Optimizing Envoy's Network Stack with Kmap

Joshua Levin
Brown University

Peter Cho
Brown University

1 Introduction

Hey, dummy citation [1]

2 Background

3 Kmap

Our design for Kmap focus on avoiding costly, unnecessary invocations of the network stack. Envoy prides itself on being application agnostic, which leads to the helpful ability to auto-inject Envoy into microservices. However, this abstract requires unnecessary invocations of the network stack. In Figure-1 we highlight in red all the calls to the network stack for the path of a single request. Then, in Figure-2 we show the reduction in those calls by using Kmap instead of the network stack for local data transfer.

Kmap works by using LD_PRELOAD to load augmented network calls before libc regular calls. Then, when Envoy and the microservice invoke read or write calls locally, rather than passing the data into the network stack, we use Kmap's shared buffers to efficiently transfer the data. Thus, we must load the shared library for both the Envoy sidecar and the microservices. The two critical challenges to realizing Kmap are:

1. Building a robust, efficient shared buffer *faster* than the network stack
2. Determining in a microservice-agnostic way which socket calls should use Kmap

3.1 LD Preload

3.2 Pipe options

Named Pipes:
Unnamed Pipes:
Shared Memory:

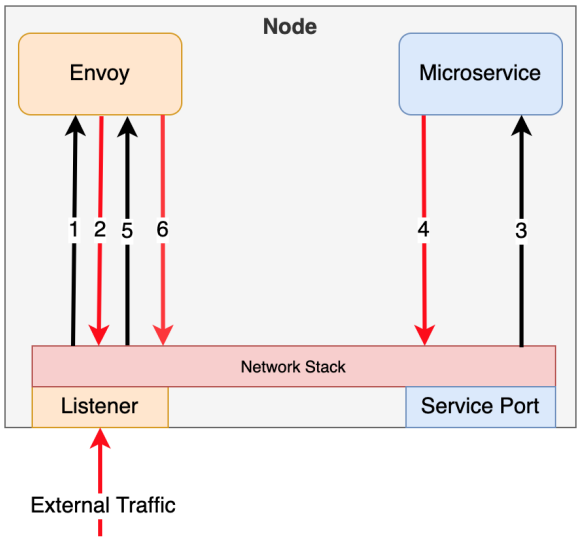

Figure 1: No Kmap Envoy Network Stack

3.3 When to apply Kmap

The network stack has a very well defined, robust API which applications use to communicate. A particular challenge for Kmap is pre-loading in front of those network calls and knowing when to pass through the call, or when to route to the local buffer. Since Kmap is designing primarily for Envoy, we use information about how it communicates with microservices to determine which file descriptors should use Kmap. This approach is not directly applicable to other sidecars (i.e. Linkerd) but is generalizable for applications, provided the applications work with Envoy first.

4 Prototype

References

[1] McCaffrey, C. Distributed sagas: A protocol for coordinating microservices.

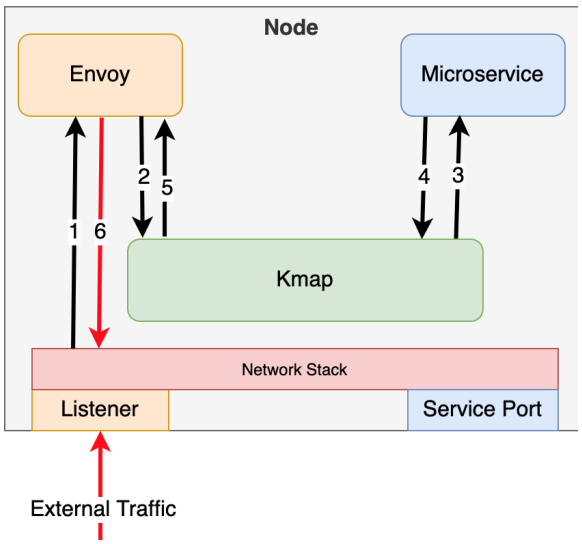

Figure 2: CPU Overhead