Reproducing Network Packet Loss and Network Delay Experiments: One-Size-Fits-None (NSDI'25)

Network Delay

 # Initial Setup with blockade.yaml

We started by trying to use Blockade as the delay injector, following the approach mentioned in the paper.

Problem Encountered:

Blockade by default creates its own Docker network, which was not aligned with our Docker Compose setup.

Fix:

We modified the blockade.yaml file to match the network and container naming from docker-compose.yaml.

```yaml
containers:
  cassandra_a:
    image: cassandra:3.11
    environment:
      - CASSANDRA_CLUSTER_NAME=TestCluster
      - CASSANDRA_LISTEN_ADDRESS=auto
      - CASSANDRA_BROADCAST_ADDRESS=auto
      - CASSANDRA_RPC_ADDRESS=0.0.0.0
      - CASSANDRA_BROADCAST_RPC_ADDRESS=auto
    command: ["bash", "-c", "cassandra -R"]

  cassandra_b:
    image: cassandra:3.11
    environment:
      - CASSANDRA_CLUSTER_NAME=TestCluster
      - CASSANDRA_SEEDS=cassandra_a
      - CASSANDRA_LISTEN_ADDRESS=auto
      - CASSANDRA_BROADCAST_ADDRESS=auto
      - CASSANDRA_RPC_ADDRESS=0.0.0.0
      - CASSANDRA_BROADCAST_RPC_ADDRESS=auto
    command: ["bash", "-c", "cassandra -R"]

  cassandra_c:
    image: cassandra:3.11
    environment:
      - CASSANDRA_CLUSTER_NAME=TestCluster
      - CASSANDRA_SEEDS=cassandra_a
      - CASSANDRA_LISTEN_ADDRESS=auto
      - CASSANDRA_BROADCAST_ADDRESS=auto
      - CASSANDRA_RPC_ADDRESS=0.0.0.0
      - CASSANDRA_BROADCAST_RPC_ADDRESS=auto
    command: ["bash", "-c", "cassandra -R"]
```

Also matched the network settings (bridge mode and subnet) accordingly.

Before

```
cc@osfn-cc: ~/cass:   Command Prompt   Command Prompt
(blockade-venv) cc@osfn-cc:~/cassandra-demo$ docker ps
CONTAINER ID   IMAGE            COMMAND                  CREATED       STATUS
     PORTS
     NAMES
8091f84306d0   cassandra:3.11   "docker-entrypoint.s…"   7 hours ago   Up 4 minut
es   7000-7001/tcp, 7199/tcp, 9160/tcp, 0.0.0.0:9242->9042/tcp, [::]:9242->9042/t
cp   cassandra_c
bdd77930f980   cassandra:3.11   "docker-entrypoint.s…"   7 hours ago   Up 4 minut
es   7000-7001/tcp, 7199/tcp, 9160/tcp, 0.0.0.0:9142->9042/tcp, [::]:9142->9042/t
cp   cassandra_b
1d5eb3655ca7   cassandra:3.11   "docker-entrypoint.s…"   7 hours ago   Up 4 minut
es   7000-7001/tcp, 7199/tcp, 9160/tcp, 0.0.0.0:9042->9042/tcp, [::]:9042->9042/t
cp   cassandra_a
(blockade-venv) cc@osfn-cc:~/cassandra-demo$ blockade up

Error:
a blockade already exists in here - you may want to destroy it first

(blockade-venv) cc@osfn-cc:~/cassandra-demo$ blockade status
NODE            CONTAINER ID    STATUS  IP              NETWORK    PARTITION
a               fc4a926b53a4    DOWN                    UNKNOWN
b               7b0b61ba2508    DOWN                    UNKNOWN
c               9bd0e5f1c30a    DOWN                    UNKNOWN
(blockade-venv) cc@osfn-cc:~/cassandra-demo$ client_loop: send disconnect: Connec
tion reset

C:\Users\jeezx>ssh -i D:\key\yizzz-mj-trace.pem cc@192.5.86.226
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.4.0-198-generic x86_64)
```

After

```
(blockade-venv) cc@osfn-cc:~/cassandra-demo$ docker compose up -d
[+] Running 3/3
 ✔Container cassandra_a  Running
                                        0.0s
 ✔Container cassandra_b  Running
                                        0.0s
 ✔Container cassandra_c  Running
                                        0.0s
(blockade-venv) cc@osfn-cc:~/cassandra-demo$ docker ps
CONTAINER ID   IMAGE            COMMAND                  CREATED       STATUS
  PORTS                                                                      NA
MES
8091f84306d0   cassandra:3.11   "docker-entrypoint.s…"   7 hours ago   Up 21 minutes
  7000-7001/tcp, 7199/tcp, 9160/tcp, 0.0.0.0:9242->9042/tcp, [::]:9242->9042/tcp   ca
ssandra_c
bdd77930f980   cassandra:3.11   "docker-entrypoint.s…"   7 hours ago   Up 21 minutes
  7000-7001/tcp, 7199/tcp, 9160/tcp, 0.0.0.0:9142->9042/tcp, [::]:9142->9042/tcp   ca
ssandra_b
1d5eb3655ca7   cassandra:3.11   "docker-entrypoint.s…"   7 hours ago   Up 21 minutes
  7000-7001/tcp, 7199/tcp, 9160/tcp, 0.0.0.0:9042->9042/tcp, [::]:9042->9042/tcp   ca
ssandra_a
(blockade-venv) cc@osfn-cc:~/cassandra-demo$ blockade up

Error:
a blockade already exists in here - you may want to destroy it first

(blockade-venv) cc@osfn-cc:~/cassandra-demo$ blockade destroy
(blockade-venv) cc@osfn-cc:~/cassandra-demo$ blockade up
NODE            CONTAINER ID    STATUS  IP              NETWORK    PARTITION
cassandra_a     afb57c718437    UP      172.17.0.2      FLAKY
cassandra_b     2bc0cd46e28d    UP      172.17.0.3      FLAKY
cassandra_c     5240091331a6    UP      172.17.0.4      FLAKY
(blockade-venv) cc@osfn-cc:~/cassandra-demo$ blockade status
NODE            CONTAINER ID    STATUS  IP              NETWORK    PARTITION
cassandra_a     afb57c718437    UP      172.17.0.2      FLAKY
cassandra_b     2bc0cd46e28d    UP      172.17.0.3      FLAKY
cassandra_c     5240091331a6    UP      172.17.0.4      FLAKY
(blockade-venv) cc@osfn-cc:~/cassandra-demo$
```

Reset to the fast mode

```
(blockade-venv) cc@osfn-cc:~/cassandra-demo$ blockade fast cassa
ndra_a cassandra_b cassandra_c
(blockade-venv) cc@osfn-cc:~/cassandra-demo$ blockade status
NODE            CONTAINER ID    STATUS  IP              NETWORK
   PARTITION
cassandra_a     afb57c718437    UP      172.17.0.2      NORMAL
cassandra_b     2bc0cd46e28d    UP      172.17.0.3      NORMAL
cassandra_c     5240091331a6    UP      172.17.0.4      NORMAL
(blockade-venv) cc@osfn-cc:~/cassandra-demo$
```

We can see, the network status turn to NORMAL from the FLAKY status before

 # Blockade Injection Test (Flaky / Packet Loss)

We attempted to inject packet loss:

blockade flaky cassandra_b 80%

Then entered cassandra_a container:

docker exec -it cassandra_a bash
ping cassandra_b

Problem Encountered:

- There was no observable effect. Despite blockade status showing FLAKY, pings succeeded with no packet loss.

  blockade flaky cassandra_b 80%
  docker exec -it cassandra_a bash

```
(blockade-venv) cc@osfn-cc:~/cassandra-demo$ blockade flaky cassandra_b 80%
(blockade-venv) cc@osfn-cc:~/cassandra-demo$ docker exec -it cassandra_a bash
ping cassandra_b
root@1d5eb3655ca7:/# ping cassandra_b
PING cassandra_b (172.18.0.3) 56(84) bytes of data.
64 bytes from cassandra_b.cassandra-demo_blockade (172.18.0.3): icmp_seq=1 ttl=64 time=0.143 ms
64 bytes from cassandra_b.cassandra-demo_blockade (172.18.0.3): icmp_seq=2 ttl=64 time=0.086 ms
64 bytes from cassandra_b.cassandra-demo_blockade (172.18.0.3): icmp_seq=3 ttl=64 time=0.028 ms
```

We can see here it is not giving any effect. The expected result must be

```
64 bytes from cassandra_b: icmp_seq=1 ttl=64 time=0.123 ms
Request timeout for icmp_seq 2
Request timeout for icmp_seq 3
```

Despite multiple patching attempts, Blockade consistently failed to apply delays or packet loss effectively because of namespace and network isolation issues with Docker Compose-managed containers. Blockade is designed for containers it launches directly. Our Cassandra containers were launched via Docker Compose, and integrating these with Blockade's control flow proved to be brittle and inconsistent.

Therefore, we switched to using `tc netem` commands (via script) inside the containers to inject delay directly at the OS level.

Hypothesis:

- Blockade's default bridge network was not correctly linked to Docker Compose's custom bridge.
- Containers launched by Docker Compose were not being effectively controlled by Blockade.

 # Reset Network State (Cleanup)

To ensure no residual qdisc (traffic control) entries or dangling networks interfered:

docker compose down
blockade destroy
docker container prune -f
docker volume prune -f
docker network prune -f

```
cc@osfn-cc:~/cassandra-demo$ docker compose down
work prune -f
cc@osfn-cc:~/cassandra-demo$ blockade destroy
blockade: command not found
cc@osfn-cc:~/cassandra-demo$ docker container prune -f
Total reclaimed space: 0B
cc@osfn-cc:~/cassandra-demo$ docker volume prune -f
Total reclaimed space: 0B
cc@osfn-cc:~/cassandra-demo$ docker network prune -f
cc@osfn-cc:~/cassandra-demo$
```

We verified that containers were clean and rebuilt the network using only Docker Compose.

Shift to Manual Injection using tc (run_netdelay.sh)

Due to Blockade's ineffectiveness, we developed a manual injection script using Linux tc (traffic control):

./run_netdelay.sh "cassandra_b cassandra_c" "100ms" 30 delay100ms_bc

This script:

1. Injects tc netem delay 100ms to cassandra_b and cassandra_c
2. Waits 15s for the cluster to stabilize
3. Runs cassandra-stress write on cassandra_a for 30s
4. Collects the result log and saves it to a unique folder
5. Removes the delay

 # Verifying tc injection

Before each run, we check tc qdisc show:

docker exec cassandra_b tc qdisc show
docker exec cassandra_c tc qdisc show

It shows:

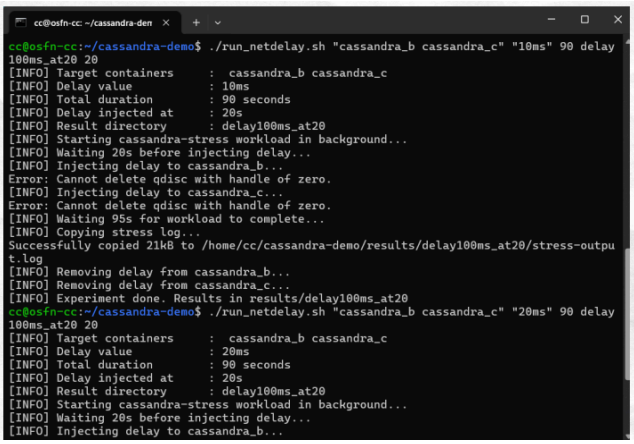
qdisc netem 803a: dev eth0 root refcnt 2 limit 1000 delay 100ms

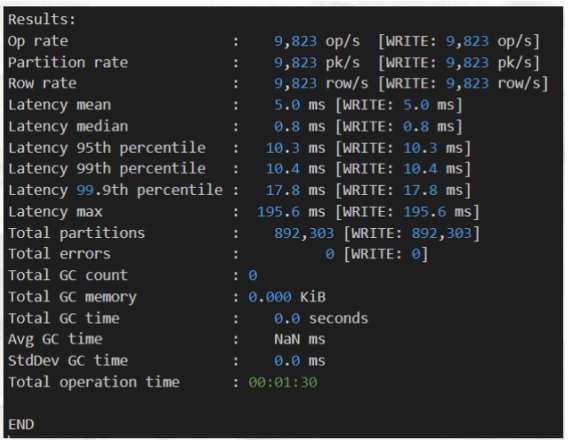
This confirms the delay is correctly injected.

 # Running Experiments

We successfully ran experiments with various delays:

- 1ms: baseline
- 50ms: intermediate
- 75ms: upper-intermediate
- 100ms: high delay





Each run used:

./run_netdelay.sh "cassandra_b cassandra_c" "<delay>" 30 delay<delay>ms_bc

```
./run_netdelay.sh "cassandra_b cassandra_c" "1ms" 30 delay1ms_bc
./run_netdelay.sh "cassandra_b cassandra_c" "50ms" 30 delay50ms_bc
./run_netdelay.sh "cassandra_b cassandra_c" "75ms" 30 delay75ms_bc
./run_netdelay.sh "cassandra_b cassandra_c" "100ms" 30 delay100ms_bc
```

Each script:

- Injects delay via `tc qdisc add ... delay Xms`
- Waits for stabilization
- Executes `cassandra-stress`
- Removes the delay via `tc qdisc del`

 # Output Collection and Visualization

We parsed stress-output.log files to extract:

- Mean latency per second
- Ops/sec (throughput)

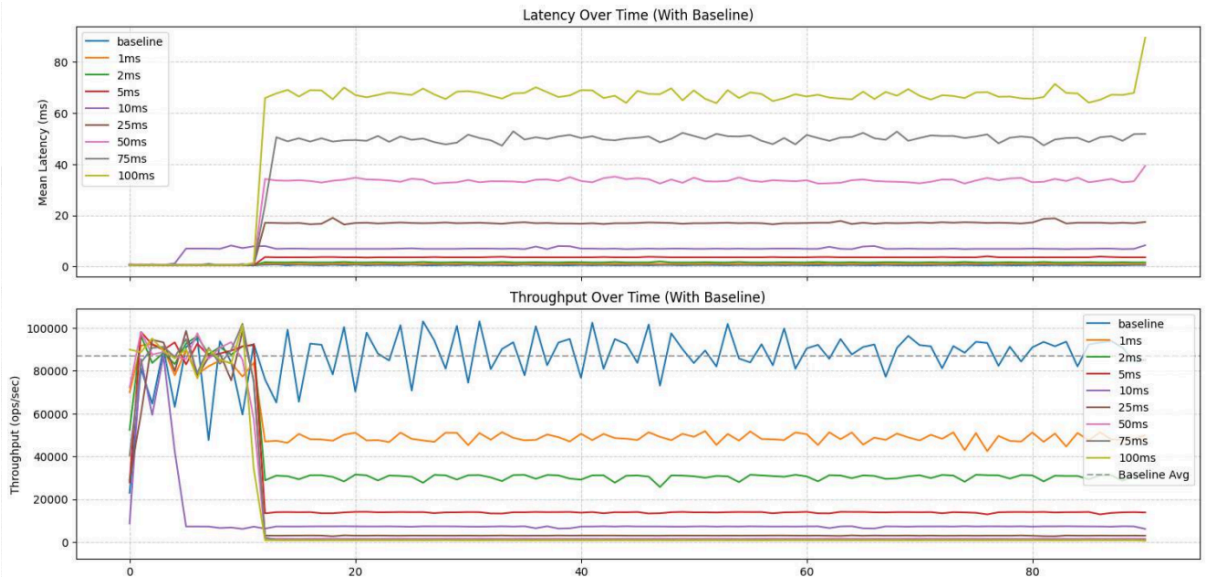
Then we visualized them using Python matplotlib, producing two subplots:

1. Latency over Time

2. Throughput over Time

Observations:

- At 1ms: ~50K ops/sec, ~1ms latency
- At 100ms: ~700 ops/sec, ~100ms latency
- 50ms and 75ms produce expected intermediate degradation



From the results, we can conclude that the experiment was unsuccessful due to several technical and experimental issues:

- Lack of observable performance impact: Across all delay configurations—from microseconds to seconds, both throughput and latency remained almost unchanged. This suggests that the injected faults did not effectively influence the system's behavior.
- Unexpected latency behavior: Higher delay values are expected to increase latency. However, the latency plots remained flat or even decreased, which contradicts expected behavior under network-level faults.
- Insufficient workload pressure: The `cassandra-stress` workload may have been too light, failing to sufficiently stress the system. As a result, even if delay was correctly injected, the system's performance remained unaffected.