MOSN 高性能网络扩展实践

王发康

2021 Gopher Meetup HZ

About Me

王发康

蚂蚁集团 可信原生技术部，技术专家

蚂蚁集团技术专家，专注于高性能网络服务器研发，MOSN、Tengine 开源项目核心成员，目前关注云原生 ServiceMesh、Nginx、Envoy、Istio 等相关领域。

喜欢开源，乐于分享。

https://github.com/wangfakang

目 录

MoE 背景介绍

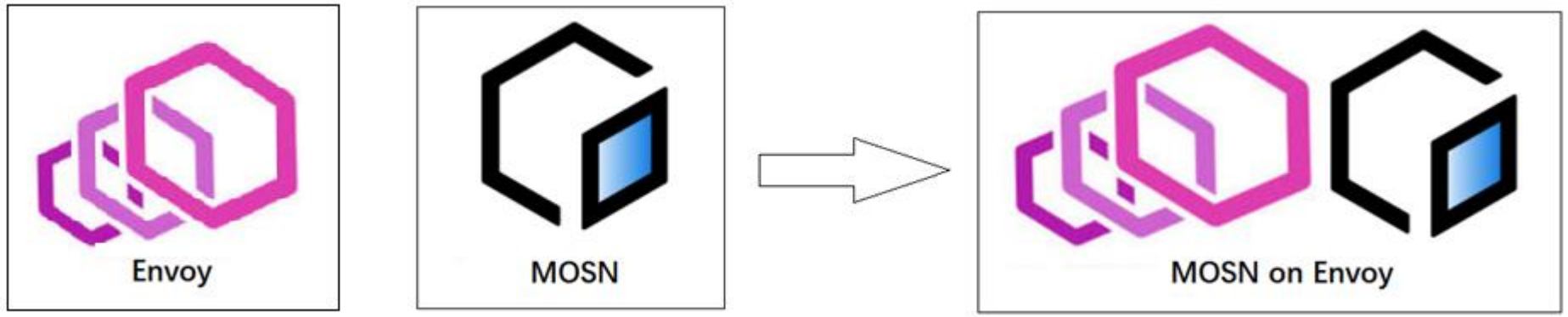
- MoE 是什么
- 为什么做 MoE
- 方案调研与分析

MoE 背景介绍 — 什么是 MoE

处理性能高
（C++）

研发效能高
(GoLang、生态)

高性能、高研发效能、生态打通

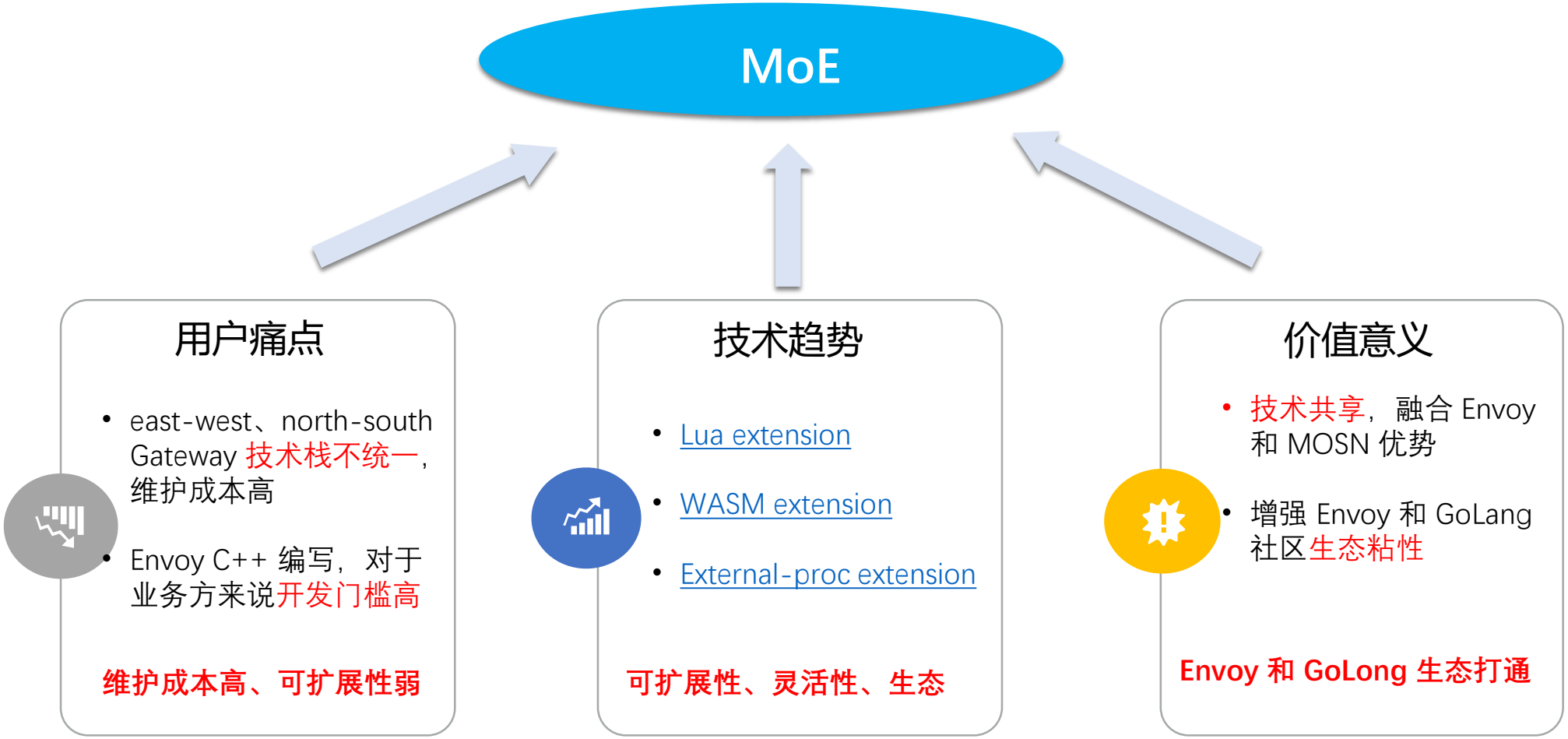


在 Service Mesh 领域，Envoy 和 MOSN 作为其数据面 sidecar 之一，用于解决传统服务治理体系下的痛点如：多语言，中间件组件开发适配成本高、SDK 升级困难、技术复用度差、治理体系不统一等。

MoE = MOSN + Envoy

相互融合，各取所长

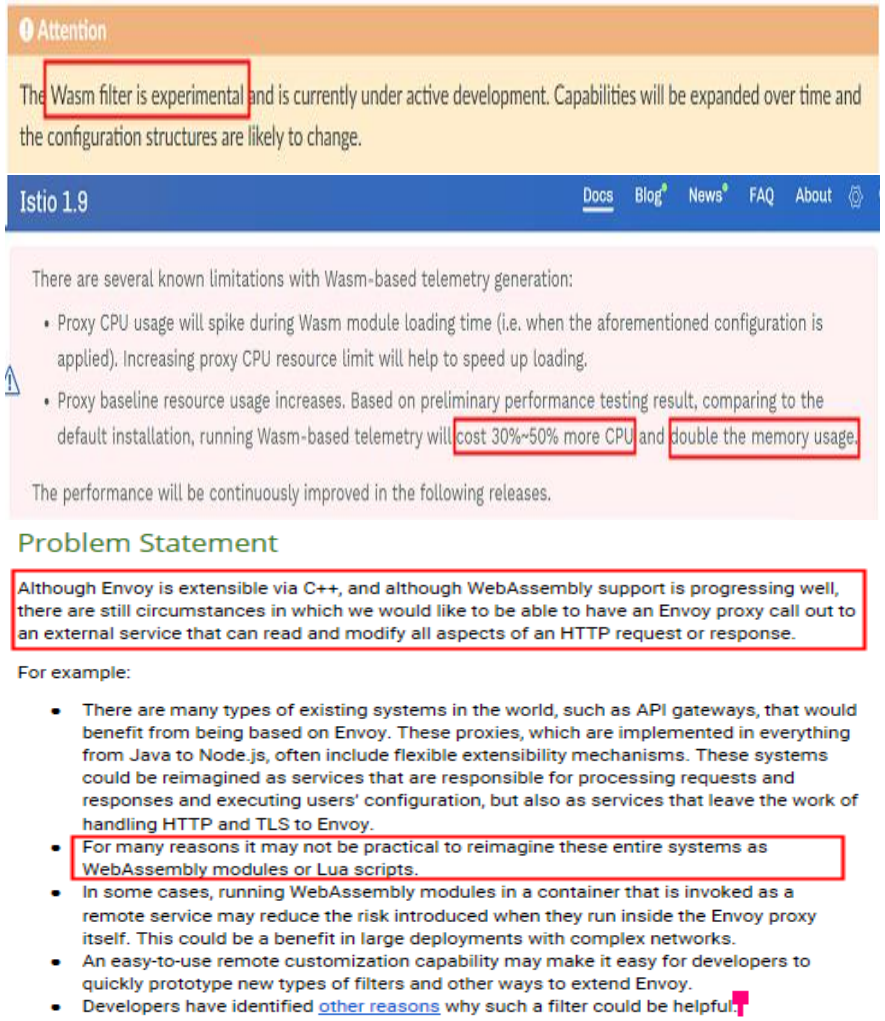
MoE 背景介绍 — 为什么做



MoE 背景介绍 — 方案调研

扩展方案调研

方案名称	优势	劣势
Lua Extension	Lua 编写简单业务处理方便	Lua 脚本语言,开发复杂功能不方便 支持的库（SDK）相对较少
WASM Extension	跨语言语言支持（C/C++/Rust）、隔离性、安全性、敏捷性	处于试验阶段，性能损耗较大; WASM 目前仅对C/C++/Rust 友好,对 GoLang Runtime 还未完全支持; 不能复用已有的 SDK，需要做网络 IO 适配改造
External-Proc Extension	跨语言支持、隔离性	需要跨进程通信性能低（UDS vs CGO 1KB Latency 差 8 倍); 需要扩展具备 gRPC server 能力,多进程管理复杂
MOSN(GoLang) Extension	可复用 MOSN 现有的 filter 能力,改造成本低; 研发效率高，灵活性高; GoLang 支持的库比较多（Consul、Redis、Kafka etc），生态较好	引入 GoLang 扩展后,有一定性能损耗,业务场景可接受，另外有优化空间



Attention

The Wasm filter is experimental and is currently under active development. Capabilities will be expanded over time and the configuration structures are likely to change.

Istio 1.9 Docs Blog News FAQ About

There are several known limitations with Wasm-based telemetry generation:

- Proxy CPU usage will spike during Wasm module loading time (i.e. when the aforementioned configuration is applied). Increasing proxy CPU resource limit will help to speed up loading.
- Proxy baseline resource usage increases. Based on preliminary performance testing result, comparing to the default installation, running Wasm-based telemetry will cost 30%~50% more CPU and double the memory usage.

The performance will be continuously improved in the following releases.

Problem Statement

Although Envoy is extensible via C++, and although WebAssembly support is progressing well, there are still circumstances in which we would like to be able to have an Envoy proxy call out to an external service that can read and modify all aspects of an HTTP request or response.

For example:

- There are many types of existing systems in the world, such as API gateways, that would benefit from being based on Envoy. These proxies, which are implemented in everything from Java to Node.js, often include flexible extensibility mechanisms. These systems could be reimagined as services that are responsible for processing requests and responses and executing users' configuration, but also as services that leave the work of handling HTTP and TLS to Envoy.
- For many reasons it may not be practical to reimagine these entire systems as WebAssembly modules or Lua scripts.
- In some cases, running WebAssembly modules in a container that is invoked as a remote service may reduce the risk introduced when they run inside the Envoy proxy itself. This could be a benefit in large deployments with complex networks.
- An easy-to-use remote customization capability may make it easy for developers to quickly prototype new types of filters and other ways to extend Envoy.
- Developers have identified other reasons why such a filter could be helpful.

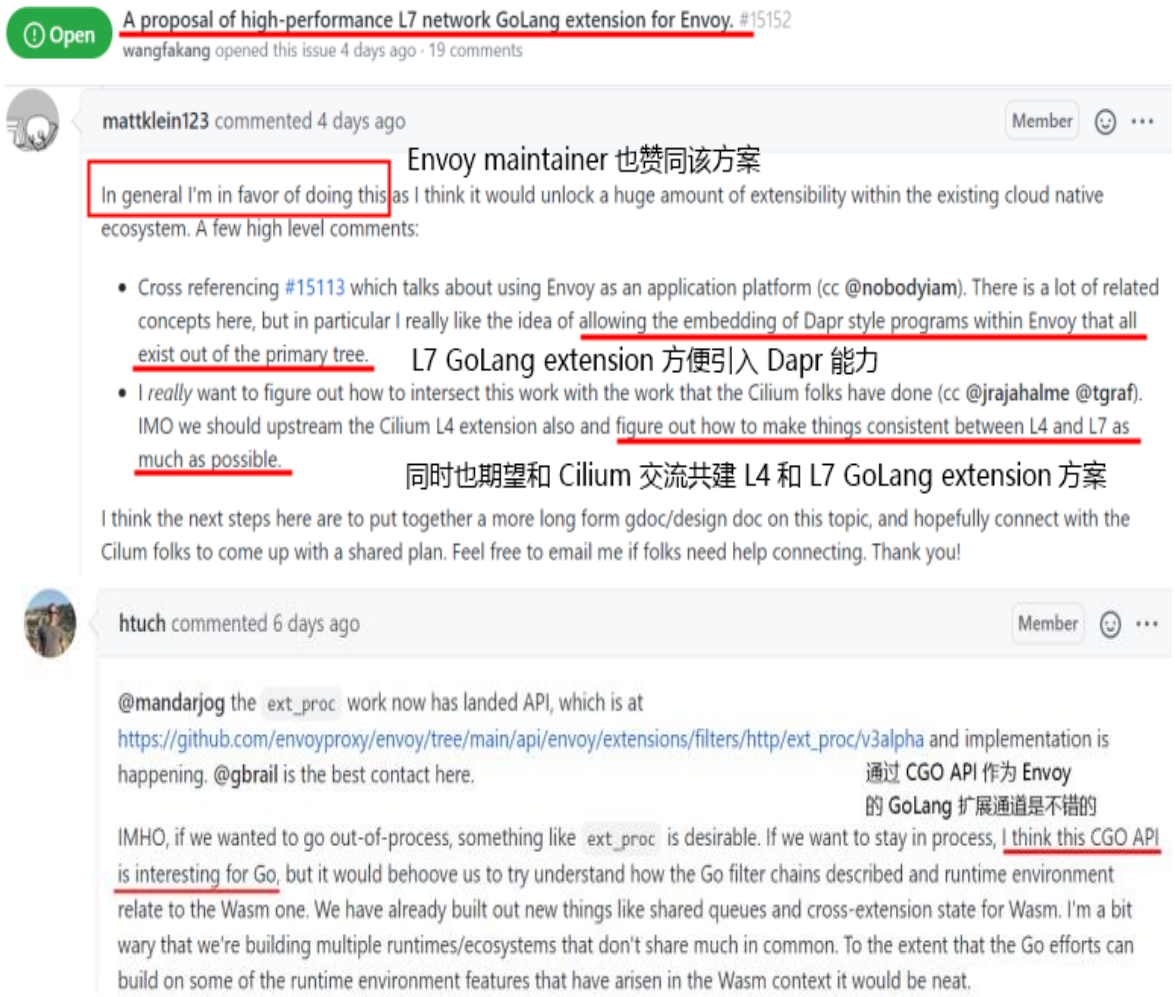
MoE 背景介绍 — 方案分析

扩展方案评估

方案名称	稳定性	性能	成本	生态
Lua Extension	高	高	高	较低
WASM Extension	ES	低	高	活跃
External Processing Filter	高	低	中	N
MOSN(GoLang) Extension	高	较高	低	活跃

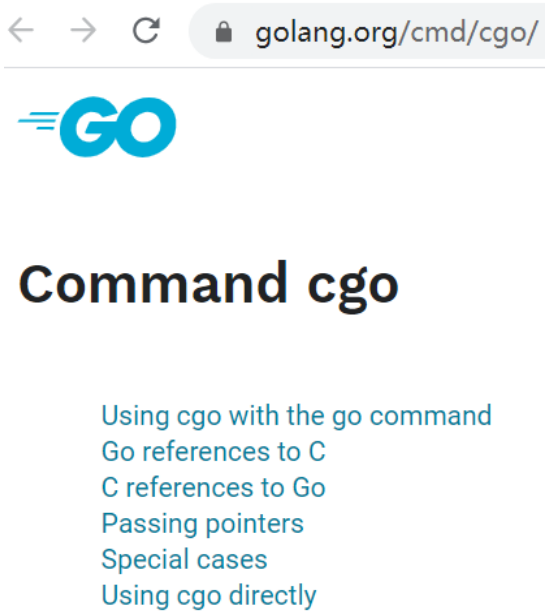
对比：MoE 相比 ext-proc 无需跨进程 gRPC，性能高；相比 WASM 无需网络 IO 操作转换成本；相比 Lua 生态好、能复用现有的 SDK，对于上层业务处理更合适

Envoy 社区讨论



MoE 背景介绍 — 方案分析

CGO 是 Go 官方出品



社区 CGO 维护频度

Go 1.16 Release Notes
Go 1.15 Release Notes
Go 1.13 Release Notes
Go 1.12 Release Notes
Go 1.11 Release Notes
Go 1.10 Release Notes
Go 1.9 Release Notes
Go 1.8 Release Notes
Go 1.7 Release Notes
Go 1.6 Release Notes
Go 1.5 Release Notes
Go 1.4 Release Notes
.....

用到 CGO 的一些项目

NanoVisor
Cilium
NginxUnit
Dragonboat
Badger
Go with OpenCV
etc

CGO 开销调研

CGO 一次调用开销在 0.08 ~ 1.626 us，另外 CGO 调用开销呈线性增长；CGO 中增加 Go 自身计算逻辑时,其 Go 的计算消耗也呈线性增长

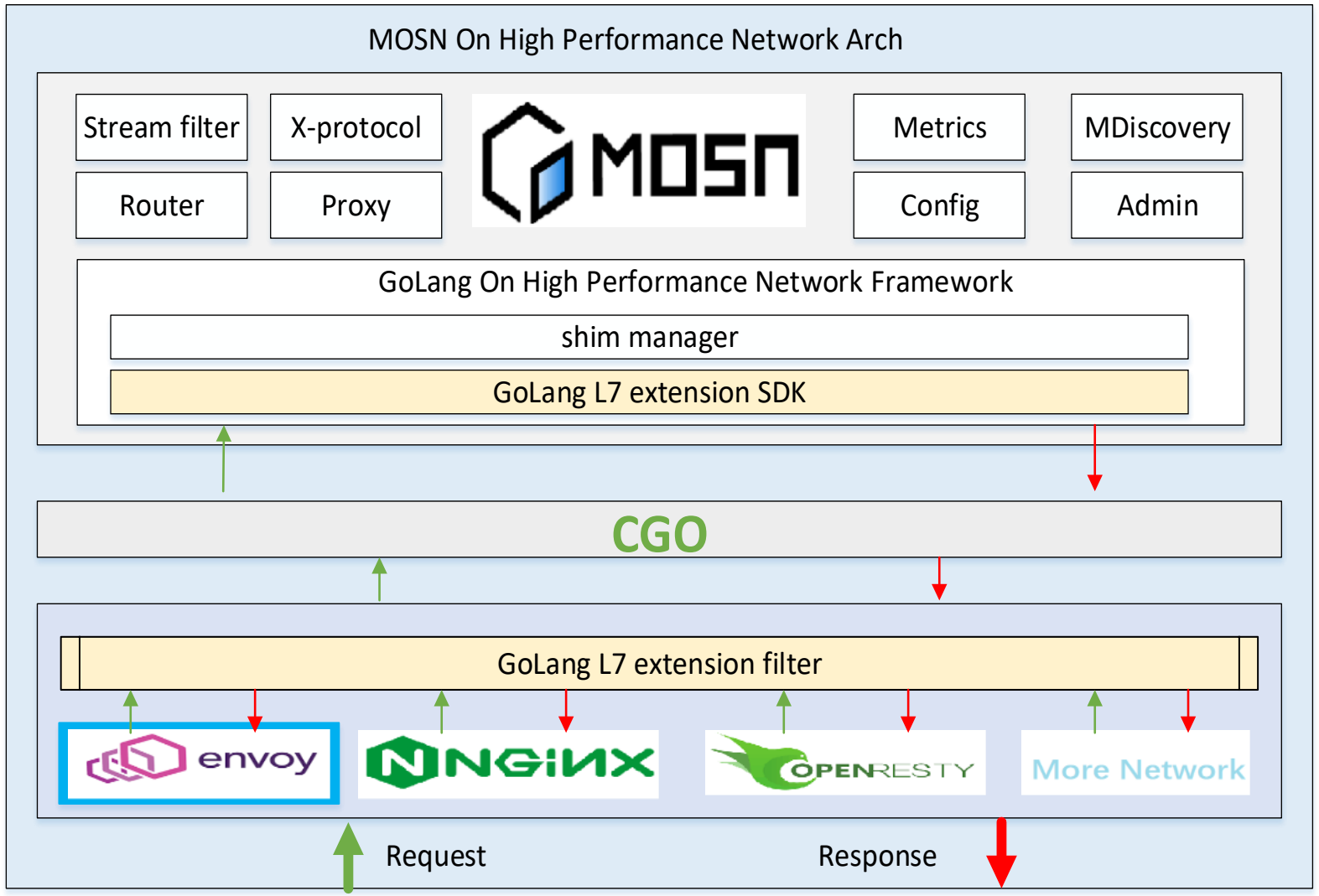
结论

综合稳定性、性能、成本、社区生态等因素评估，MoE 解决方案无论在当前阶段还是未来都具备一定优势

MoE 方案介绍

- MoE 整体架构
- MoE 功能职责
- MoE TraceID 事例分析
- MoE 方案优势

MoE 方案介绍 — 整体架构



同时，我们也会将其剥离出来形成一套标准的扩展: proxy_golang，类似 proxy_wasm，方便 Nginx、OpenResty 等也能够支持 GoLang 扩展。

proxy_golang API spec

proxy_golang_request

- **params**

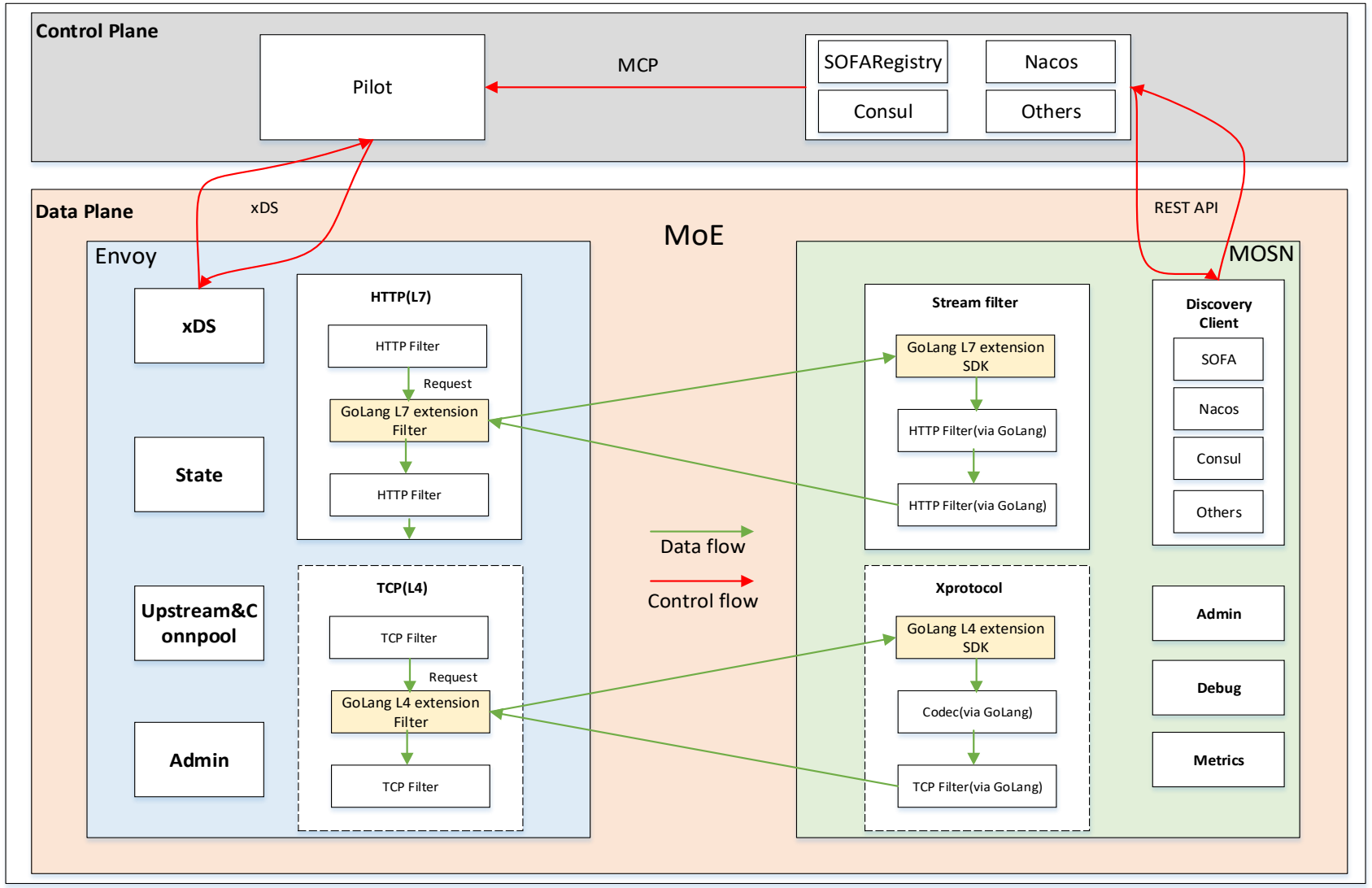
  headers
  body
  trailers

- **returns**

  C.Response{
  headers,
  body,
  trailers,
  optionFlags}

……

MoE 方案介绍 — 功能职责



MOSN 做业务扩展

- 扩展非 xDS 服务发现
- 扩展 L4/L7 filter
- 扩展 Xprotocol 支持
- Debug 及 Admin 管理
- Metrics 监控统计

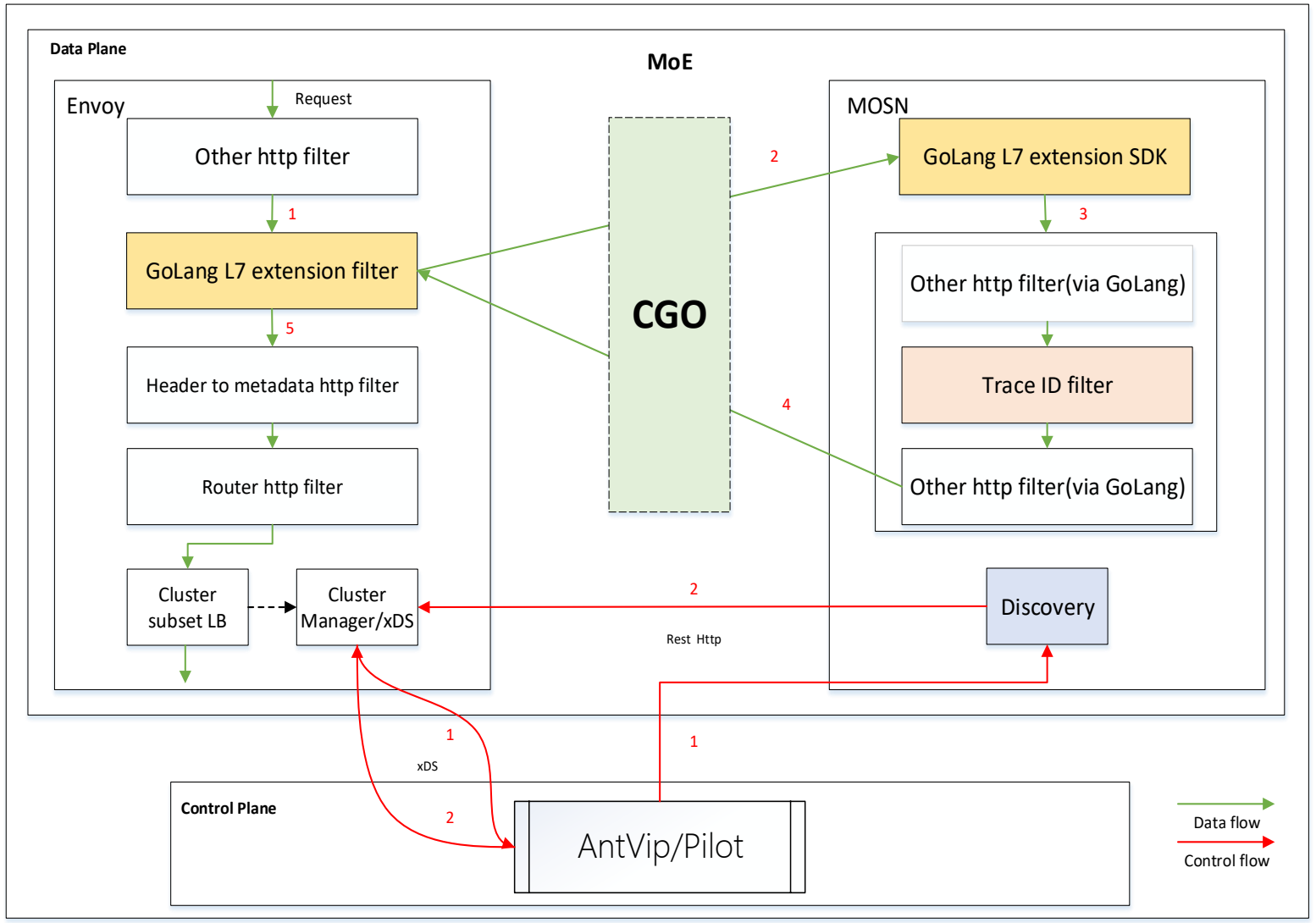
Envoy 复用基础能力

- 复用高效 Eventloop 模型
- 复用 xDS 服务元数据通道
- 复用 L4/L7 filter
- 复用 Cluster LB
- 复用 State 统计

Proxy-golang 扩展能力

- Proxy-golang API
- Filter manager

MoE 方案介绍 — TraceID 事例

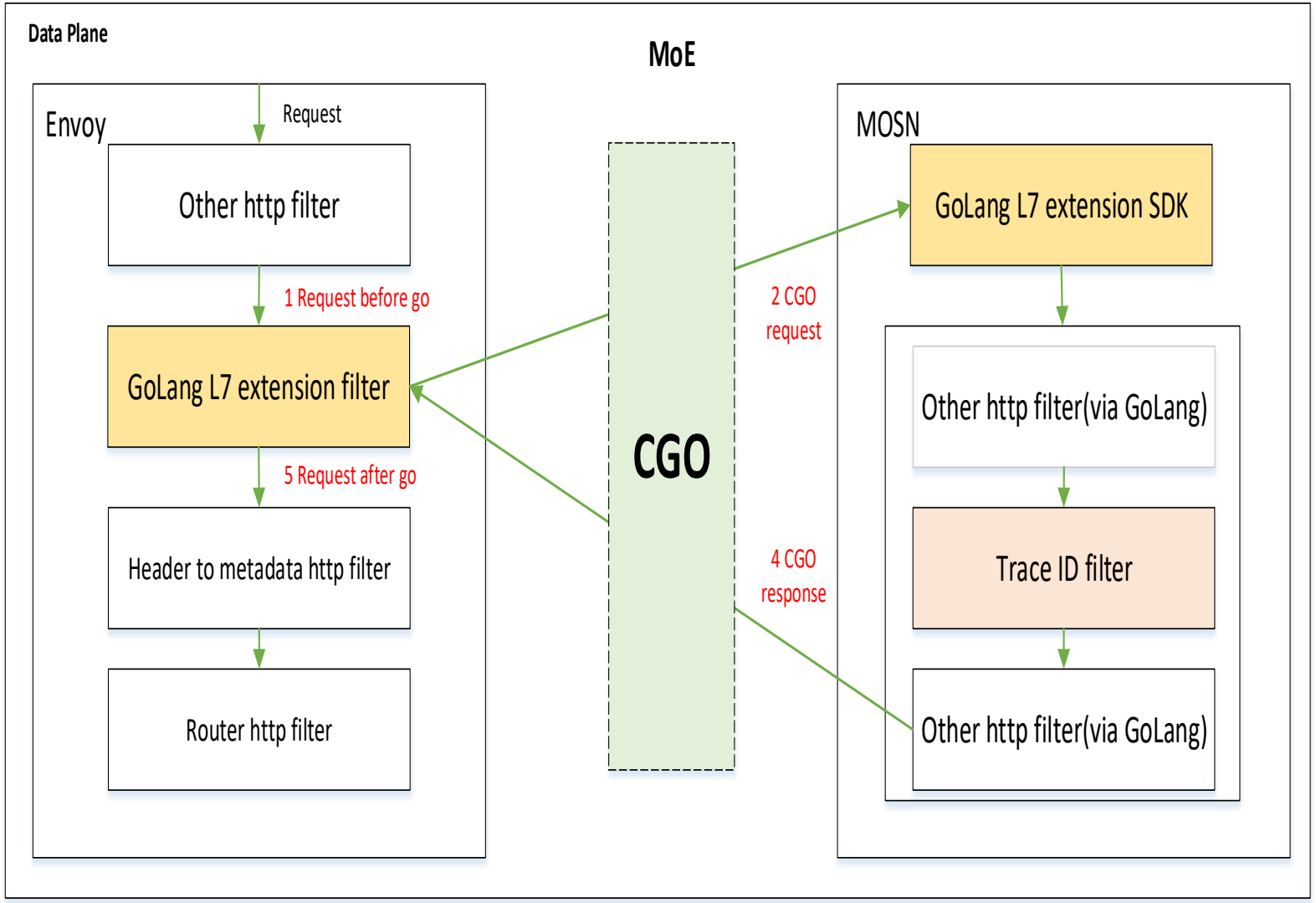


相关问题点

- Envoy 和 MOSN 如何交互(1、2、4、5)
- 内存如何管理(2、4)
- 阻塞操作处理(2)
- GMP 中 P 资源问题(3)
- 服务发现如何兼容（1、2）
- 如何 Debug
- recover 失效如何处理

……

MoE 方案介绍 — MOSN 和 Envoy如何交互



将 Envoy 的请求使用 **GoLang L7 extension filter** 的 proxy_golang API 进行封装，通过 CGO 通知 MOSN 侧 **GoLang L7 extension SDK** 获取处理

同 Envoy、Cilium、WASM 社区合作共建 API 规范：

proxy_golang API spec

proxy_golang_request

- **params**

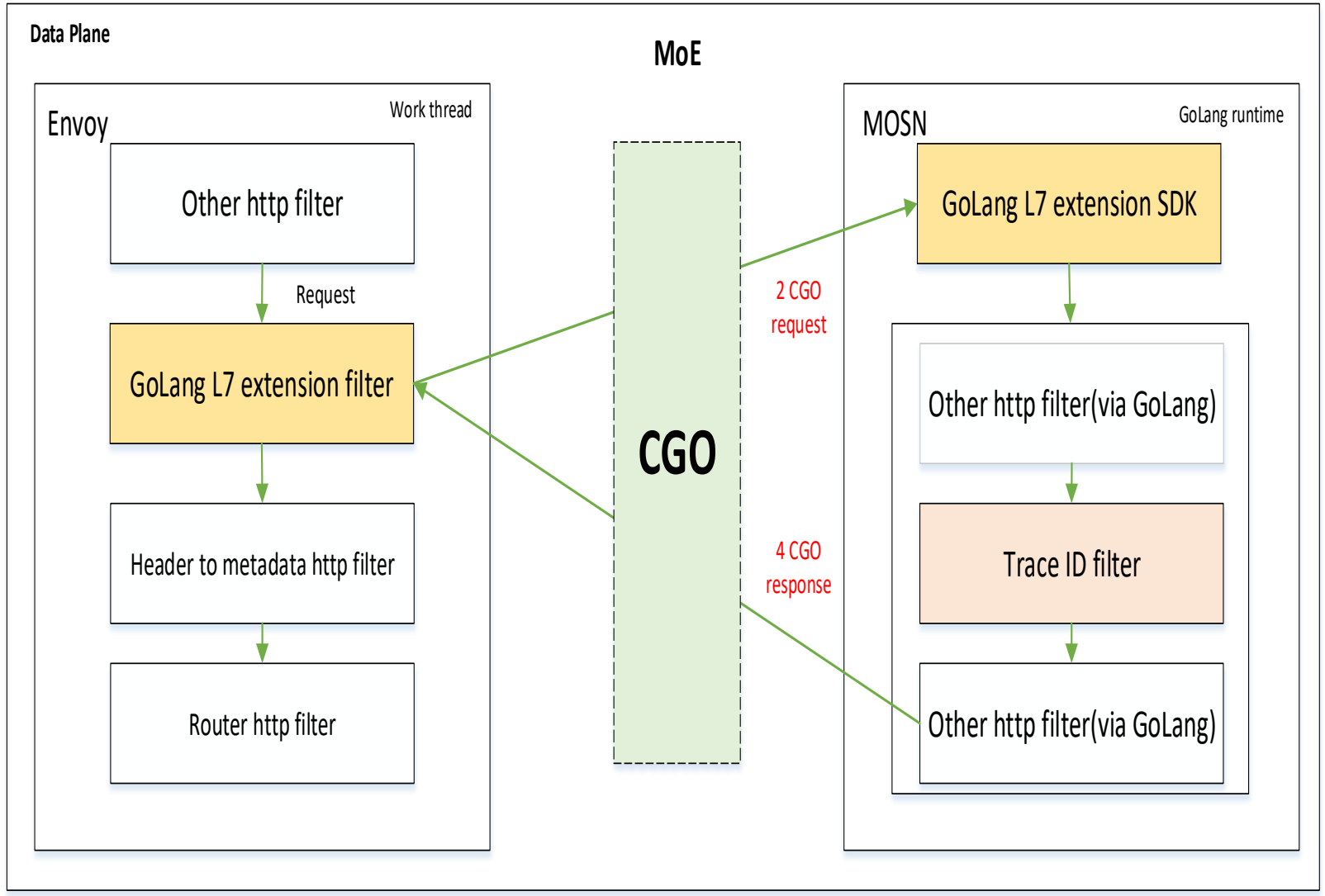
  headers
  body
  trailers

- **returns**

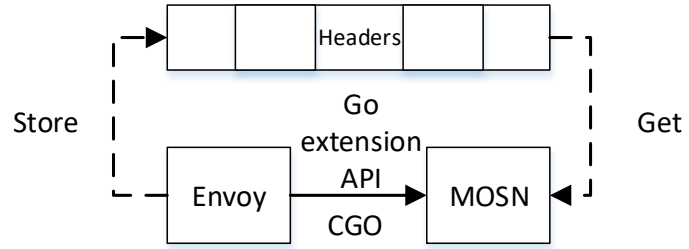
  C.Response{
  headers,
  body,
  trailers,
  optionFlags}

……

MoE 方案介绍 — MOSN 和 Envoy 内存如何管理

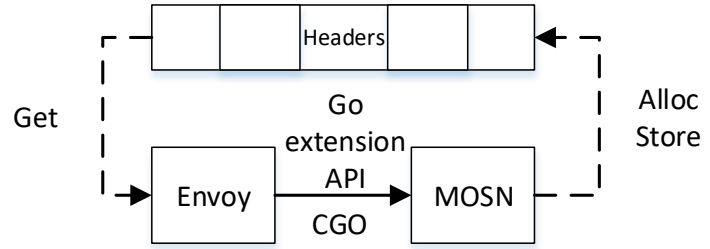


【请求链路】将 Envoy 中的请求信息拷贝一份传递给 MOSN？需要额外内存分配和拷贝？

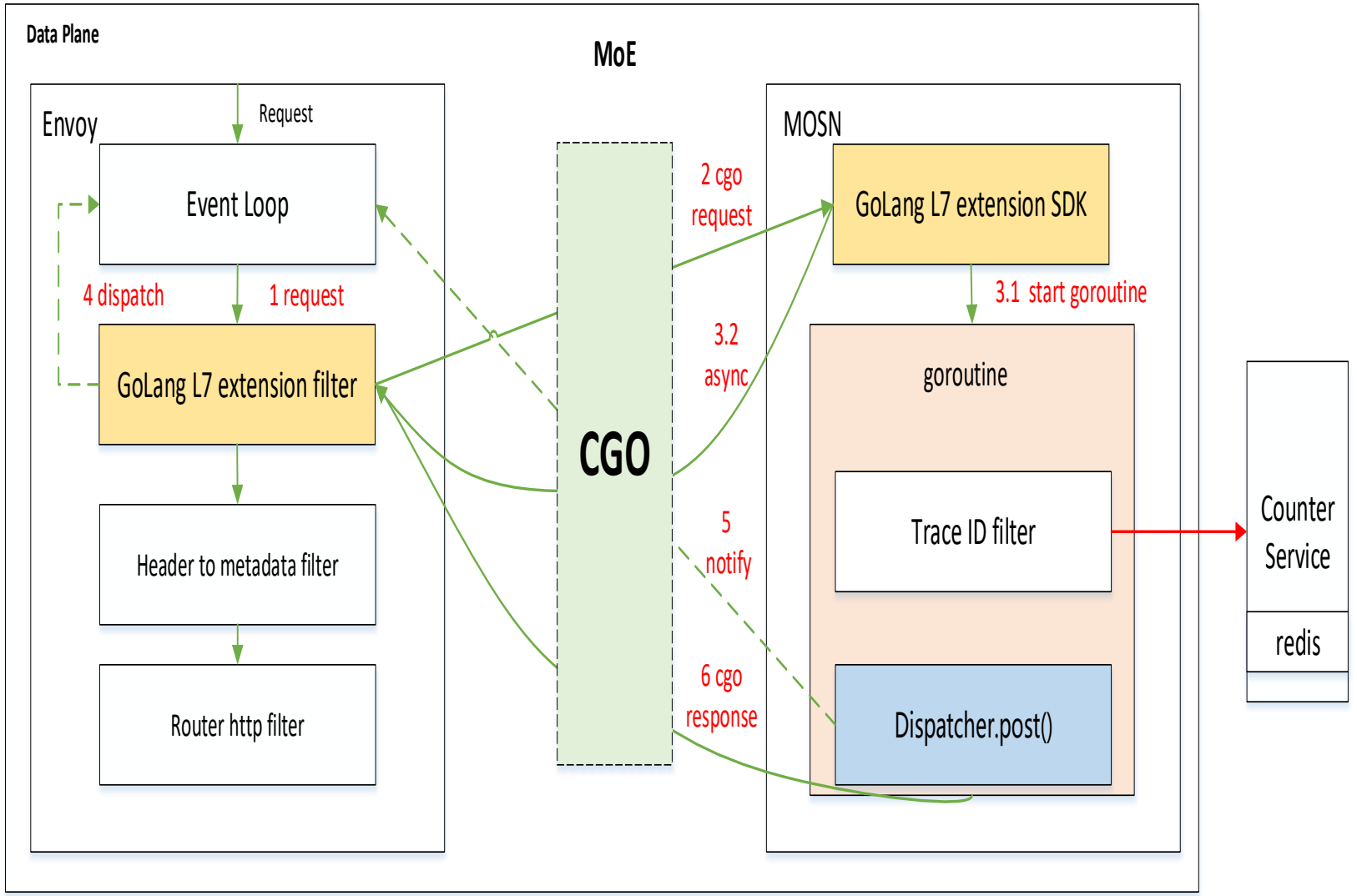


相比 WASM、Lua, MoE 内存 Zero Copy

【响应链路】请求到 MOSN 执行一些操作，其处理结果返回给 Envoy，Envoy 直接使用 GoLang 返回的内存安全吗？



MoE 方案介绍 — 阻塞操作如何处理



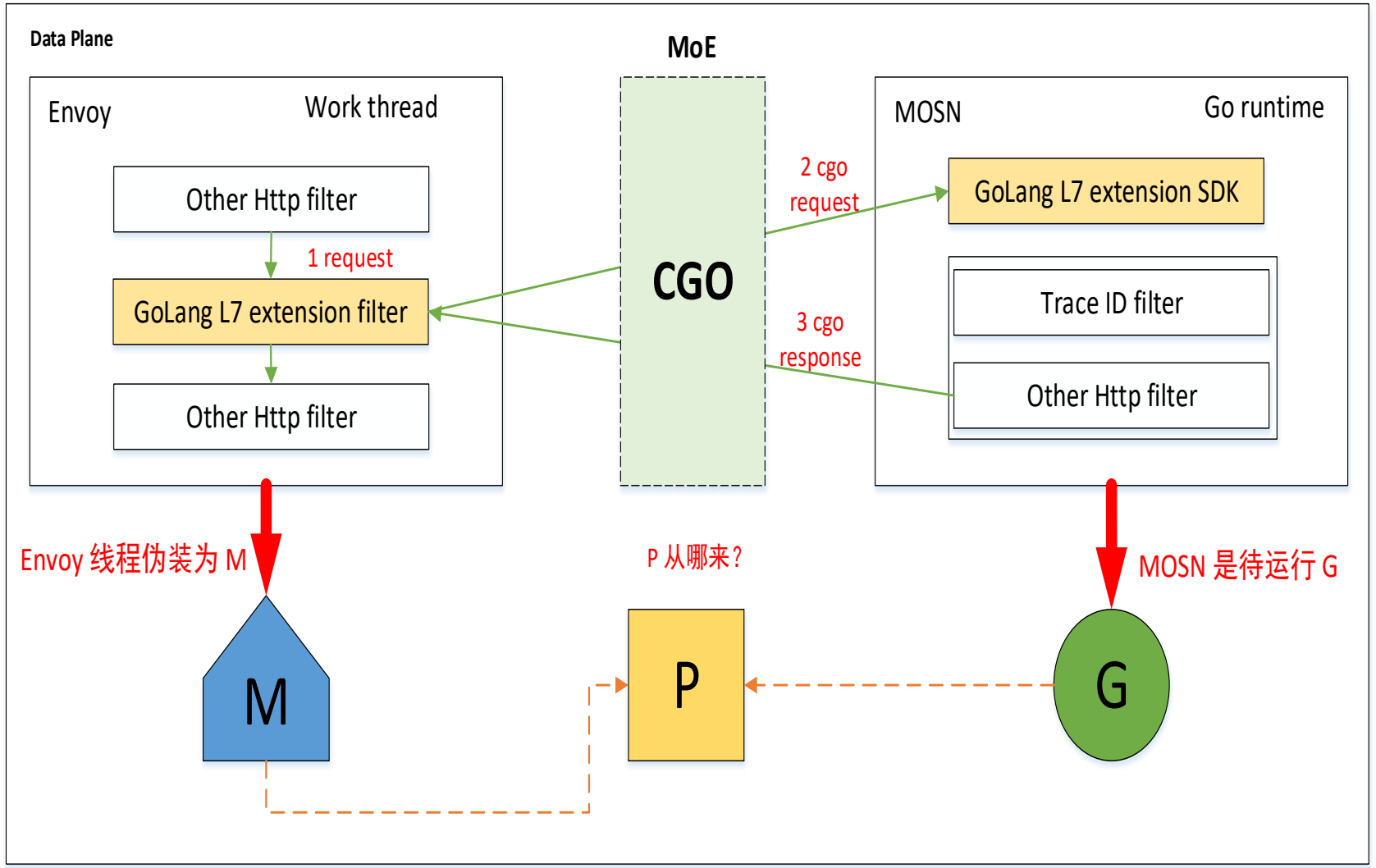
Envoy 是异步非阻塞事件模型，那 MOSN(GoLang) filter 中存在阻塞操作需要如何处理？

对于纯计算(非阻塞)或请求链路中的旁路阻塞操作，按照正常流程执行即可

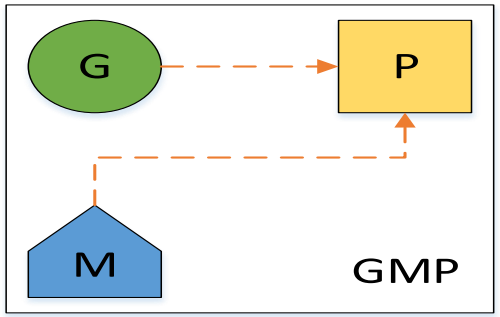
对于阻塞操作，通过 GoLang 的 groutine（协程） 结合 Envoy 的 event loop callback机制：

当 MOSN 中有阻塞操作则 Envoy 立刻返回，MOSN 启动一个协程，执行完阻塞操作后 notify Envoy

MoE 方案介绍 — GMP 中 P 资源问题

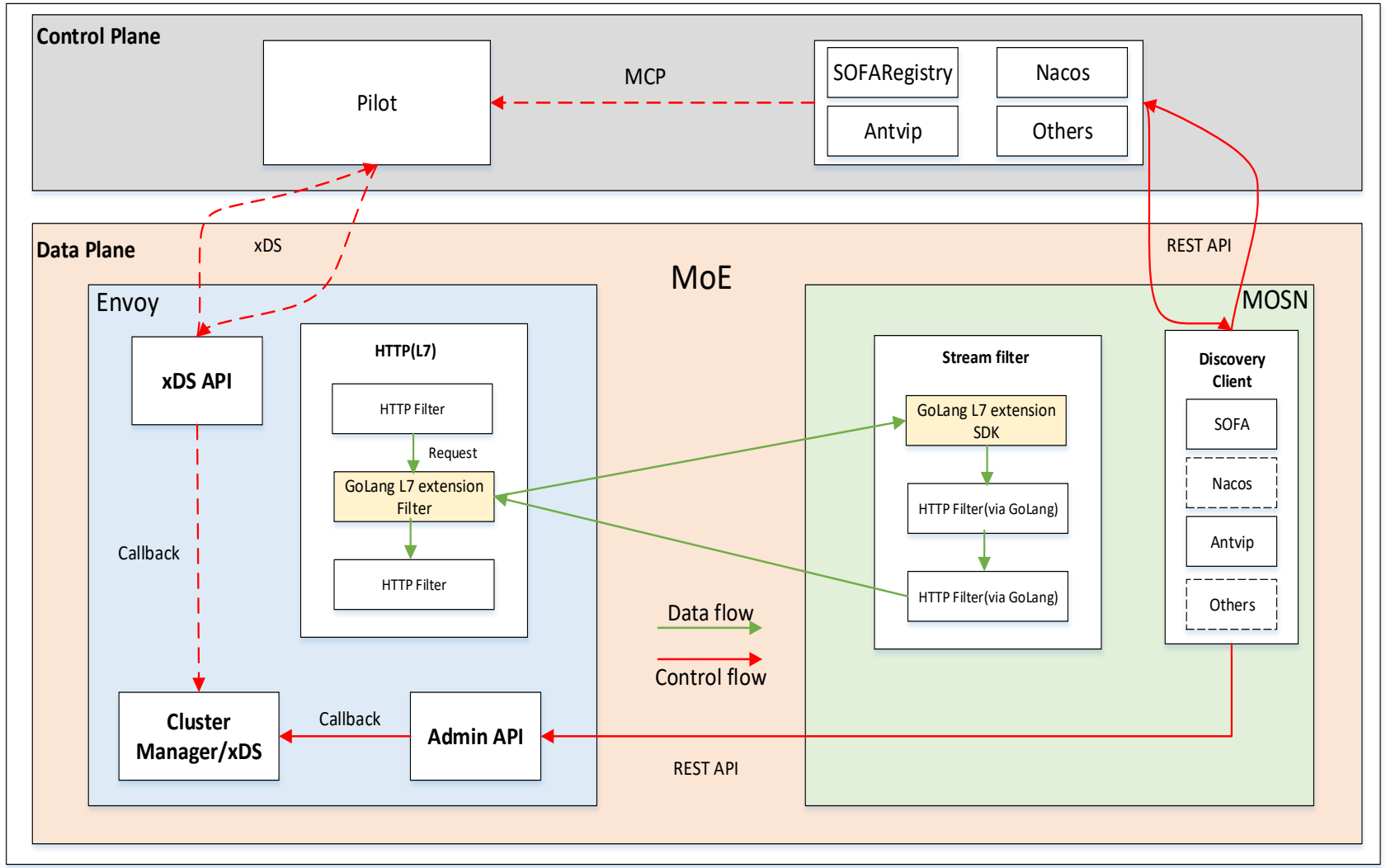


Envoy 通过 CGO 执行 MOSN(GoLang),此时 P 的数量如何管理？M 从哪来？



为 Envoy 每个 work thread 都预留对应的 P，保证每个 G 都可以立刻找到 P

MoE 方案介绍 — 服务相关元数据如何管理



MOSN 和 Envoy 的相关服务元数据信息，是如何交互管理的?

通过扩展 Envoy 中的 Admin API 使其支持 xDS 同等功能的 API, MOSN 集成的 Service Discovery 组件通过该 API(rest http) 和 Envoy 交互

使其 MoE 的服务发现能力也具备“双模”能力，可同时满足大规模及云原生的服务发现通道

MoE 方案介绍 — 相关采坑记录

GoLang 相关

- GoLang recover 失效
- GoLang 返回的字符串被截断
- export function type 关联出错
- 不同方式加载 CGO 程序，则 GoLang runtime 运行时机可能不一样
- CGO 交互内存生命周期管理

Envoy 相关

- 默认配置不不支持 HTTP1.0
- Envoy 时间模块使用的是 UTC
- upstream 支持 HTTP2 需要显示配置
- 访问日志换行需要自行配置 format 支持
- 异常场景下响应状态码不标准
- 各个 work 处理请求均衡性问题
- access_log handler 执行顺序不合理
- etc

MoE 方案介绍 — 如何 Debug

Envoy

- Admin API
- Debug log
- Request/Connection metrics

MOSN(GoLang)

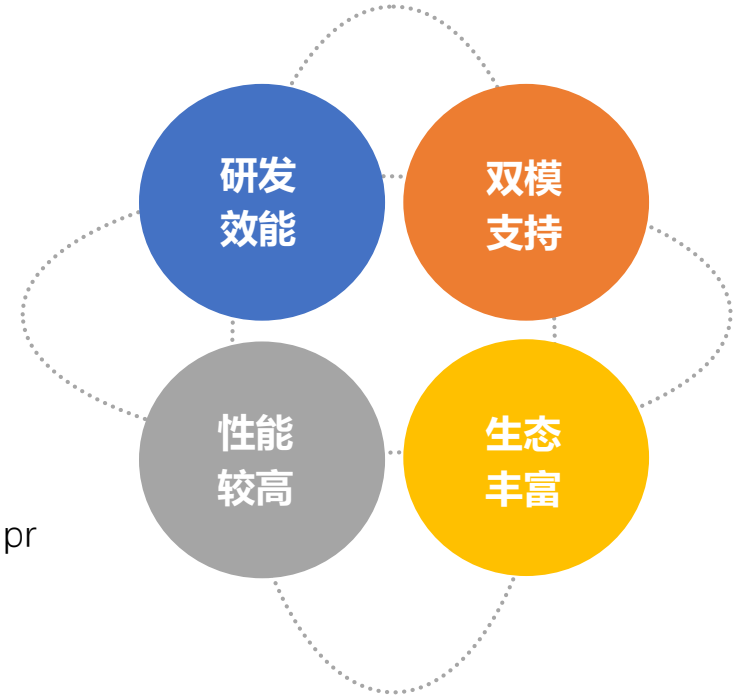
- Admin API
- Debug log
- GoLang runtime 指标

Envoy 和 MOSN 交互层

- MOSN（GoLang） 侧执行时间统计
- 交互异常数统计
- GoLang 程序异常场景下的容灾处理

MoE 方案介绍 — 方案总结

- 将 MOSN 作为 Envoy 动态 so，提升编译速度
- 增强 Envoy 扩展能力，复用 MOSN 现有的 filter 能力



- 同时具备云原生 xDS 、REST API服务元数据管理通道能力

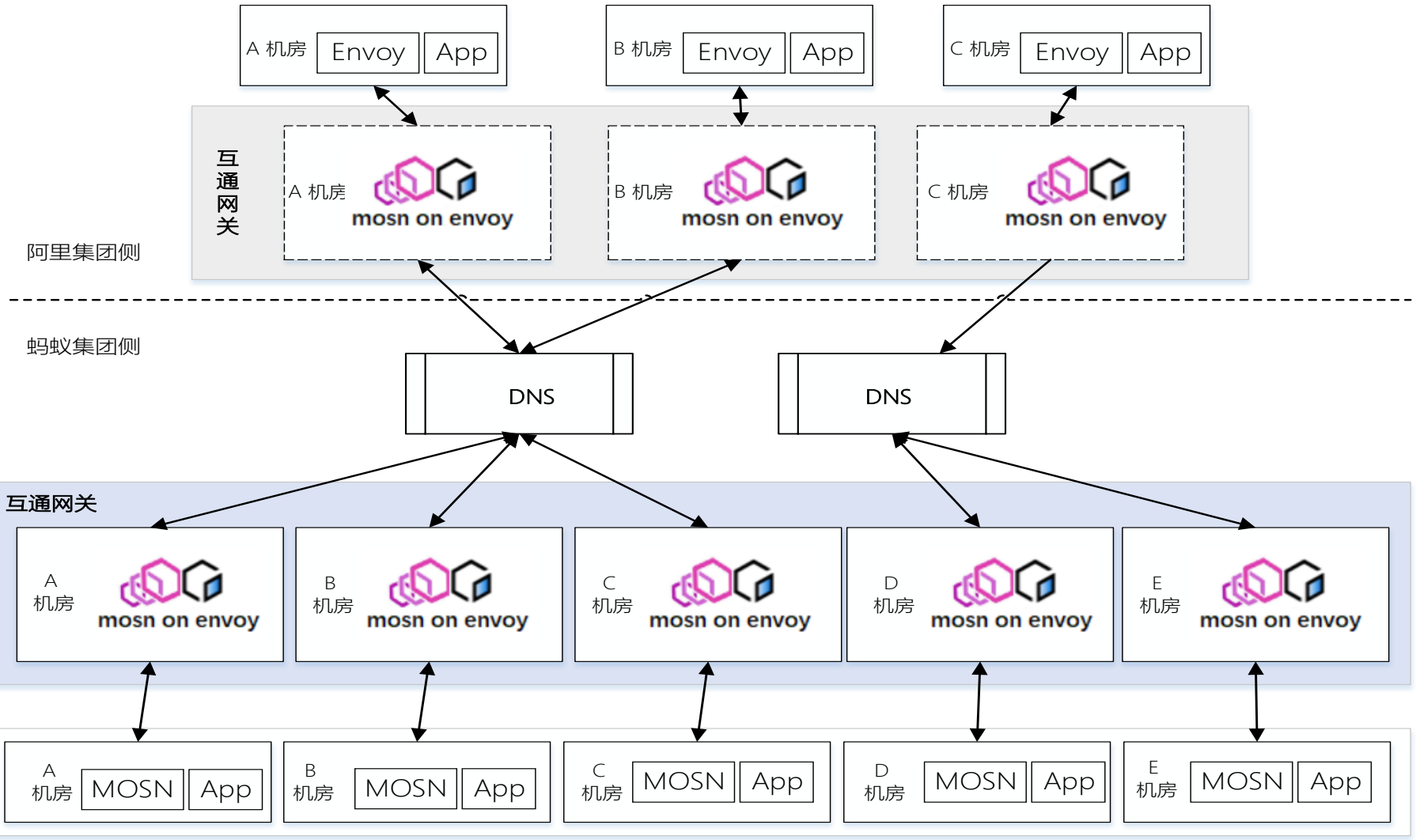
- 复用 Envoy 高效网络通道，如为 Dapr 能力提供底层 gRPC 通道
- 具备硬件加速集成能力
- 内存管理 Zero Copy

- MOSN/GoLang 和 Envoy 生态拉通
- 实现多个社区技术共享，增强 Service Mesh、Dapr 等领域的生态

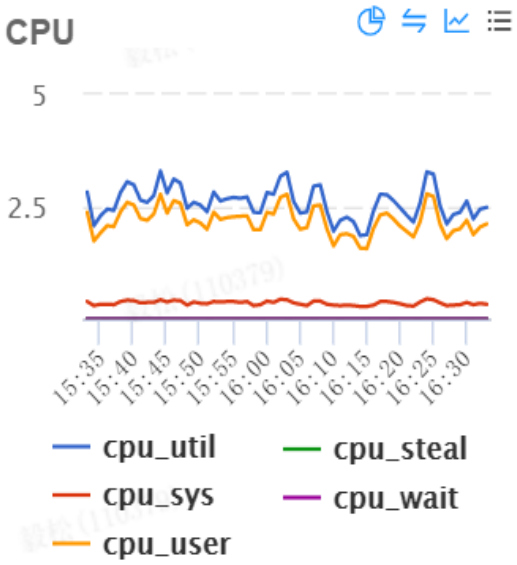
MoE 运行效果

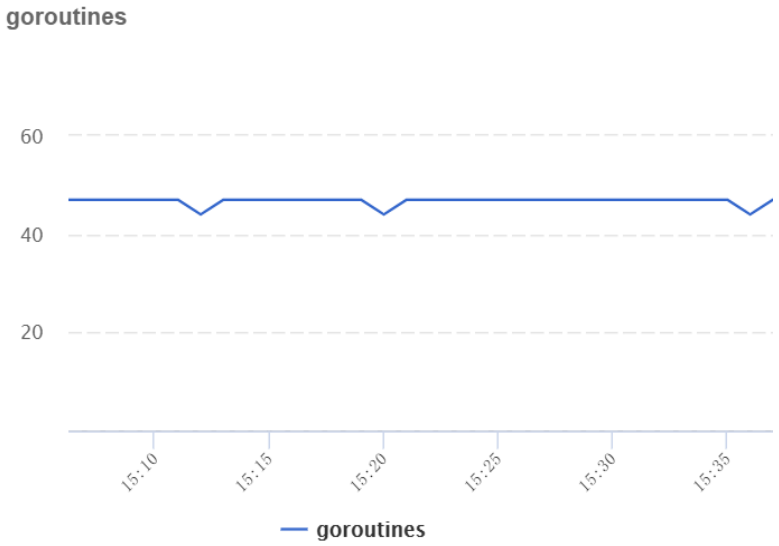
- MoE 部署架构
- MoE 为运行效果
- MoE 相关指标

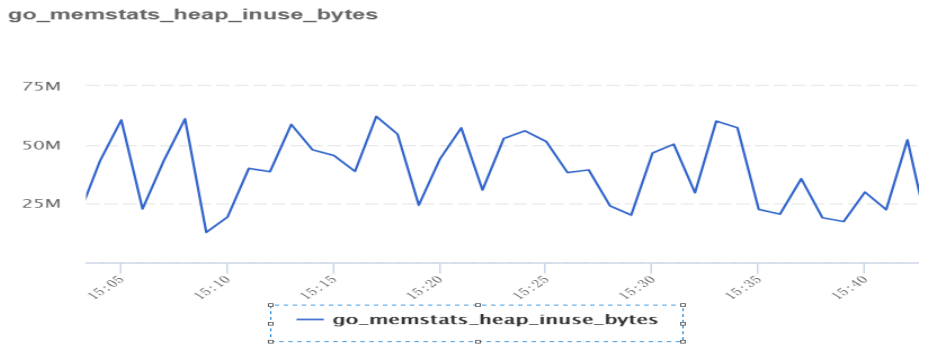
MoE 方案介绍 — 部署架构



MoE 方案介绍 — 运行效果







MoE 方案介绍 — 相关指标

经济体互通网关蚂蚁侧场景，当前灰度了少量的线上流量，已经平稳运行了 1 个月左右；

- MoE 中 MOSN RT 消耗在 0.05 ms 左右
- MoE 相比于 GoLang 自身 HTTP2 处理能力具有 4 倍左右性能提升
- MoE 相比于 Envoy 性能下降 20%，虽然牺牲部分性能，但解决了用户在其可扩展性、灵活性、生态上的痛点，另外对性能方面也有优化空间：

  - 业务代码优化，如减少对象数量
  - 内存管理优化，如 jemalloc 替换 tcmalloc、堆外内存
  - runtime 相关优化，如 cgocheck 调优、P 分组管理等
  - 交互协议优化，如减少 CGO 交互次数等

MoE 社区共建

- MoE Roadmap
- Q&A

MoE Roadmap

定位：云原生网络代理平台

理念：通用能力回馈社区，同社区共建标准

Q ? A : E

MOSN 官网 http://mosn.io
MOSN Github http://github.com/mosn/mosn

MOSN 开源交流群

欢迎技术交流

Thanks