Application Layer Processing Offload in the Kernel

Giulio Sidoretti
University of Rome Tor Vergata
CNIT

Sebastiano Miano
Queen Mary University of London

Stefano Salsano
University of Rome Tor Vergata
CNIT

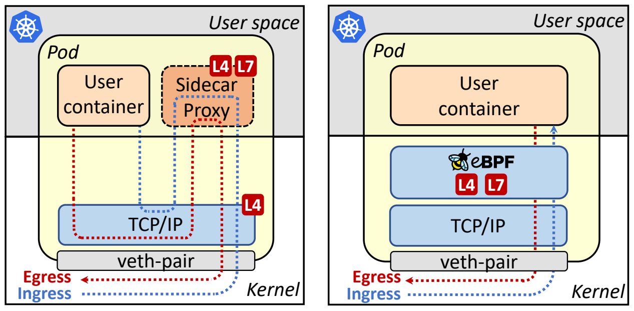
Gianni Antichi
Politecnico di Milano

Aurojit Panda
New York University

1 Introduction

Current service mesh deployments adopt a *sidecar* model, and associate a userspace proxy, such as Envoy, with each component (process or container). All network traffic to-or-from the container traverses the proxy, see Figure 1a.

The current implementations add significant overheads, that are mainly due to the additional kernel-to-userspace transitions [4]. Recent efforts, such as the Cillium Service Mesh [2] have proposed moving part of the processing of the proxies into the kernel. Layer 3 (L3) and Layer 4 (L4) policies are implemented in eBPF, but there is the need to fall back to the userspace sidecar for Layer 7 (L7) processing. We, instead, advocate for a full proxy offload (Figure 1b). L7 policies are of high importance in a sidecar proxy, considering for example the high amount of gRPC traffic in production datacenters [3]. Unfortunately, moving layer 7 functionality to the kernel is challenging. For example, gRPC vectors are unbounded and hence hard to deserialize.



(a) Standard **(b) Our Approach**

Figure 1: Networking processing involved for ingress (blue line) and egress (red line) packets in a service mesh. In the standard case (a), the sidecar proxy is in charge of the L4 and L7 processing. Our approach (b) is to completely remove the sidecar proxy and offloading L4 and L7 functionality in the kernel using eBPF.

2 Challenges

eBPF is a technology to load and run programs in a sandboxed environment in the kernel space, without having to modify the kernel itself. eBPF programs have limited complexity. Some use cases lead to increasingly complex programs that may not be implementable. We might extend the **capabilities** of eBPF, i.e. by incorporating a kernel module for gRPC serialized data processing which provides an interface to eBPF. It is a design choice that depends on the use case. There is a tradeoff between using a highly complex eBPF program versus modifying the kernel itself.

A client-server architecture, such as that used by gRPC, typically involves only two parties. When a userspace proxy, such as Envoy, is used, the proxy simply terminates the client connection and opens a new one to the server. With eBPF, this architecture has to be further questioned. Specifically, the eBPF proxy is attached to a socket hook, which means that a listening socket must be present on the machine, waiting for client requests.

3 Benchmark for the eBPF Proxy

We have deployed a testbed to measure the performances of an eBPF proxy in the context of gRPC monitoring. The eBPF program is attached to the *SK_SKB* hook, where we can access directly the content of the HTTP2/gRPC packet. Some information, such as the path of the called RPC, is written in the HTTP2 headers and is easier to parse. To access the data carried in the gRPC request itself, the program needs to deserialize the message. In our current implementation, this is done directly in eBPF, with a function that can process only a specific RPC, but in the future we need to generalize the process, eventually introducing a kernel module. We conducted our benchmark using ghz [1]. The performance of the eBPF proxy was compared with a NGINX gRPC proxy. When using eBPF, the latency for every gRPC request went from 1.58ms (NGINX) to 1.03ms, with a reduction of 34.8%.

References

[1] Bojan D. 2023. ghz. ghz.sh. [Online; accessed 09-03-23].
[2] Thomas Graf. 2022. Cilium Service Mesh. https://isovalent.com/blog/post/cilium-service-mesh/. [Online; accessed 09-March-2023].
[3] Svilen Kanev et al. 2015. Profiling a warehouse-scale computer. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*. 158–169.
[4] Xiangfeng Zhu et al. 2022. Dissecting Service Mesh Overheads. *arXiv preprint arXiv:2207.00592* (2022).