Packets in the Mesh

Free Sample · One Chapter

iptables versus eBPF Service Routing in Kubernetes

Md Nasim Sheikh

www.nasimstg.dev

About this sample

This is one complete chapter from *Packets in the Mesh*, a short field guide to the Kubernetes service data plane. The iptables sync-time figures here are measured by a reproducible harness; the eBPF and IPVS figures are cited, with a cluster recipe for reproducing them. The full book explains all three data planes and turns the comparison into a selection framework. The chapter below, "The Measured Comparison," reports the result.

4

The Measured Comparison

This chapter reports what can be measured directly and states plainly what cannot. The iptables sync cost is measured by the harness in the `bench/` directory. The eBPF and IPVS figures are cited from the projects' documentation and published sources, because reproducing them faithfully needs a multi-node cluster, which the appendix's recipe sets up for readers who want the full comparison.

4.1 Measured: iptables sync time grows with services

The harness builds a Kubernetes-style `nat` table for a growing number of services (three rules per service, the modest end of what kube-proxy emits) and times how long `iptables-restore` takes to install it atomically. That install is what kube-proxy does on every sync. Table 4.1 and Figure 4.1 show the result.

Table 4.1. Measured `iptables-restore` time to install the service table, by service count (AMD Ryzen 7 5700X, privileged container, 3 rules per service). Reproduce with the included harness.

Services	Rules installed	Sync time (s)
100	307	0.032
1,000	3,007	0.082
2,000	6,007	0.132
5,000	15,007	0.320
10,000	30,007	0.632

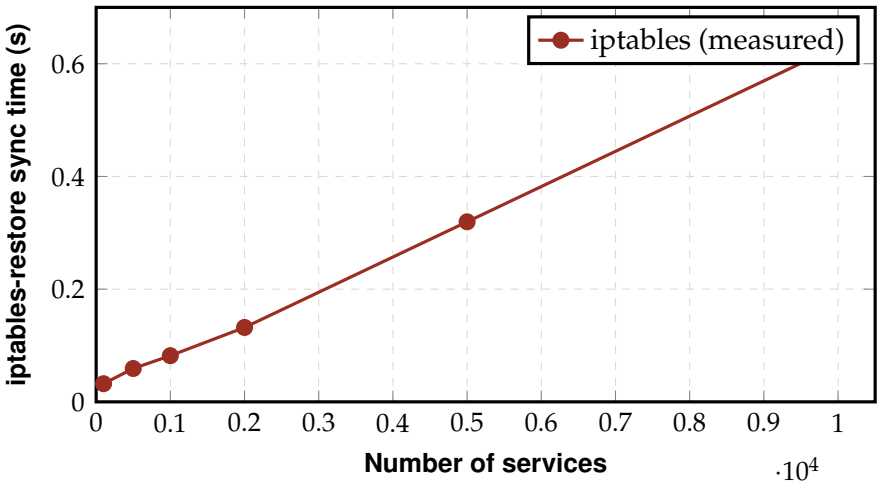

Figure 4.1. Measured iptables sync time against service count. The install cost rises roughly linearly with the number of services, and kube-proxy pays it on every change. A hash-based data plane's update cost is, by contrast, flat in the service count (cited below).

At ten thousand services the kernel needs about 0.63 seconds to install the table, and real kube-proxy, which emits more rules per service and reloads the whole table on each change, is slower still. This is the sync cost that operators feel as lagging endpoint updates on large clusters.

4.2 Cited: how the data planes compare

The data-path and update costs of IPVS and eBPF were not measured here. Table 4.2 summarises them from the projects' documentation and published benchmarks, and marks clearly which cells are cited rather than measured.

Table 4.2. Service data planes compared. The iptables sync column is measured (Figure 4.1); the remaining behaviour is cited.

	iptables	**IPVS**	**eBPF (Cilium)**
Data-path lookup	Linear scan, O(n) [1]	Hash, O(1) [1]	Hash map, O(1) [2]
Lookup latency	Grows with services [1]	Constant [1]	∼2–5 µs (cited) [2]
Sync on change	Full-table reload (measured)	Incremental [1]	Map update, flat [2]
Conntrack dependency	Yes [1]	Yes [1]	Reducible; supports DSR [2]
Kernel requirement	Universal	Module	Recent kernel [2][3]

What is measured and what is cited

Measured here: the iptables sync curve in Figure 4.1, on one machine, in a privileged container.

Cited, not measured: the IPVS and eBPF numbers, including Cilium's reported few-microsecond lookup and its constant-time updates. These need a real cluster; the appendix gives a `kind`+Cilium recipe so you can reproduce them yourself.

4.3 What the comparison shows

The measured curve confirms the mechanism of Chapter 2: iptables sync cost rises with the service count, because the whole table is rebuilt on every change. The cited figures place the other two data planes where their designs predict: IPVS removes the linear data-path scan with a hash but keeps the conntrack dependency, and eBPF removes both the linear scan and much of the per-change cost while adding features such as direct server return [2]. The gap is negligible on a small cluster and widens with scale, which is the theme the next chapter turns into a decision.

Get the full book

The complete *Packets in the Mesh* covers the iptables, IPVS, and eBPF data planes in full, the selection framework, and a protocol for measuring your own cluster, and ships the reproducible harness plus a kind+Cilium recipe.

www.nasimstg.dev

Open-source benchmark harness included.