Multi-Paxos-PoC

Proof of concept of Multi-Paxos Electrode optimization for Future Internet Technologies course at AGH University of Cracow 2023/24.

Inctroduction

After many trials and battles with flags for Electrode applications, we decided to measure impact only for frist optimization flag, TC_BROADCAST.

In Paxos protocols, one-to-all message broadcasting is widely used:

- the leader node sends preparation messages to all follower nodes
- after receiving enough acknowledgments from followers the leader node sends commit messages to all follower nodes

To implement the above message broadcasting, the most common way is sending the same message multiple times in the user space to different destinations. However, the overhead (i.e., user-kernel crossing and kernel networking stack traversing) of this implementation on the leader node increases linearly as the number of followers increases, while the overhead on each follower node remains constant (so the leader node becomes the system bottleneck).

Electrode provides a flexible host-based broadcasting solution by utilizing eBPF on the TC hook. Here, we require the eBPF program that implements broadcasting operations to attach to the TC hook, because only the TC hook can intercept and process outgoing packets (§2.2). After attaching the eBPF program, user-space applications can call the elec_broadcast() function specified sock_fd, message, and a list of destination IPs to broadcast the message to these destinations through the socket.

Under the hood, the eBPF program makes clones of the message packet using the bpf_clone_redirect() helper function, modifies the destination addresses of cloned packets accordingly, and sends these packets out. The benefit of cloning packets and broadcasting in the kernel compared with sending the same message multiple times in the user space is that we only need to cross the user-kernel boundary and traverse the UDP and socket layer once.

Structure of the repository

In directory `docs` you can find our preliminary understanding of two NSDI conferences and SIGCOMM on the practical use of the eBPF service to optimize the performance of applications or other services.

In directory `Electrode` you can find implementation of Multi-Paxos algorithm with Electrode optimization, cloned from its author repository. The implementation is based on the paper Multi-Paxos with Electrodes: A Framework for Optimizing State Machine Replication Protocols, which we have considered to be the most interesting and promising for our proof of concept, so we decide to reproduce it on few different scenarios.

We decided to reproduce the results of the paper using Docker images and Docker Compose orchestration. In the root directory you can find Dockerfiles and Docker Compose files for each scenario. For each scenario you need to build the Docker image and run the Docker Compose file. The scenarios are:

- `scenario3` - three replicas, one client

```
docker build -t electrode:scenario3 -f ./scenario3/Dockerfile .
docker compose -f ./scenario3/docker-compose.yaml up
# to cleanup containers after running the scenario
docker compose -f ./scenario3/docker-compose.yaml down
```

- `scenario5` - five replicas, one client

```
docker build -t electrode:scenario5 -f ./scenario5/Dockerfile .
docker compose -f ./scenario5/docker-compose.yaml up
# to cleanup containers after running the scenario
docker compose -f ./scenario5/docker-compose.yaml down
```

- `scenario7` - seven replicas, one client

```
docker build -t electrode:scenario7 -f ./scenario7/Dockerfile .
docker compose -f ./scenario7/docker-compose.yaml up
# to cleanup containers after running the scenario
docker compose -f ./scenario7/docker-compose.yaml down
```

Each scenario directory contains:

- `Dockerfile` - Dockerfile for building the Docker image
- `docker-compose.yaml` - Docker Compose file for running the scenario with according number of replicas
- `config.txt` - Text file with replicas IP addresses and ports (container names instead of IP addresses thanks to Docker Compose)
- `fast_user.c` - C script with replicas MAC addresses on line 281
- `fast_common.h` - Header file with cluster size on line 17

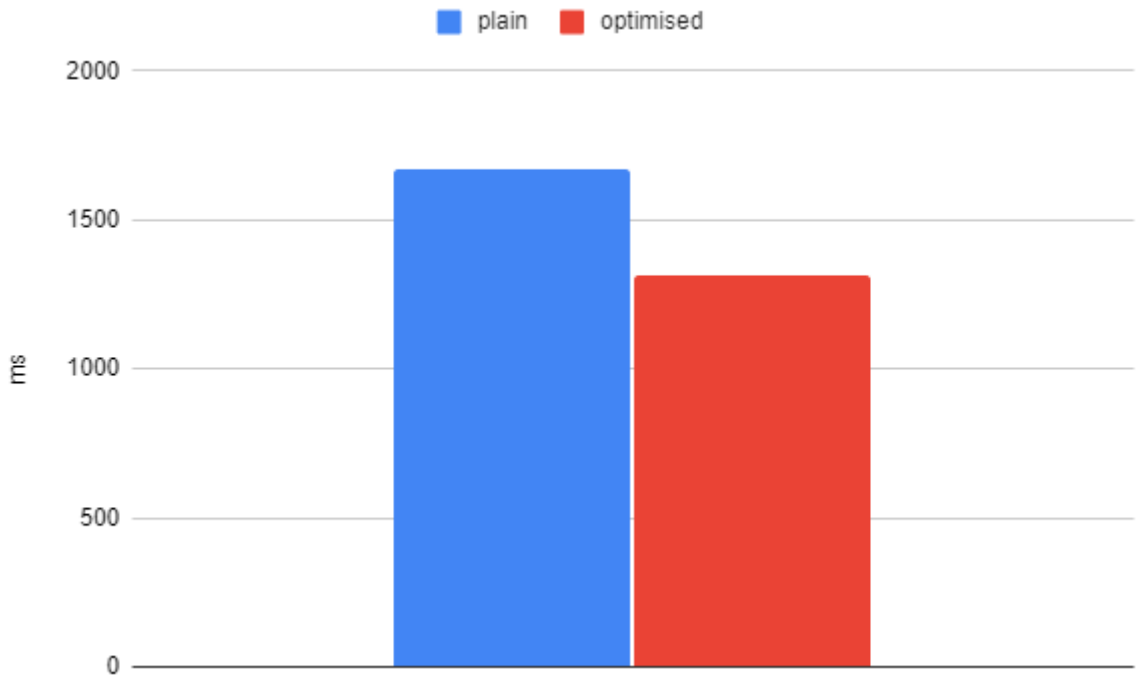
All Dockerfiles have in general the same structure:

- Install required apt packages on Ubuntu 20.04
- Clone the Electrode repository and files specific for the scenario
- Build the Electrode
- Configure NIC and irqbalance
- Execute script with:
    - running `fast` script on NIC
    - command you specify in the Docker Compose file

Client container after proper initialization will send requests to the replicas and measure the latency of the responses. The results will be visible in the terminal.

Results

For 3 replicas, the optimization makes latency drop from 1670ms to 1314ms, which is approximately 21% acceleration.



To check whether the eBPF program is attached to the TC hook, you can go inside any replica and use the following command:

```
docker exec -it replica1 /bin/bash
bpftool prog show
```

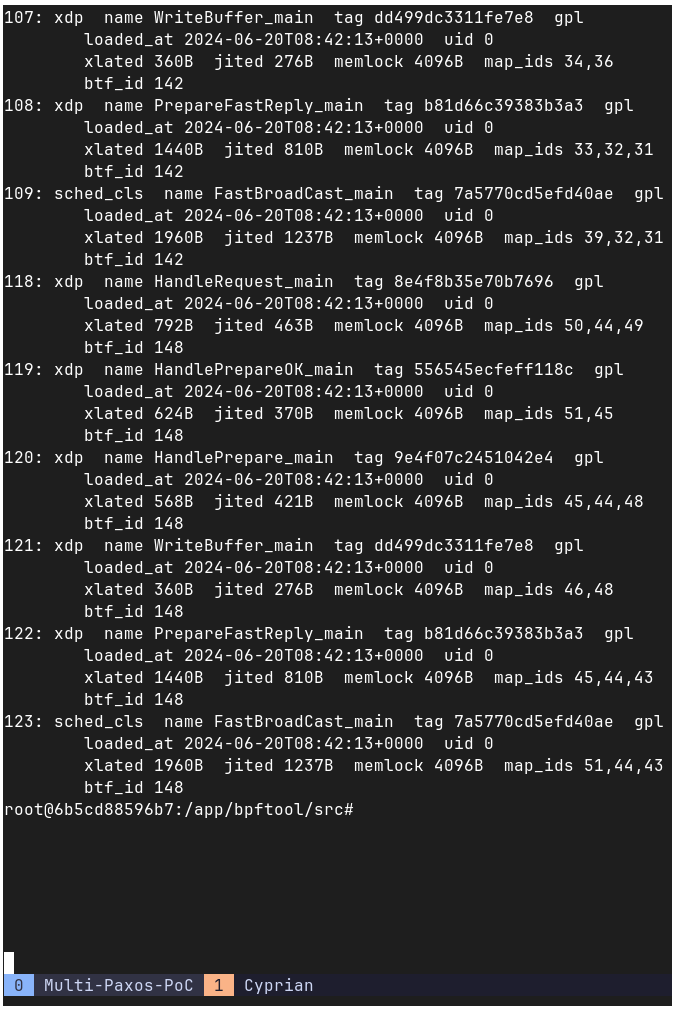
This will display a list of all eBPF programs currently loaded into the kernel. Below are all the programs we loaded during our experiments.

```
root@6b5cd88596b7:/app/bpftool/src# bpftool prog show
2: tracing  name hid_tail_call  tag 7cc47bbf07148bfe  gpl
        loaded_at 2024-06-20T08:41:07+0000  uid 0
        xlated 56B  jited 118B  memlock 4096B  map_ids 2
        btf_id 2
29: lsm  name restrict_filesystems  tag 713a545fe0530ce7  gpl
        loaded_at 2024-06-20T08:41:14+0000  uid 0
        xlated 560B  jited 308B  memlock 4096B  map_ids 13
        btf_id 50
36: cgroup_device  name sd_devices  tag 40ddf486530245f5  gpl
        loaded_at 2024-06-20T08:41:15+0000  uid 0
        xlated 504B  jited 321B  memlock 4096B
37: cgroup_skb  name sd_fw_egress  tag 6deef7357e7b4530  gpl
        loaded_at 2024-06-20T08:41:15+0000  uid 0
        xlated 64B  jited 66B  memlock 4096B
38: cgroup_skb  name sd_fw_ingress  tag 6deef7357e7b4530  gpl
        loaded_at 2024-06-20T08:41:15+0000  uid 0
        xlated 64B  jited 66B  memlock 4096B
39: cgroup_device  name sd_devices  tag ee0e253c78993a24  gpl
        loaded_at 2024-06-20T08:41:15+0000  uid 0
        xlated 416B  jited 267B  memlock 4096B
40: cgroup_skb  name sd_fw_egress  tag 6deef7357e7b4530  gpl
        loaded_at 2024-06-20T08:41:15+0000  uid 0
        xlated 64B  jited 66B  memlock 4096B
41: cgroup_skb  name sd_fw_ingress  tag 6deef7357e7b4530  gpl
        loaded_at 2024-06-20T08:41:15+0000  uid 0
        xlated 64B  jited 66B  memlock 4096B
42: cgroup_skb  name sd_fw_egress  tag 6deef7357e7b4530  gpl
        loaded_at 2024-06-20T08:41:15+0000  uid 0
        xlated 64B  jited 66B  memlock 4096B
43: cgroup_skb  name sd_fw_ingress  tag 6deef7357e7b4530  gpl
        loaded_at 2024-06-20T08:41:15+0000  uid 0
        xlated 64B  jited 66B  memlock 4096B
44: cgroup_device  name sd_devices  tag be31ae23198a0378  gpl
        loaded_at 2024-06-20T08:41:16+0000  uid 0
        xlated 464B  jited 300B  memlock 4096B
45: cgroup_skb  name sd_fw_egress  tag 6deef7357e7b4530  gpl
        loaded_at 2024-06-20T08:41:16+0000  uid 0
        xlated 64B  jited 66B  memlock 4096B
46: cgroup_skb  name sd_fw_ingress  tag 6deef7357e7b4530  gpl
        loaded_at 2024-06-20T08:41:16+0000  uid 0
        xlated 64B  jited 66B  memlock 4096B
47: cgroup_device  name sd_devices  tag be31ae23198a0378  gpl
0 Multi-Paxos-PoC 1 Cyprian
```

```
47: cgroup_device  name sd_devices  tag be31ae23198a0378  gpl
        loaded_at 2024-06-20T08:41:16+0000  uid 0
        xlated 464B  jited 300B  memlock 4096B
48: cgroup_device  name sd_devices  tag ee0e253c78993a24  gpl
        loaded_at 2024-06-20T08:41:16+0000  uid 0
        xlated 416B  jited 267B  memlock 4096B
49: cgroup_device  name sd_devices  tag be31ae23198a0378  gpl
        loaded_at 2024-06-20T08:41:16+0000  uid 0
        xlated 464B  jited 300B  memlock 4096B
50: cgroup_device  name sd_devices  tag ee0e253c78993a24  gpl
        loaded_at 2024-06-20T08:41:16+0000  uid 0
        xlated 416B  jited 267B  memlock 4096B
51: cgroup_device  name sd_devices  tag b37200ab714f0e17  gpl
        loaded_at 2024-06-20T08:41:16+0000  uid 0
        xlated 184B  jited 113B  memlock 4096B
52: cgroup_device  name sd_devices  tag be31ae23198a0378  gpl
        loaded_at 2024-06-20T08:41:16+0000  uid 0
        xlated 464B  jited 300B  memlock 4096B
53: cgroup_device  name sd_devices  tag ee0e253c78993a24  gpl
        loaded_at 2024-06-20T08:41:16+0000  uid 0
        xlated 416B  jited 267B  memlock 4096B
54: cgroup_device  name sd_devices  tag b90a282ee45cfed9  gpl
        loaded_at 2024-06-20T08:41:16+0000  uid 0
        xlated 616B  jited 396B  memlock 4096B
55: cgroup_device  name sd_devices  tag 3a0ef5414c2f6fca  gpl
        loaded_at 2024-06-20T08:41:16+0000  uid 0
        xlated 744B  jited 459B  memlock 4096B
56: cgroup_skb  name sd_fw_egress  tag 6deef7357e7b4530  gpl
        loaded_at 2024-06-20T08:41:16+0000  uid 0
        xlated 64B  jited 66B  memlock 4096B
57: cgroup_skb  name sd_fw_ingress  tag 6deef7357e7b4530  gpl
        loaded_at 2024-06-20T08:41:16+0000  uid 0
        xlated 64B  jited 66B  memlock 4096B
58: cgroup_skb  name sd_fw_egress  tag 6deef7357e7b4530  gpl
        loaded_at 2024-06-20T08:41:16+0000  uid 0
        xlated 64B  jited 66B  memlock 4096B
59: cgroup_skb  name sd_fw_ingress  tag 6deef7357e7b4530  gpl
        loaded_at 2024-06-20T08:41:16+0000  uid 0
        xlated 64B  jited 66B  memlock 4096B
60: cgroup_skb  name sd_fw_egress  tag 6deef7357e7b4530  gpl
        loaded_at 2024-06-20T08:41:16+0000  uid 0
        xlated 64B  jited 66B  memlock 4096B
61: cgroup_skb  name sd_fw_ingress  tag 6deef7357e7b4530  gpl
        loaded_at 2024-06-20T08:41:16+0000  uid 0
0 Multi-Paxos-PoC 1 Cyprian
```

```
61: cgroup_skb  name sd_fw_ingress  tag 6deef7357e7b4530  gpl
        loaded_at 2024-06-20T08:41:16+0000  uid 0
        xlated 64B  jited 66B  memlock 4096B
62: cgroup_device  name sd_devices  tag ccbbf91f3c6979c7  gpl
        loaded_at 2024-06-20T08:41:16+0000  uid 0
        xlated 560B  jited 363B  memlock 4096B
70: cgroup_device  name sd_devices  tag be31ae23198a0378  gpl
        loaded_at 2024-06-20T08:41:37+0000  uid 0
        xlated 464B  jited 300B  memlock 4096B
73: cgroup_device  tag 3918c82a5f4c0360
        loaded_at 2024-06-20T08:42:12+0000  uid 0
        xlated 64B  jited 52B  memlock 4096B
75: tracing  name dump_bpf_map  tag 115f40b5ddc24085  gpl
        loaded_at 2024-06-20T08:42:12+0000  uid 0
        xlated 280B  jited 178B  memlock 4096B  map_ids 16
        btf_id 134
76: tracing  name dump_bpf_prog  tag aa00aac7fe275aa1  gpl
        loaded_at 2024-06-20T08:42:12+0000  uid 0
        xlated 520B  jited 659B  memlock 4096B  map_ids 16
        btf_id 134
81: xdp  name fastPaxos_main  tag 3b185187f1855c4c  gpl
        loaded_at 2024-06-20T08:42:12+0000  uid 0
        xlated 16B  jited 30B  memlock 4096B
        btf_id 138
        pids fast(5221)
84: cgroup_device  tag 3918c82a5f4c0360
        loaded_at 2024-06-20T08:42:12+0000  uid 0
        xlated 64B  jited 52B  memlock 4096B
89: xdp  name fastPaxos_main  tag 3b185187f1855c4c  gpl
        loaded_at 2024-06-20T08:42:13+0000  uid 0
        xlated 16B  jited 30B  memlock 4096B
        btf_id 142
90: xdp  name HandleRequest_main  tag 8e4f8b35e70b7696  gpl
        loaded_at 2024-06-20T08:42:12+0000  uid 0
        xlated 792B  jited 463B  memlock 4096B  map_ids 26,20
        btf_id 138
        pids fast(5221)
91: xdp  name HandlePrepareOK_main  tag 556545ecfeff118c  gpl
        loaded_at 2024-06-20T08:42:13+0000  uid 0
        xlated 624B  jited 370B  memlock 4096B  map_ids 27,21
        btf_id 138
        pids fast(5221)
92: xdp  name HandlePrepare_main  tag 9e4f07c2451042e4  gpl
        loaded_at 2024-06-20T08:42:13+0000  uid 0
0 Multi-Paxos-PoC 1 Cyprian
```

```
92: xdp  name HandlePrepare_main  tag 9e4f07c2451042e4  gpl
        loaded_at 2024-06-20T08:42:13+0000  uid 0
        xlated 568B  jited 421B  memlock 4096B  map_ids 21,20,24
        btf_id 138
        pids fast(5221)
95: cgroup_device  tag 3918c82a5f4c0360
        loaded_at 2024-06-20T08:42:13+0000  uid 0
        xlated 64B  jited 52B  memlock 4096B
96: xdp  name WriteBuffer_main  tag dd499dc3311fe7e8  gpl
        loaded_at 2024-06-20T08:42:13+0000  uid 0
        xlated 360B  jited 276B  memlock 4096B  map_ids 22,24
        btf_id 138
        pids fast(5221)
97: xdp  name PrepareFastReply_main  tag b81d66c39383b3a3  gpl
        loaded_at 2024-06-20T08:42:13+0000  uid 0
        xlated 1440B  jited 810B  memlock 4096B  map_ids 21,20,19
        btf_id 138
        pids fast(5221)
98: sched_cls  name FastBroadCast_main  tag 7a5770cd5efd40ae  gpl
        loaded_at 2024-06-20T08:42:13+0000  uid 0
        xlated 1960B  jited 1237B  memlock 4096B  map_ids 27,20,19
        btf_id 138
        pids fast(5221)
103: xdp  name fastPaxos_main  tag 3b185187f1855c4c  gpl
        loaded_at 2024-06-20T08:42:13+0000  uid 0
        xlated 16B  jited 30B  memlock 4096B
        btf_id 148
104: xdp  name HandleRequest_main  tag 8e4f8b35e70b7696  gpl
        loaded_at 2024-06-20T08:42:13+0000  uid 0
        xlated 792B  jited 463B  memlock 4096B  map_ids 38,32,37
        btf_id 142
105: xdp  name HandlePrepareOK_main  tag 556545ecfeff118c  gpl
        loaded_at 2024-06-20T08:42:13+0000  uid 0
        xlated 624B  jited 370B  memlock 4096B  map_ids 39,33
        btf_id 142
106: xdp  name HandlePrepare_main  tag 9e4f07c2451042e4  gpl
        loaded_at 2024-06-20T08:42:13+0000  uid 0
        xlated 568B  jited 421B  memlock 4096B  map_ids 33,32,36
        btf_id 142
107: xdp  name WriteBuffer_main  tag dd499dc3311fe7e8  gpl
        loaded_at 2024-06-20T08:42:13+0000  uid 0
        xlated 360B  jited 276B  memlock 4096B  map_ids 34,36
        btf_id 142
108: xdp  name PrepareFastReply_main  tag b81d66c39383b3a3  gpl
0 Multi-Paxos-PoC 1 Cyprian
```



Unfortunately, we faced issues with 5 and 7 replicas scenarios. After launching the scenario, the client container warms up correctly, but then replicas start being confused about request and suddenly part of them (with higher indexes) errors with `Segmentation fault`. We tried to debug the issue, but we didn't manage to find the root cause. We suspect that the issue is related to the kernel version (Docker container takes the host kernel version) or the wrong combination of steps we made in the Dockerfile.

We also tried deploying solution on AWS (to overpass the kernel version issue), but we faced the same issue as in the Docker container (installing kernel headers and rebooting kernel didn't help).

Conclusion

We managed to reproduce the results of the paper for the scenario with 3 replicas. We faced issues with 5 and 7 replicas scenarios, which we suspect are related to the kernel version or the wrong combination of steps in the Dockerfile. We also tried deploying the solution on AWS, but we faced the same issue as in the Docker container. The other option that may work is to use local Virtual Machines with proper kernel version, but we didn't have time to try it.