CS 433 Computer Networks
Project Report

CONTAINER NETWORKING
IPVS, IPTABLES & EBPF DATAPLANE

November 28, 2022

Divyanshu Meena 19110081
divyanshu.m@iitgn.ac.in
Mihir Chauhan 19110079
mihir.hc@iitgn.ac.in
Paarth Sachan 19110055
paarth.s@iitgn.ac.in
Varun Barala 19110105
barala.v@iitgn.ac.in
Indian Institute of Technology, Gandhinagar

1 Introduction

Deploying applications on the cloud has always been a daunting task. Virtual Machines virtualize the entire machine down to the hardware layers, which takes a lot of resources and computation power. Containers aim to solve this problem, as they only visualize software layers above the operating system.

With the popularization of microservices-based architecture, modern-day applications need multiple containers running simultaneously, with each service running inside a different container. With numerous containers comes the need for a tool to manage them. Kubernetes is one tool that can handle container orchestration and control their scaling and management for us. It groups containers that make up an application into logical units for easy management and discovery.

Kubernetes has various comments, and kube-proxy is one of them . kube-proxy role is to load-balance traffic, and maintain rules for allowing network communication from inside or outside our cluster.

In this project, we had the following goals:-

- Gain an understanding of Docker Networking. Learn about different Docker drivers available and their use cases.
- Learn about networking inside a cluster, CNIs, their advantages, and when to use them.
- Compare the performance difference when using networking modes, such as IPTables and IPVS in kube-proxy, and the eBPF data plane in Calico.

2 Literature Review

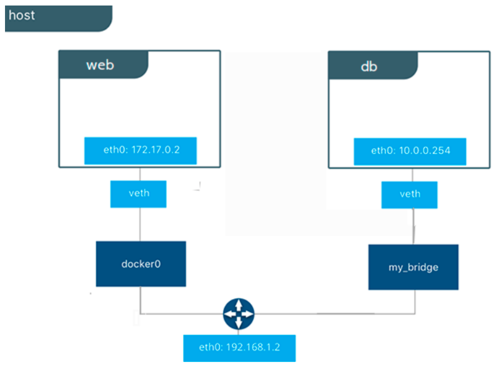
2.1 Docker Networking

Docker networking allows us to attach our running containers to the networks of our choice. Docker network drivers provide pluggable interfaces for actual network implementations.7 different drivers offered are:

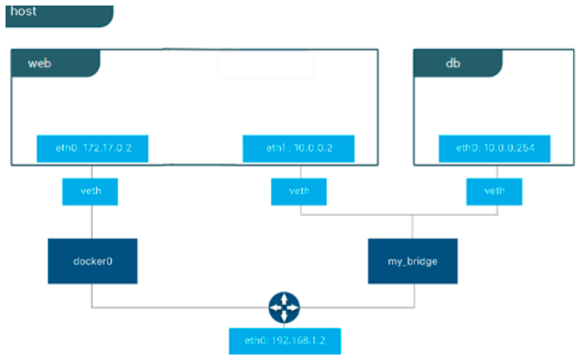
1. bridge
2. overlay
3. host
4. ipvlan
5. macvlan

6. none

7. Third party network plugins



(a) Default bridge network



(b) User defined bridge network

Figure 1: Brigde Networks

Bridge Networks are one of the more popular network drivers which are used. There could be default bridge networks and user defined bridge networks. User defined bridge networks are better than default bridge networks as they provide better isolation, dns resolution ,etc.

2.2 Kube-proxy

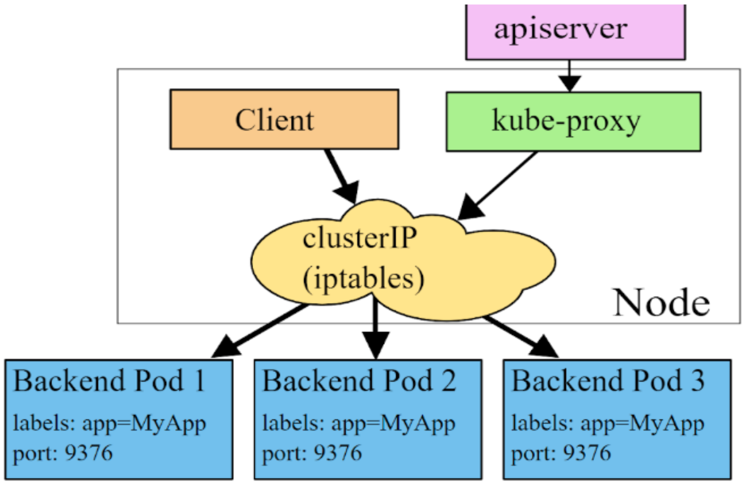


Figure 2: Kube-proxy

Kube-proxy is an essential component of a Kubernetes node. Kube-proxy enforces specific network rules and takes care of load balancing. These rules allow network communication to Pods from network sessions inside or outside a cluster. Three modes of Kube-proxy are iptables,ipvs and userspace. Userspace mode is legacy is very slow and is not recommended. We look below at two suggested methods, namely iptables and ipvs.

2.2.1 IPTables

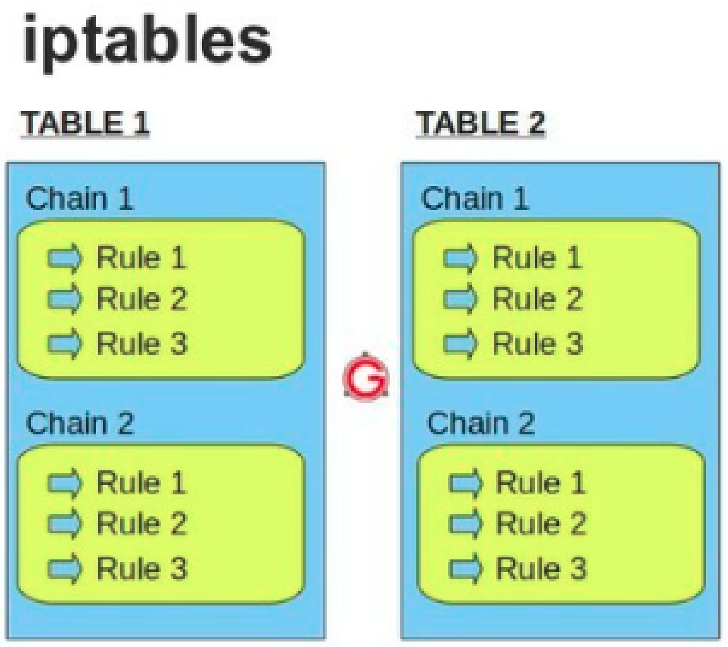


Figure 3: iptables

IPTables monitor the traffic from servers. This monitoring is done by using tables. A table contains a set of rules. This set of rules is called chains which filter incoming and outgoing packets. When a packet is received it is matched against a set of rules and then decision is taken on what to do with it.
The connection is processed sequentially when the kube proxy runs in iptables mode. This leads to the complexity of O(n). n grows in proportion to the number of services and backend pods.

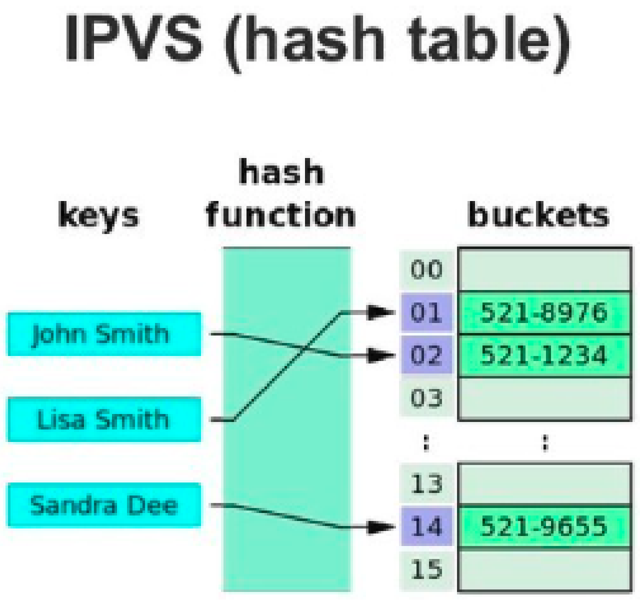


Figure 4: ipvs

2.2.2 IPVS

In ipvs mode a hash table is used by the kernel in order to establish the destination of the packet. As a hash table is used for processing, the complexity is O(1). So the connection performance stays independent of the cluster size.

2.3 CNI plugins and eBPF

CNI plugins are required to enforce specific network policies. Calico is one of the CNI plugins. Some of the CNI plugins are flannel, weave and canal.Calico can operate in three different dataplanes: Standard Linux, Windows HNS and eBPF. eBPF is one of the newer additions to calico.

eBPF stands for extended Berkeley Packet Filter. We can load some small programs using eBPF, which could be attached to certain hooks. These hooks are triggered whenever a specific event occurs. Some of the functionalities that eBPF provides us are tracing and traffic control. It also claims it has first and subsequent packet latency and a better throughput.

3 Tools Used

Tools we used in this project were:-

- Kubernetes :- Kubernetes is an open-source container orchestration system for automating software deployment, scaling, and management.
- Minikube :- Minikube is a lightweight Kubernetes implementation that creates a VM on your local machine and deploys a simple cluster containing only one node.
- kubectl :- kubectl is a Kubernetes command-line tool. kubectl allows to run commands against Kubernetes clusters.
- Calico :- Calico implements the Kubernetes Container Network Interface (CNI) as a plug-in and provides agents for Kubernetes to provide networking for containers and pods.

4 Design and Implementation

The main goal of our project was to compare the performance difference between using different networking models available in Kubernetes. We mainly focused on comparing the performance difference between the IPTable and IPVS mode in kube-proxy and the eBPF data plane in the Calico CNI plugin. For the performance comparison, we compared the throughput using the different modes and did tests for comparing the RTT.

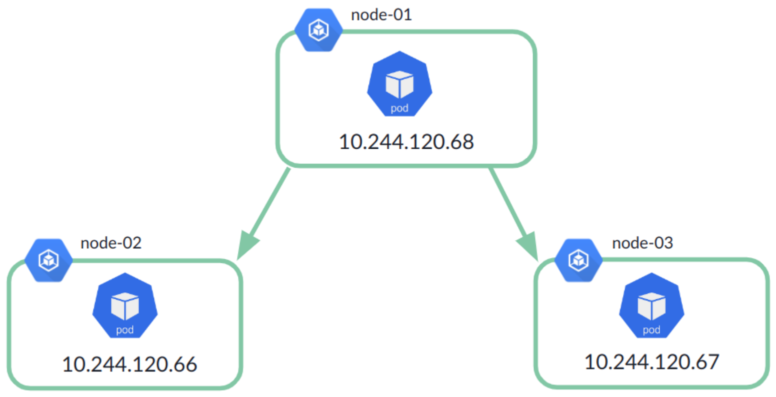


Figure 5: Cluster Architecture for Testing

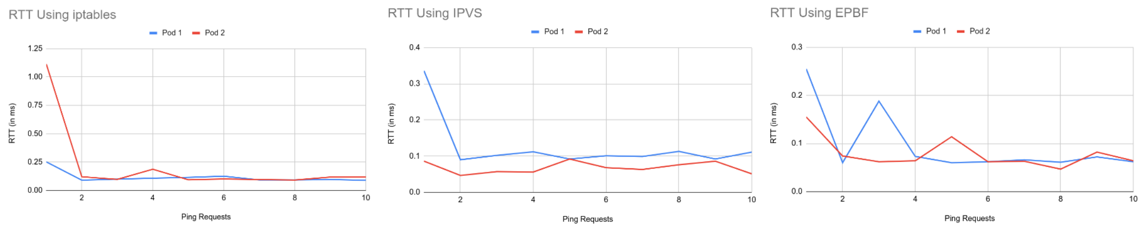
The architecture used for testing the RTT times can be seen in Figure 5. We have three nodes running on our cluster. Each node will have a single DaemonSet running with an IP address assigned to it. We can send requests to other pods using the ping command and measure the RTT times.

Similarly for testing the throughput times, we used the methodology mentioned by IBM for investigating network performance here. For testing the throughput, only two Daemon-Sets were used. Using the iperf tool, one was made client, and another server, and requests were sent for 120 seconds.

5 Evaluation Results

The results for our testing are hosted at the following GitHub Repository. For the RTT comparison, we noticed a significant improvement in the first packet RTT while using IPVS and eBPF mode.

After switching from IPTable to IPVS mode, the first packet RTT to Pod-2, dropped from 1.113 seconds to 0.092 ms. Although in IPVS mode, as it is visible in the average in Table 1, there is not a significant difference in the RTT times for subsequent packets and only a minor difference was observed in the average. This might be because in our case number of services were very less, and according to the Calico website significant difference only starts to occurs when number of pods $\geq 10,000$.



(a) RTT using IPTable (b) RTT using IPVS (c) RTT using eBPF

Figure 6: RTT Comparison

Doing the comparison between IPTable and eBPF mode, we again noticed an improvement in the first packet RTT as compared to while using the IPTable mode. This time the first-packet RTT was 0.115 ms to Pod-2. In contrast to while using IPVS, there was an improvement in subsequent packet RTTs too, as we can see in Table 1. The average RTT for eBPF is even less than 1ms for both the pods, but that is not the case for the other two modes.

Also, it is possible that gap in improvement is very less in our case, but as the number of services increase, the difference in improvement should also increase and will be much more noticeable.

Figure 7 contains the results for our throughput testing. As it is visible from log files and from the plots, we did not observe a significant difference in the average throughput. We observed an average of 250Mbytes/sec for IPTable, 249Mbytes/sec for IPVS and 245Mbytes/sec for eBPF data plane. We might not have much difference in the throughput

	Min	Average	Max
Pod 1	0.089	0.116	0.252
Pod 2	0.092	0.214	1.113

(a) RTT statistics for IPTable

	Min	Average	Max
Pod 1	0.090	0.124	0.336
Pod 2	0.046	0.068	0.092

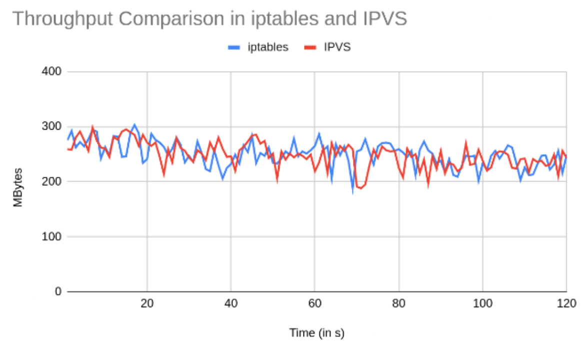
(b) RTT statistics for IPVS

	Min	Average	Max
Pod 1	0.060	0.095	0.255
Pod 2	0.046	0.074	0.115

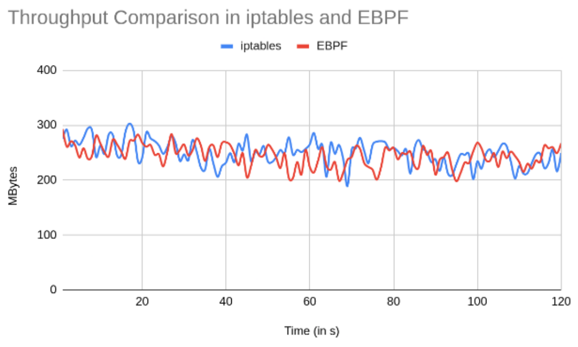
(c) RTT statistics for eBPF

Table 1: RTT Statistics

because significant performance difference only start to occur at higher load such 10GBit+ but the load was not that high in our testing environment.



(a) Throughput comparison between IPTable and IPVS



(b) Throughput comparison between IPTable and eBPF

Figure 7: Throughput Comparison

6 Summary

- The first packet RTT was better in eBPF and IPVS when compared to standard linux dataplane.
- The subsequent packet RTT is better in eBPF when compared with standard linux dataplane. However we did not observe any difference between IPVS and IPTables because the number of services were less than 1000 (10,000 pods).

- In case of throughput we did not observe any difference between IPTables, IPVS and eBPF because the load was not very high. Difference will be observed in case of high loads of approximately 10GBit+.

7 Takeaways

Following were the main takeaways of our project:-

- Learnt about various docker drivers and their use-cases.
- Hands-on experience with Kubernetes and Docker.
- Learned about IPTables and IPVS in Linux Networking, their internal working, and why they have different network processing complexities.
- We learned about different use cases for setting up network policies inside a cluster and how they are implemented using CNI plugins.
- Learned about eBPF and its benefits and use cases. Did performance analysis for different modes of IPTables, IPVS, and eBPF.

8 References

- Docker Networking Overview. https://docs.docker.com/network/
- Use bridge networks. https://docs.docker.com/network/bridge/
- Comparing kube-proxy modes: iptables or IPVS?. https://www.tigera.io/blog/comparing-kube-proxy-modes-iptables-or-ipvs/
- Calico Networking with eBPF. https://www.cncf.io/wp-content/uploads/2020/08/Calico-eBPF-Dataplane-CNCF-Webinar-Slides.pdf
- Introducing the Calico eBPF dataplane. https://www.tigera.io/blog/introducing-the-calico-ebpf-dataplane/
- Investigating Network Performance in an IBM Cloud Kubernetes Service Cluster. https://www.ibm.com/cloud/blog/investigating-network-performance-in-an-ibm-cloud-kubernetes-service-cluster