-11-23 03:03:25                         id, type, pet_type, breed, color,
2025-11-23 03:03:25                         latitude, longitude, address,
2025-11-23 03:03:25                         timestamp, description, photo_urls,
2025-11-23 03:03:25                         contact_info, region, version
2025-11-23 03:03:25                     ) VALUES (
2025-11-23 03:03:25                         '0537e0d3-c02d-48fd-b7a4-94bc842f8c2f', 'lost', 'dog', 'labrador', 'golden', 40.7128,  -74.006, 'New York, NY', 17638886
05, 'Lost golden labrador, very friendly', '{}', 'john@example.com', 'us-east', 1
2025-11-23 03:03:25                     )
2025-11-23 03:03:25
2025-11-23 03:03:25 DEBUG: Node 1 appended new log entry 0: [LEADER]|CREATE_REPORT
2025-11-23 03:03:26 DEBUG: Node 1 preparing heartbeat - Log Entries (no committed entries):
2025-11-23 03:03:26 DEBUG: - Index 0: term=1, op=[LEADER]|CREATE_REPORT (UNCOMMITTED)
2025-11-23 03:03:26 INFO: Node 1 sends RPC AppendEntries to Node 2
2025-11-23 03:03:26 INFO: Node 1 sends RPC AppendEntries to Node 3
2025-11-23 03:03:26 INFO: Node 1 sends RPC AppendEntries to Node 4
2025-11-23 03:03:26 INFO: Node 1 sends RPC AppendEntries to Node 5
2025-11-23 03:03:26 DEBUG: Node 1 will now commit entries up to index 0
2025-11-23 03:03:26 DEBUG: Node 1 executing log entry 0: [LEADER]|CREATE_REPORT
2025-11-23 03:03:26                     INSERT INTO reports (
2025-11-23 03:03:26                         id, type, pet_type, breed, color,
2025-11-23 03:03:26                         latitude, longitude, address,
2025-11-23 03:03:26                         timestamp, description, photo_urls,
2025-11-23 03:03:26                         contact_info, region, version
2025-11-23 03:03:26                     ) VALUES (
2025-11-23 03:03:26                         '0537e0d3-c02d-48fd-b7a4-94bc842f8c2f', 'lost', 'dog', 'labrador', 'golden', 40.7128,  -74.006, 'New York, NY', 17638886
05, 'Lost golden labrador, very friendly', '{}', 'john@example.com', 'us-east', 1
2025-11-23 03:03:26                     )
```



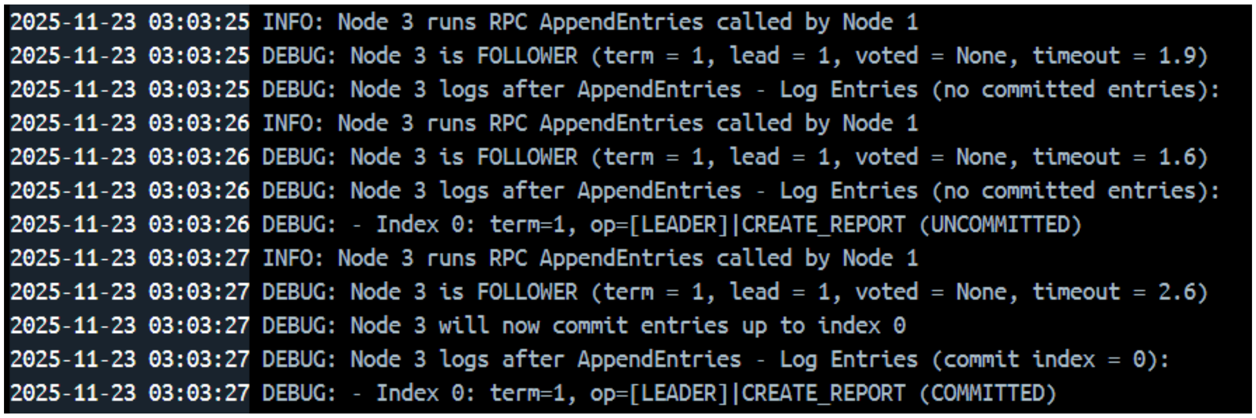
```
2025-11-23 03:03:27 DEBUG: Node 1 preparing heartbeat - Log Entries (commit index = 0):
2025-11-23 03:03:27 DEBUG: - Index 0: term=1, op=[LEADER]|CREATE_REPORT (COMMITTED)
2025-11-23 03:03:27 INFO: Node 1 sends RPC AppendEntries to Node 2
2025-11-23 03:03:27 INFO: Node 1 sends RPC AppendEntries to Node 3
2025-11-23 03:03:27 INFO: Node 1 sends RPC AppendEntries to Node 4
2025-11-23 03:03:27 INFO: Node 1 sends RPC AppendEntries to Node 5
```

- Node 2’s log below



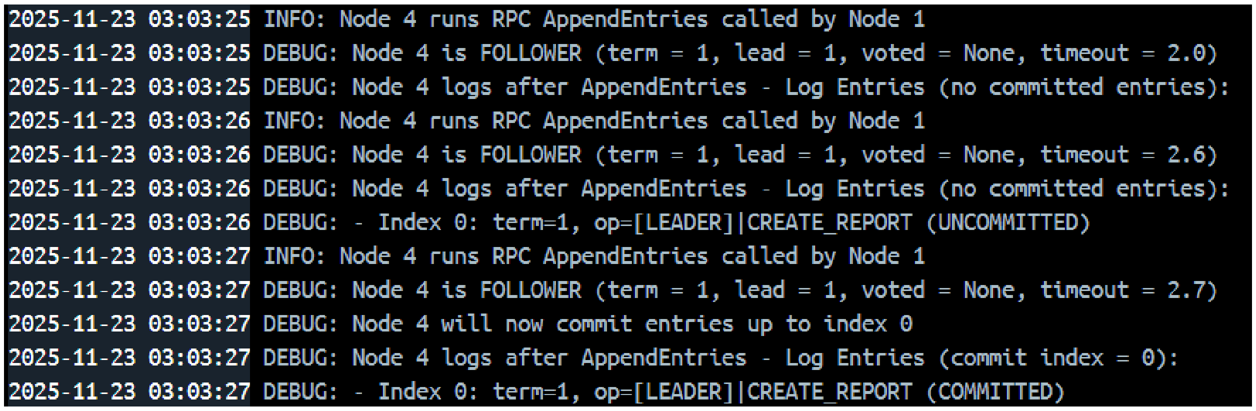
```
2025-11-23 03:03:25 INFO: Node 2 runs RPC AppendEntries called by Node 1
2025-11-23 03:03:25 DEBUG: Node 2 is FOLLOWER (term = 1, lead = 1, voted = None, timeout = 2.8)
2025-11-23 03:03:25 DEBUG: Node 2 logs after AppendEntries - Log Entries (no committed entries):
2025-11-23 03:03:26 INFO: Node 2 runs RPC AppendEntries called by Node 1
2025-11-23 03:03:26 DEBUG: Node 2 is FOLLOWER (term = 1, lead = 1, voted = None, timeout = 2.3)
2025-11-23 03:03:26 DEBUG: Node 2 logs after AppendEntries - Log Entries (no committed entries):
2025-11-23 03:03:26 DEBUG: - Index 0: term=1, op=[LEADER]|CREATE_REPORT (UNCOMMITTED)
2025-11-23 03:03:27 INFO: Node 2 runs RPC AppendEntries called by Node 1
2025-11-23 03:03:27 DEBUG: Node 2 is FOLLOWER (term = 1, lead = 1, voted = None, timeout = 2.9)
2025-11-23 03:03:27 DEBUG: Node 2 will now commit entries up to index 0
2025-11-23 03:03:27 DEBUG: Node 2 logs after AppendEntries - Log Entries (commit index = 0):
2025-11-23 03:03:27 DEBUG: - Index 0: term=1, op=[LEADER]|CREATE_REPORT (COMMITTED)
```

- Node 3’s log below



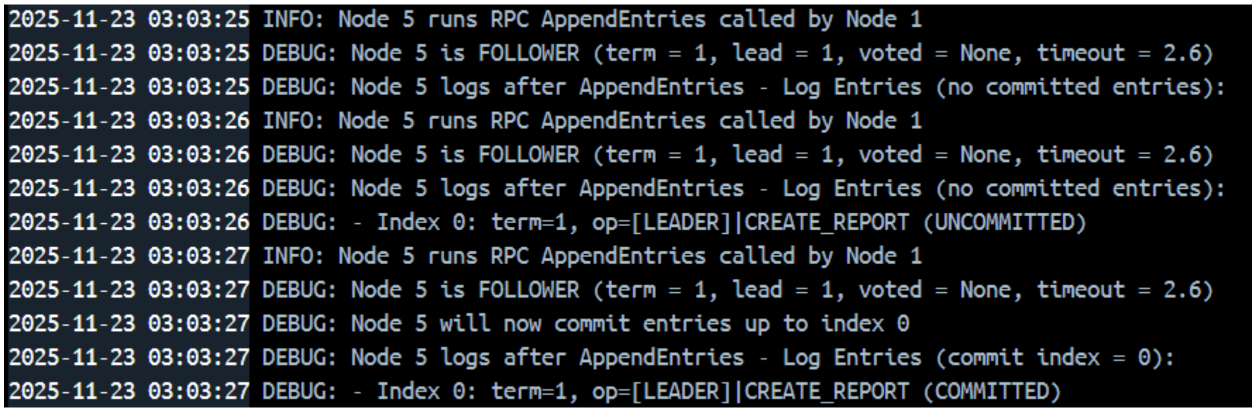
```
2025-11-23 03:03:25 INFO: Node 3 runs RPC AppendEntries called by Node 1
2025-11-23 03:03:25 DEBUG: Node 3 is FOLLOWER (term = 1, lead = 1, voted = None, timeout = 1.9)
2025-11-23 03:03:25 DEBUG: Node 3 logs after AppendEntries - Log Entries (no committed entries):
2025-11-23 03:03:26 INFO: Node 3 runs RPC AppendEntries called by Node 1
2025-11-23 03:03:26 DEBUG: Node 3 is FOLLOWER (term = 1, lead = 1, voted = None, timeout = 1.6)
2025-11-23 03:03:26 DEBUG: Node 3 logs after AppendEntries - Log Entries (no committed entries):
2025-11-23 03:03:26 DEBUG: - Index 0: term=1, op=[LEADER]|CREATE_REPORT (UNCOMMITTED)
2025-11-23 03:03:27 INFO: Node 3 runs RPC AppendEntries called by Node 1
2025-11-23 03:03:27 DEBUG: Node 3 is FOLLOWER (term = 1, lead = 1, voted = None, timeout = 2.6)
2025-11-23 03:03:27 DEBUG: Node 3 will now commit entries up to index 0
2025-11-23 03:03:27 DEBUG: Node 3 logs after AppendEntries - Log Entries (commit index = 0):
2025-11-23 03:03:27 DEBUG: - Index 0: term=1, op=[LEADER]|CREATE_REPORT (COMMITTED)
```

- Node 4’s log below



```
2025-11-23 03:03:25 INFO: Node 4 runs RPC AppendEntries called by Node 1
2025-11-23 03:03:25 DEBUG: Node 4 is FOLLOWER (term = 1, lead = 1, voted = None, timeout = 2.0)
2025-11-23 03:03:25 DEBUG: Node 4 logs after AppendEntries - Log Entries (no committed entries):
2025-11-23 03:03:26 INFO: Node 4 runs RPC AppendEntries called by Node 1
2025-11-23 03:03:26 DEBUG: Node 4 is FOLLOWER (term = 1, lead = 1, voted = None, timeout = 2.6)
2025-11-23 03:03:26 DEBUG: Node 4 logs after AppendEntries - Log Entries (no committed entries):
2025-11-23 03:03:26 DEBUG: - Index 0: term=1, op=[LEADER]|CREATE_REPORT (UNCOMMITTED)
2025-11-23 03:03:27 INFO: Node 4 runs RPC AppendEntries called by Node 1
2025-11-23 03:03:27 DEBUG: Node 4 is FOLLOWER (term = 1, lead = 1, voted = None, timeout = 2.7)
2025-11-23 03:03:27 DEBUG: Node 4 will now commit entries up to index 0
2025-11-23 03:03:27 DEBUG: Node 4 logs after AppendEntries - Log Entries (commit index = 0):
2025-11-23 03:03:27 DEBUG: - Index 0: term=1, op=[LEADER]|CREATE_REPORT (COMMITTED)
```

- ○ Node 5's log below



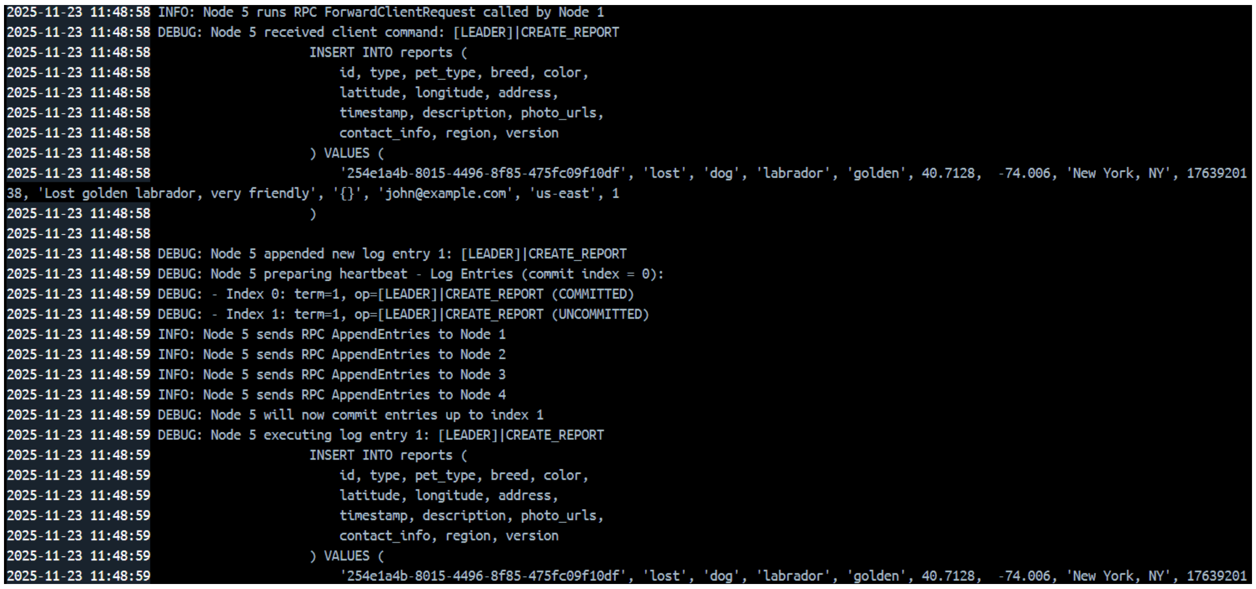
```
2025-11-23 03:03:25 INFO: Node 5 runs RPC AppendEntries called by Node 1
2025-11-23 03:03:25 DEBUG: Node 5 is FOLLOWER (term = 1, lead = 1, voted = None, timeout = 2.6)
2025-11-23 03:03:25 DEBUG: Node 5 logs after AppendEntries - Log Entries (no committed entries):
2025-11-23 03:03:26 INFO: Node 5 runs RPC AppendEntries called by Node 1
2025-11-23 03:03:26 DEBUG: Node 5 is FOLLOWER (term = 1, lead = 1, voted = None, timeout = 2.6)
2025-11-23 03:03:26 DEBUG: Node 5 logs after AppendEntries - Log Entries (no committed entries):
2025-11-23 03:03:26 DEBUG: - Index 0: term=1, op=[LEADER]|CREATE_REPORT (UNCOMMITTED)
2025-11-23 03:03:27 INFO: Node 5 runs RPC AppendEntries called by Node 1
2025-11-23 03:03:27 DEBUG: Node 5 is FOLLOWER (term = 1, lead = 1, voted = None, timeout = 2.6)
2025-11-23 03:03:27 DEBUG: Node 5 will now commit entries up to index 0
2025-11-23 03:03:27 DEBUG: Node 5 logs after AppendEntries - Log Entries (commit index = 0):
2025-11-23 03:03:27 DEBUG: - Index 0: term=1, op=[LEADER]|CREATE_REPORT (COMMITTED)
```

- ○ As such, this test case is successful. The leader node (Node 1) received the client request (a CREATE_REPORT query) and added the command to its log. The leader node then sent its log on a heartbeat to other nodes, and other nodes replicated the log and replied back with an ACK. When the leader node received majority ACKs, it executed and committed the entry. The commit was then replicated to the other nodes on the next heartbeat. Note that the follower nodes did not execute the command/entry as it is a leader only command.

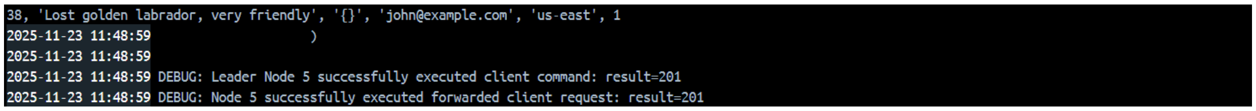
Test Case 4: Follower Nodes Properly Forwards ClientRequest to the Leader Node And Client Receives Response

- Scenario: A client submits a POST request to create a report via the API gateway and the NGINX service forwards the request to a follower node. All nodes are running.
- Expected Behavior:
  - ○ The follower node receives the client request and using the ForwardClientRequest RPC sends it to the leader node.
  - ○ The leader node will then receive the client request and create a new entry in its log. On the next heartbeat, the AppendEntries RPC will contain this new log.
  - ○ When followers complete the AppendEntries RPC from the leader node and return sufficient ACKs, the leader will then execute the operation in the log entry.
  - ○ The result is then returned to the follower node who forwarded it. The follower node then returns the result back to the client through NGINX and the layers.
- Evaluation:

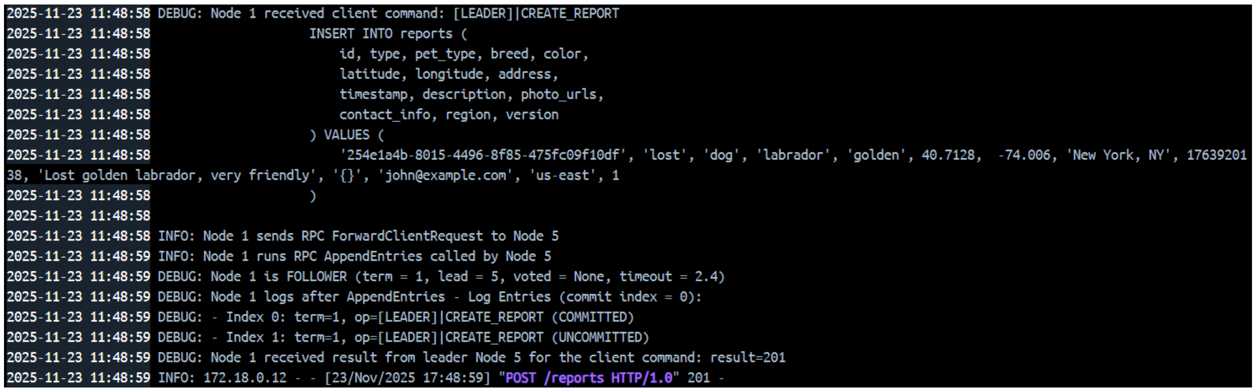
- Run `docker-compose -f docker-compose-layered-raft.yml up --build` and wait until AppendEntries RPCs are being sent
- In a separate terminal, execute the following command:
```
curl -X POST http://localhost:8080/api/reports \
  -H "Content-Type: application/json" \
  -d '{
    "type": "lost",
    "pet_type": "dog",
    "breed": "labrador",
    "color": "golden",
    "location": {
      "latitude": 40.7128,
      "longitude": -74.0060,
      "address": "New York, NY"
    },
    "description": "Lost golden labrador, very friendly",
    "contact_info": "john@example.com"
  }'
```
- Observe the node's logs and check to see that the leader received the client command. From running a ForwardClientRequest RPC. This command may need to be executed more than once to see the ForwardClientRequest RPC logs on the leader node's log. Keep track of the source node that called the ForwardClientRequest RPC.
- Verify that the leader node commits and executes the new client operation. Along with the result returned from executing the operation.
- Verify in the logs of the sender Node that called the ForwardClientRequest RPC that the result is from the leader's execution.
- Verify that the result obtained in the sender Node is seen by the client.
- Leader Node's log below, two screenshots (Node 5)



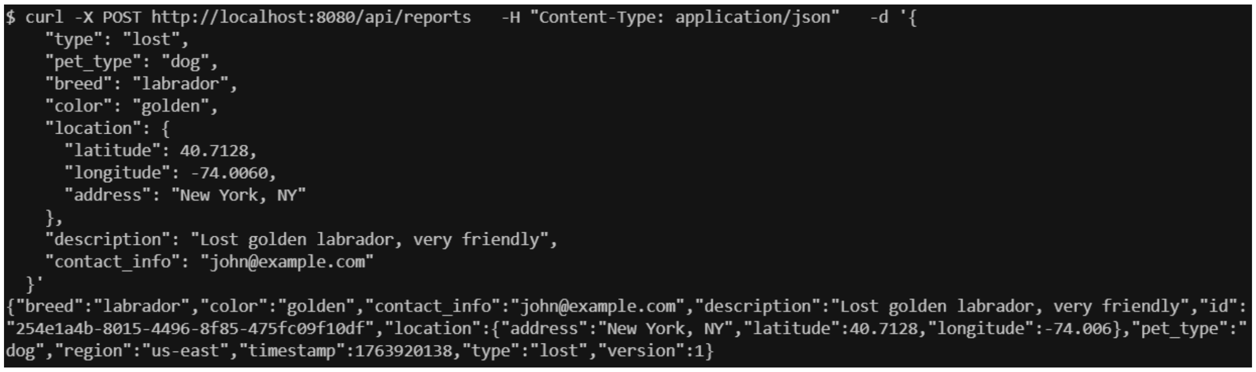
```
2025-11-23 11:48:58 INFO: Node 5 runs RPC ForwardClientRequest called by Node 1
2025-11-23 11:48:58 DEBUG: Node 5 received client command: [LEADER]|CREATE_REPORT
2025-11-23 11:48:58                     INSERT INTO reports (
2025-11-23 11:48:58                         id, type, pet_type, breed, color,
2025-11-23 11:48:58                         latitude, longitude, address,
2025-11-23 11:48:58                         timestamp, description, photo_urls,
2025-11-23 11:48:58                         contact_info, region, version
2025-11-23 11:48:58                     ) VALUES (
2025-11-23 11:48:58                         '254e1a4b-8015-4496-8f85-475fc09f10df', 'lost', 'dog', 'labrador', 'golden', 40.7128,  -74.006, 'New York, NY', 17639201
38, 'Lost golden labrador, very friendly', '{}', 'john@example.com', 'us-east', 1
2025-11-23 11:48:58                     )
2025-11-23 11:48:58
2025-11-23 11:48:58 DEBUG: Node 5 appended new log entry 1: [LEADER]|CREATE_REPORT
2025-11-23 11:48:59 DEBUG: Node 5 preparing heartbeat - Log Entries (commit index = 0):
2025-11-23 11:48:59 DEBUG: - Index 0: term=1, op=[LEADER]|CREATE_REPORT (COMMITTED)
2025-11-23 11:48:59 DEBUG: - Index 1: term=1, op=[LEADER]|CREATE_REPORT (UNCOMMITTED)
2025-11-23 11:48:59 INFO: Node 5 sends RPC AppendEntries to Node 1
2025-11-23 11:48:59 INFO: Node 5 sends RPC AppendEntries to Node 2
2025-11-23 11:48:59 INFO: Node 5 sends RPC AppendEntries to Node 3
2025-11-23 11:48:59 INFO: Node 5 sends RPC AppendEntries to Node 4
2025-11-23 11:48:59 DEBUG: Node 5 will now commit entries up to index 1
2025-11-23 11:48:59 DEBUG: Node 5 executing log entry 1: [LEADER]|CREATE_REPORT
2025-11-23 11:48:59                     INSERT INTO reports (
2025-11-23 11:48:59                         id, type, pet_type, breed, color,
2025-11-23 11:48:59                         latitude, longitude, address,
2025-11-23 11:48:59                         timestamp, description, photo_urls,
2025-11-23 11:48:59                         contact_info, region, version
2025-11-23 11:48:59                     ) VALUES (
2025-11-23 11:48:59                         '254e1a4b-8015-4496-8f85-475fc09f10df', 'lost', 'dog', 'labrador', 'golden', 40.7128,  -74.006, 'New York, NY', 17639201
```


```
38, 'Lost golden labrador, very friendly', '{}', 'john@example.com', 'us-east', 1
2025-11-23 11:48:59                     )
2025-11-23 11:48:59
2025-11-23 11:48:59 DEBUG: Leader Node 5 successfully executed client command: result=201
2025-11-23 11:48:59 DEBUG: Node 5 successfully executed forwarded client request: result=201
```

- From these screenshots above, Node 5 runs the ForwardClientRequest RPC called by Node 1, receives the client command, and then continues to create a log entry with the command as the operation.
- Later on, the operation is then sent on the next heartbeat, and the cluster returns majority ACKs to the leader. The leader then executes and commits the operation. Once execution is completed, the leader node returns the result back to the caller of the ForwardClientRequest RPC
- Below is the log of the caller of the ForwardClientRequest RPC (Node 1) at around the same time interval


```
2025-11-23 11:48:58 DEBUG: Node 1 received client command: [LEADER]|CREATE_REPORT
2025-11-23 11:48:58                     INSERT INTO reports (
2025-11-23 11:48:58                         id, type, pet_type, breed, color,
2025-11-23 11:48:58                         latitude, longitude, address,
2025-11-23 11:48:58                         timestamp, description, photo_urls,
2025-11-23 11:48:58                         contact_info, region, version
2025-11-23 11:48:58                     ) VALUES (
2025-11-23 11:48:58                         '254e1a4b-8015-4496-8f85-475fc09f10df', 'lost', 'dog', 'labrador', 'golden', 40.7128,  -74.006, 'New York, NY', 17639201
38, 'Lost golden labrador, very friendly', '{}', 'john@example.com', 'us-east', 1
2025-11-23 11:48:58                     )
2025-11-23 11:48:58
2025-11-23 11:48:58 INFO: Node 1 sends RPC ForwardClientRequest to Node 5
2025-11-23 11:48:59 INFO: Node 1 runs RPC AppendEntries called by Node 5
2025-11-23 11:48:59 DEBUG: Node 1 is FOLLOWER (term = 1, lead = 5, voted = None, timeout = 2.4)
2025-11-23 11:48:59 DEBUG: Node 1 logs after AppendEntries - Log Entries (commit index = 0):
2025-11-23 11:48:59 DEBUG: - Index 0: term=1, op=[LEADER]|CREATE_REPORT (COMMITTED)
2025-11-23 11:48:59 DEBUG: - Index 1: term=1, op=[LEADER]|CREATE_REPORT (UNCOMMITTED)
2025-11-23 11:48:59 DEBUG: Node 1 received result from leader Node 5 for the client command: result=201
2025-11-23 11:48:59 INFO: 172.18.0.12 - - [23/Nov/2025 17:48:59] "POST /reports HTTP/1.0" 201 -
```

- Here, Node 1 received the client request (and is not the leader), so it calls the ForwardClientRequest RPC to the leader node (times match up to the leader's Docker logs too). Node 1 also gave an acknowledgement to Node 5 for the new log entry. Also, once Node 5 sends the result from executing the operation back to Node 1, Node 1 captures this result and then proceeds to send an HTTP response back with that same captured result.
- Also from the client, we can see that the request was sent and processed without an error occurring (meaning an OK response code was received, being 201).


```
$ curl -X POST http://localhost:8080/api/reports   -H "Content-Type: application/json"   -d '{
    "type": "lost",
    "pet_type": "dog",
    "breed": "labrador",
    "color": "golden",
    "location": {
      "latitude": 40.7128,
      "longitude": -74.0060,
      "address": "New York, NY"
    },
    "description": "Lost golden labrador, very friendly",
    "contact_info": "john@example.com"
  }'
{"breed":"labrador","color":"golden","contact_info":"john@example.com","description":"Lost golden labrador, very friendly","id":
"254e1a4b-8015-4496-8f85-475fc09f10df","location":{"address":"New York, NY","latitude":40.7128,"longitude":-74.006},"pet_type":"
dog","region":"us-east","timestamp":1763920138,"type":"lost","version":1}
```

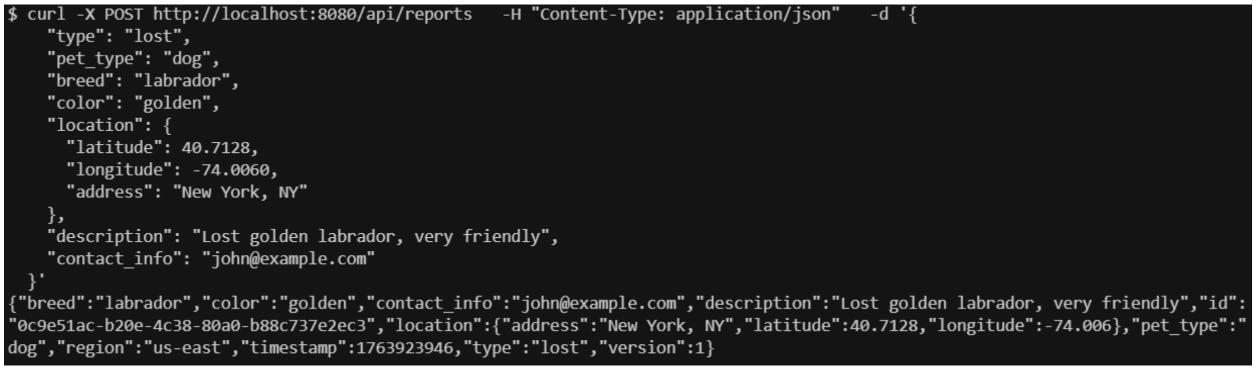
- As such, this test case is successful. A follower node received the client request and forwarded it to the leader node. Once the leader node executed the request, that follower node received the result. This result was then captured and sent back to the client successfully.

Test Case 5: The Raft Cluster Informs The Client If There Is No Leader Node. However, the Raft Cluster Can Still Serve Read Only Requests.

- Scenario: Similar to *Test Case 2*, all but two nodes are functioning in the cluster. Both nodes must not be the leader node.
- Expected Behavior:
    - If the client submits a create report request (requiring consensus and a leader node to execute), the client should see a response detailing the state of the raft cluster.
    - The client is still able to make report requests, as this is a read operation and does not modify the state of the database or nodes. This operation is not sent to the Raft Cluster, and instead is executed locally on the node that received it.
- Evaluation:
    - Run `docker-compose -f docker-compose-layered-raft.yml up --build` and wait until AppendEntries RPCs are being sent
    - In a separate terminal, execute the following command:

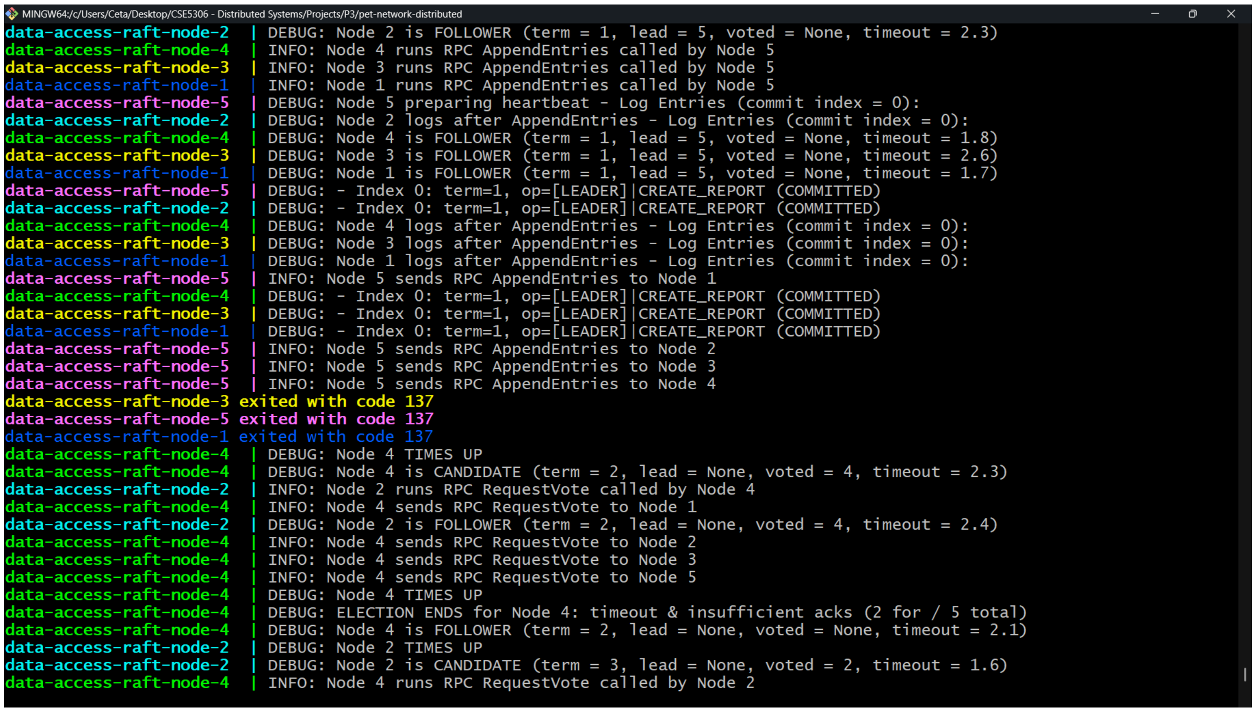
```
curl -X POST http://localhost:8080/api/reports \
  -H "Content-Type: application/json" \
  -d '{
    "type": "lost",
    "pet_type": "dog",
    "breed": "labrador",
    "color": "golden",
    "location": {
      "latitude": 40.7128,
      "longitude": -74.0060,
      "address": "New York, NY"
    },
    "description": "Lost golden labrador, very friendly",
    "contact_info": "john@example.com"
  }'
```

    - From the return result to the client of this command, identify the id and save that value somewhere. It will be used to test a GET request later on. The screenshot below shows an example output after running the command.
      (The id for this evaluation is `0c9e51ac-b20e-4c38-80a0-b88c737e2ec3`)



```
$ curl -X POST http://localhost:8080/api/reports   -H "Content-Type: application/json"   -d '{
    "type": "lost",
    "pet_type": "dog",
    "breed": "labrador",
    "color": "golden",
    "location": {
      "latitude": 40.7128,
      "longitude": -74.0060,
      "address": "New York, NY"
    },
    "description": "Lost golden labrador, very friendly",
    "contact_info": "john@example.com"
  }'
{"breed":"labrador","color":"golden","contact_info":"john@example.com","description":"Lost golden labrador, very friendly","id":
"0c9e51ac-b20e-4c38-80a0-b88c737e2ec3","location":{"address":"New York, NY","latitude":40.7128,"longitude":-74.006},"pet_type":"
dog","region":"us-east","timestamp":1763923946,"type":"lost","version":1}
```

- Identify the leader node and stop its container and two other nodes containers. In the terminal output below, Node 5 is the leader, so the containers for Node 5, Node 1 and Node 3 were stopped.

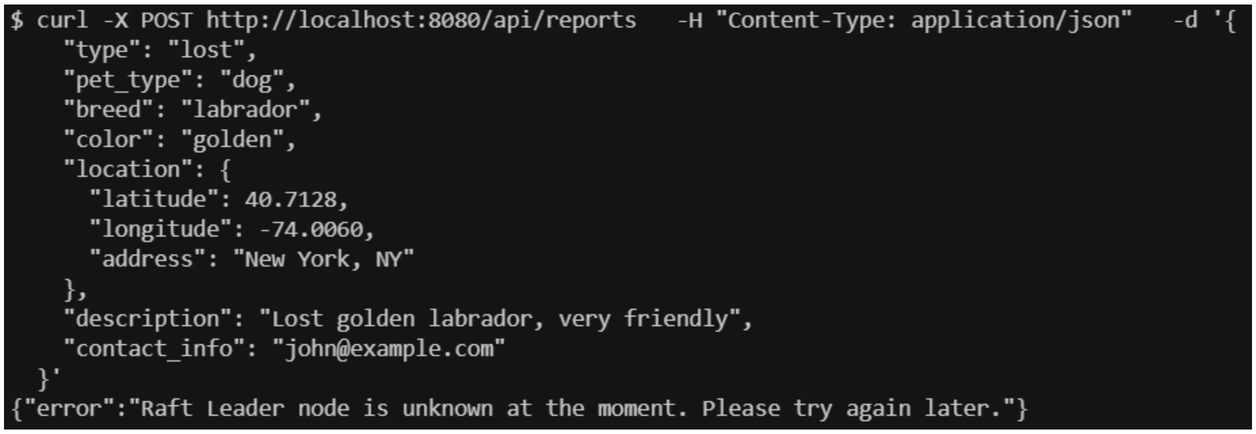


```
MINGW64:/c/Users/Ceta/Desktop/CSE5306 - Distributed Systems/Projects/P3/pet-network-distributed
data-access-raft-node-2  | DEBUG: Node 2 is FOLLOWER (term = 1, lead = 5, voted = None, timeout = 2.3)
data-access-raft-node-4  | INFO: Node 4 runs RPC AppendEntries called by Node 5
data-access-raft-node-3  | INFO: Node 3 runs RPC AppendEntries called by Node 5
data-access-raft-node-1  | INFO: Node 1 runs RPC AppendEntries called by Node 5
data-access-raft-node-5  | DEBUG: Node 5 preparing heartbeat - Log Entries (commit index = 0):
data-access-raft-node-2  | DEBUG: Node 2 logs after AppendEntries - Log Entries (commit index = 0):
data-access-raft-node-4  | DEBUG: Node 4 is FOLLOWER (term = 1, lead = 5, voted = None, timeout = 1.8)
data-access-raft-node-3  | DEBUG: Node 3 is FOLLOWER (term = 1, lead = 5, voted = None, timeout = 2.6)
data-access-raft-node-1  | DEBUG: Node 1 is FOLLOWER (term = 1, lead = 5, voted = None, timeout = 1.7)
data-access-raft-node-5  | DEBUG: - Index 0: term=1, op=[LEADER]|CREATE_REPORT (COMMITTED)
data-access-raft-node-2  | DEBUG: - Index 0: term=1, op=[LEADER]|CREATE_REPORT (COMMITTED)
data-access-raft-node-4  | DEBUG: Node 4 logs after AppendEntries - Log Entries (commit index = 0):
data-access-raft-node-3  | DEBUG: Node 3 logs after AppendEntries - Log Entries (commit index = 0):
data-access-raft-node-1  | DEBUG: Node 1 logs after AppendEntries - Log Entries (commit index = 0):
data-access-raft-node-5  | INFO: Node 5 sends RPC AppendEntries to Node 1
data-access-raft-node-4  | DEBUG: - Index 0: term=1, op=[LEADER]|CREATE_REPORT (COMMITTED)
data-access-raft-node-3  | DEBUG: - Index 0: term=1, op=[LEADER]|CREATE_REPORT (COMMITTED)
data-access-raft-node-1  | DEBUG: - Index 0: term=1, op=[LEADER]|CREATE_REPORT (COMMITTED)
data-access-raft-node-5  | INFO: Node 5 sends RPC AppendEntries to Node 2
data-access-raft-node-5  | INFO: Node 5 sends RPC AppendEntries to Node 3
data-access-raft-node-5  | INFO: Node 5 sends RPC AppendEntries to Node 4
data-access-raft-node-3 exited with code 137
data-access-raft-node-5 exited with code 137
data-access-raft-node-1 exited with code 137
data-access-raft-node-4  | DEBUG: Node 4 TIMES UP
data-access-raft-node-4  | DEBUG: Node 4 is CANDIDATE (term = 2, lead = None, voted = 4, timeout = 2.3)
data-access-raft-node-2  | INFO: Node 2 runs RPC RequestVote called by Node 4
data-access-raft-node-4  | INFO: Node 4 sends RPC RequestVote to Node 1
data-access-raft-node-2  | DEBUG: Node 2 is FOLLOWER (term = 2, lead = None, voted = 4, timeout = 2.4)
data-access-raft-node-4  | INFO: Node 4 sends RPC RequestVote to Node 2
data-access-raft-node-4  | INFO: Node 4 sends RPC RequestVote to Node 3
data-access-raft-node-4  | INFO: Node 4 sends RPC RequestVote to Node 5
data-access-raft-node-4  | DEBUG: Node 4 TIMES UP
data-access-raft-node-4  | DEBUG: ELECTION ENDS for Node 4: timeout & insufficient acks (2 for / 5 total)
data-access-raft-node-4  | DEBUG: Node 4 is FOLLOWER (term = 2, lead = None, voted = None, timeout = 2.1)
data-access-raft-node-2  | DEBUG: Node 2 TIMES UP
data-access-raft-node-2  | DEBUG: Node 2 is CANDIDATE (term = 3, lead = None, voted = 2, timeout = 1.6)
data-access-raft-node-4  | INFO: Node 4 runs RPC RequestVote called by Node 2
```

- In a separate terminal, execute the following command:

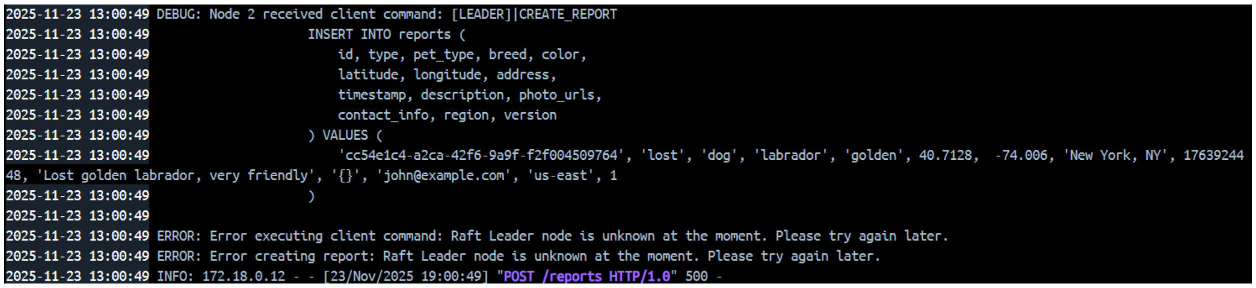
```
curl -X POST http://localhost:8080/api/reports \
  -H "Content-Type: application/json" \
  -d '{
    "type": "lost",
    "pet_type": "dog",
    "breed": "labrador",
    "color": "golden",
    "location": {
      "latitude": 40.7128,
      "longitude": -74.0060,
      "address": "New York, NY"
    },
    "description": "Lost golden labrador, very friendly",
    "contact_info": "john@example.com"
  }'
```

- Verify that the response message states that there is no Raft leader and the server cannot process the request. The below screenshots shows this scenario.



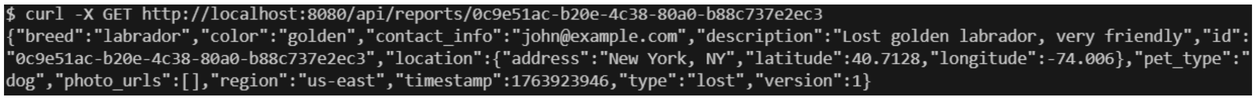
```
$ curl -X POST http://localhost:8080/api/reports   -H "Content-Type: application/json"   -d '{
    "type": "lost",
    "pet_type": "dog",
    "breed": "labrador",
    "color": "golden",
    "location": {
      "latitude": 40.7128,
      "longitude": -74.0060,
      "address": "New York, NY"
    },
    "description": "Lost golden labrador, very friendly",
    "contact_info": "john@example.com"
  }'
{"error":"Raft Leader node is unknown at the moment. Please try again later."}
```

- Observe and identify the logs of the Node that received the client request. Below shows the logs of Node 2, which detected that there was no leader, and returned back to the client that there was no leader.



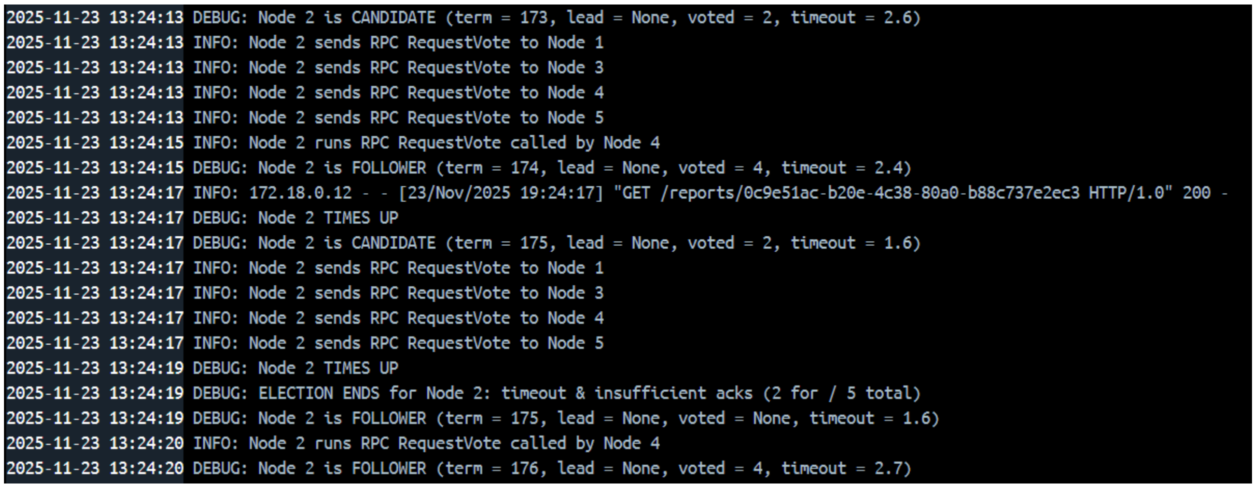
```
2025-11-23 13:00:49 DEBUG: Node 2 received client command: [LEADER]|CREATE_REPORT
2025-11-23 13:00:49                     INSERT INTO reports (
2025-11-23 13:00:49                         id, type, pet_type, breed, color,
2025-11-23 13:00:49                         latitude, longitude, address,
2025-11-23 13:00:49                         timestamp, description, photo_urls,
2025-11-23 13:00:49                         contact_info, region, version
2025-11-23 13:00:49                     ) VALUES (
2025-11-23 13:00:49                         'cc54e1c4-a2ca-42f6-9a9f-f2f004509764', 'lost', 'dog', 'labrador', 'golden', 40.7128,  -74.006, 'New York, NY', 17639244
48, 'Lost golden labrador, very friendly', '{}', 'john@example.com', 'us-east', 1
2025-11-23 13:00:49                     )
2025-11-23 13:00:49
2025-11-23 13:00:49 ERROR: Error executing client command: Raft Leader node is unknown at the moment. Please try again later.
2025-11-23 13:00:49 ERROR: Error creating report: Raft Leader node is unknown at the moment. Please try again later.
2025-11-23 13:00:49 INFO: 172.18.0.12 - - [23/Nov/2025 19:00:49] "POST /reports HTTP/1.0" 500 -
```

- Due to how the original project is structured with the cache layer, to test if the Raft cluster can serve GET requests the cache system of the project must be disabled. To disable the cache, stop the container running `redis-layered` in Docker Desktop or run the docker stop command.
- Using the id from the first POST request, execute the following command in a separate terminal:
  `curl -X GET http://localhost:8080/api/reports/<report-id>`
- The following image shows the result of executing the above command with the id defined back with the first POST request command.



```
$ curl -X GET http://localhost:8080/api/reports/0c9e51ac-b20e-4c38-80a0-b88c737e2ec3
{"breed":"labrador","color":"golden","contact_info":"john@example.com","description":"Lost golden labrador, very friendly","id":
"0c9e51ac-b20e-4c38-80a0-b88c737e2ec3","location":{"address":"New York, NY","latitude":40.7128,"longitude":-74.006},"pet_type":"
dog","photo_urls":[],"region":"us-east","timestamp":1763923946,"type":"lost","version":1}
```

- Observe the remaining node's logs to see which one handled the client GET request. Here, Node 2 was responsible for serving that request as shown in the following screenshot below



```
2025-11-23 13:24:13 DEBUG: Node 2 is CANDIDATE (term = 173, lead = None, voted = 2, timeout = 2.6)
2025-11-23 13:24:13 INFO: Node 2 sends RPC RequestVote to Node 1
2025-11-23 13:24:13 INFO: Node 2 sends RPC RequestVote to Node 3
2025-11-23 13:24:13 INFO: Node 2 sends RPC RequestVote to Node 4
2025-11-23 13:24:13 INFO: Node 2 sends RPC RequestVote to Node 5
2025-11-23 13:24:15 INFO: Node 2 runs RPC RequestVote called by Node 4
2025-11-23 13:24:15 DEBUG: Node 2 is FOLLOWER (term = 174, lead = None, voted = 4, timeout = 2.4)
2025-11-23 13:24:17 INFO: 172.18.0.12 - - [23/Nov/2025 19:24:17] "GET /reports/0c9e51ac-b20e-4c38-80a0-b88c737e2ec3 HTTP/1.0" 200 -
2025-11-23 13:24:17 DEBUG: Node 2 TIMES UP
2025-11-23 13:24:17 DEBUG: Node 2 is CANDIDATE (term = 175, lead = None, voted = 2, timeout = 1.6)
2025-11-23 13:24:17 INFO: Node 2 sends RPC RequestVote to Node 1
2025-11-23 13:24:17 INFO: Node 2 sends RPC RequestVote to Node 3
2025-11-23 13:24:17 INFO: Node 2 sends RPC RequestVote to Node 4
2025-11-23 13:24:17 INFO: Node 2 sends RPC RequestVote to Node 5
2025-11-23 13:24:19 DEBUG: Node 2 TIMES UP
2025-11-23 13:24:19 DEBUG: ELECTION ENDS for Node 2: timeout & insufficient acks (2 for / 5 total)
2025-11-23 13:24:19 DEBUG: Node 2 is FOLLOWER (term = 175, lead = None, voted = None, timeout = 1.6)
2025-11-23 13:24:20 INFO: Node 2 runs RPC RequestVote called by Node 4
2025-11-23 13:24:20 DEBUG: Node 2 is FOLLOWER (term = 176, lead = None, voted = 4, timeout = 2.7)
```

- And Node 2 (along with Node 4) are still trying to establish a leader, but the Raft Cluster is still able to serve commands that do not modify the database state, as shown by the log of the GET request of that same id with a 200 response code.
- As such, this test case is successful. Given that the Raft Cluster does not have a leader, the client is informed of this when attempting to create a report (an operation that modifies the database). But even though the Raft Cluster doesn’t have a leader, individual nodes are able to serve client requests that don’t modify the database (in this case a GET request that only reads data).