MOSN 在云原生的探索和实践

王发康

2021 GopherChina

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，目前专注于云原生 ServiceMesh、Nginx、Envoy、Istio 等相关领域。

喜欢开源，乐于分享。

https://github.com/wangfakang

MOSN 开源交流群2

GopherChina 2021

目 录

MOSN
云原生演进历程

MOSN 简介 — 演进历程

MOSN 从 Service Mesh 技术调研，到产品孵化，历经重重困难，最终通过双 11 规模化验证。借力开源、反哺开源，进行 Cloud Native 生态融合，在实践的道路上一步步的走向云原生。

2018年3月

MOSN 诞生
支持 Service Mesh

2019年双11

核心支付链路覆盖

2019年12月

MOSN 宣布独立运营
CNCF landscape

2020年6月

V0.13.0 发布，
进行云原生组
件生态融合

2020年7月

Istio 官方推荐
数据面

2020年12月

商业化落地
江西农信 Mesh
阿里云 CDN

2021年

MOSN 和 Envoy、
Dapr、WASM
开始展开生态合作

SOFASTACK
Istio
Dubbo
Skywalking
Sentinel

MOSN 简介 — 开源社区

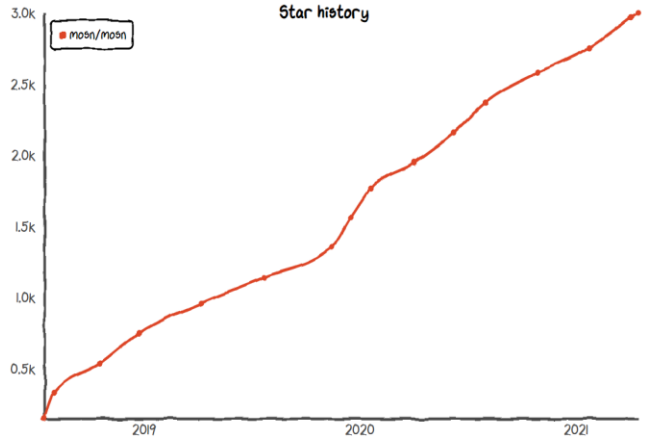
定位：云原生网络代理平台

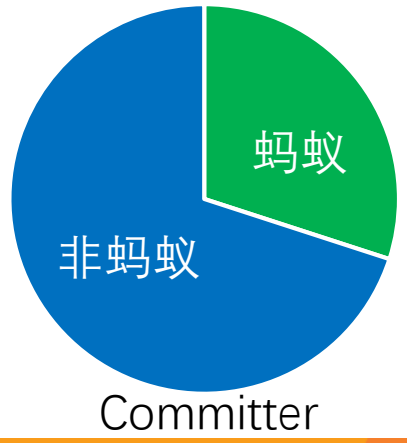
开源理念

- 社区是开源软件发展的动力
- 借力开源，反哺开源
- 持续向云原生演进

Star: 3100
Committer: 10
Contributor: 78

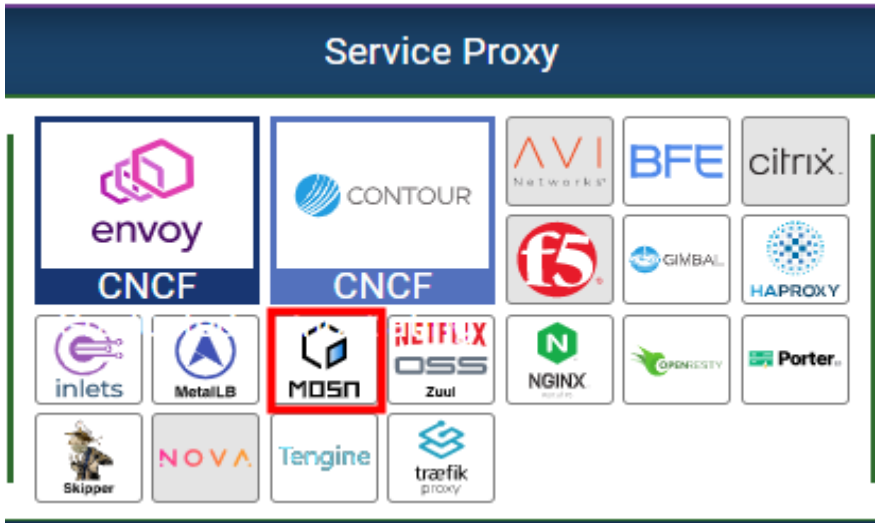
Corporate users: 20+
Commits: 4894
Release: 27





Committer

- mosn
- holmes
- BabaSSL
- mosn.io
- community
- proxy-wasm-go-host
- mosn-tutorial
- pkg
- api
- mosn.github.io



MOSN 简介 — 生态建设

Istio 1.6

ISTIO / BLOG / 2020 POSTS / USING MOSN WITH ISTIO: AN ALTERNATIVE DATA PLANE

Using MOSN with Istio: an alternative data plane

A Cloud Native Proxy for Edge or Service Mesh

BY **WANG FAKANG (MOSN.IO)** | JULY 28, 2020 | 6 MINUTE READ

WebAssembly for Proxies (ABI specification)

Implementations

SDKs

- AssemblyScript SDK
- C++ SDK
- Go (TinyGo) SDK
- Rust SDK
- Zig SDK

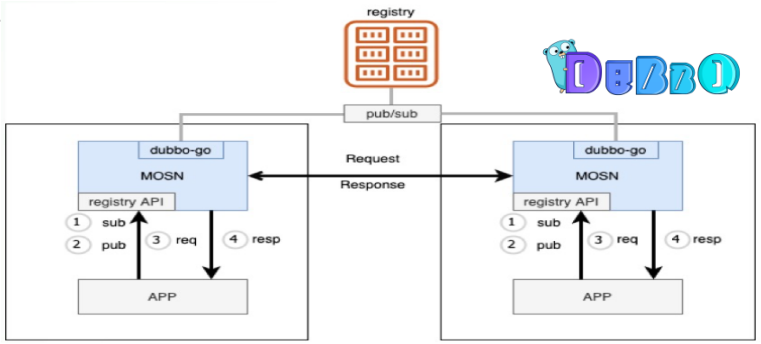
Host environments

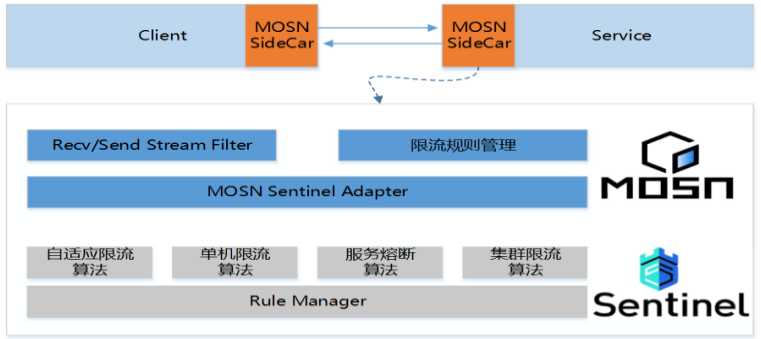
Servers

- Envoy
- Istio Proxy (Envoy-based)
- MOSN
- ATS (proof-of-concept)

Libraries

- C++ Host
- Go Host





MOSN 简介 — 2021 roadmap

- 云原生网络平台建设
- 升级 Xprotocol 框架
- 支持 WASM
- 区块链网络框架
- 代码热更新

- **高性能网络层扩展**
- fastGRPC
- 协程收敛 epoll 模型
- CGO 性能优化

- 支持 zipkin，Jaeger 等
- 支持 ZK，Nacos 等
- 支持 Dubbo 3.0
- 支持 thrift， kafka 等协议

- 支持 Istio 1.10
- 支持 Ingress 和 Gateway
- 推动 UDPA 多协议建设

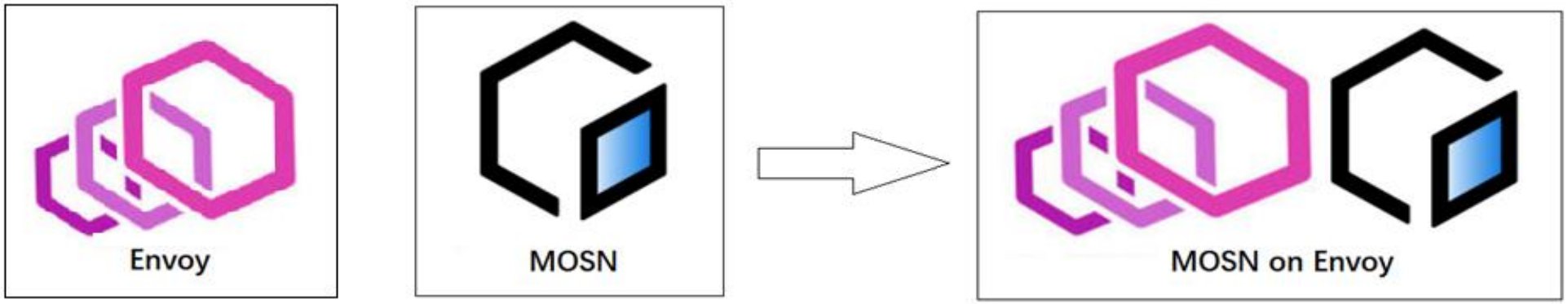
MOSN
网络层扩展思考和选型

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

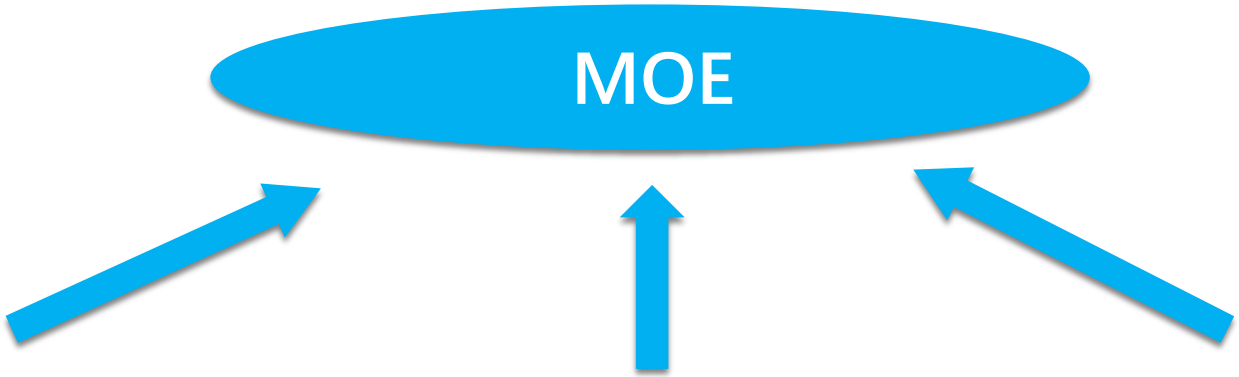


在 Service Mesh 领域，Envoy 和 MOSN 作为其数据面 Sidecar 之一，用于解决传统服务治理体系下的痛点如：多语言中间件组件开发适配成本高、SDK 升级困难、技术复用度差、治理体系不统一等。

MOE = MOSN + Envoy

相互融合，各取所长

MOE 背景介绍 — 为什么做



用户痛点

- east-west、north-south Gateway 技术栈不统一，维护成本高
- Envoy C++ 编写，对于业务方来说开发门槛高

维护成本高、可扩展性弱

技术趋势

- Lua extension
- WASM extension
- External-proc extension

可扩展性、灵活性、生态

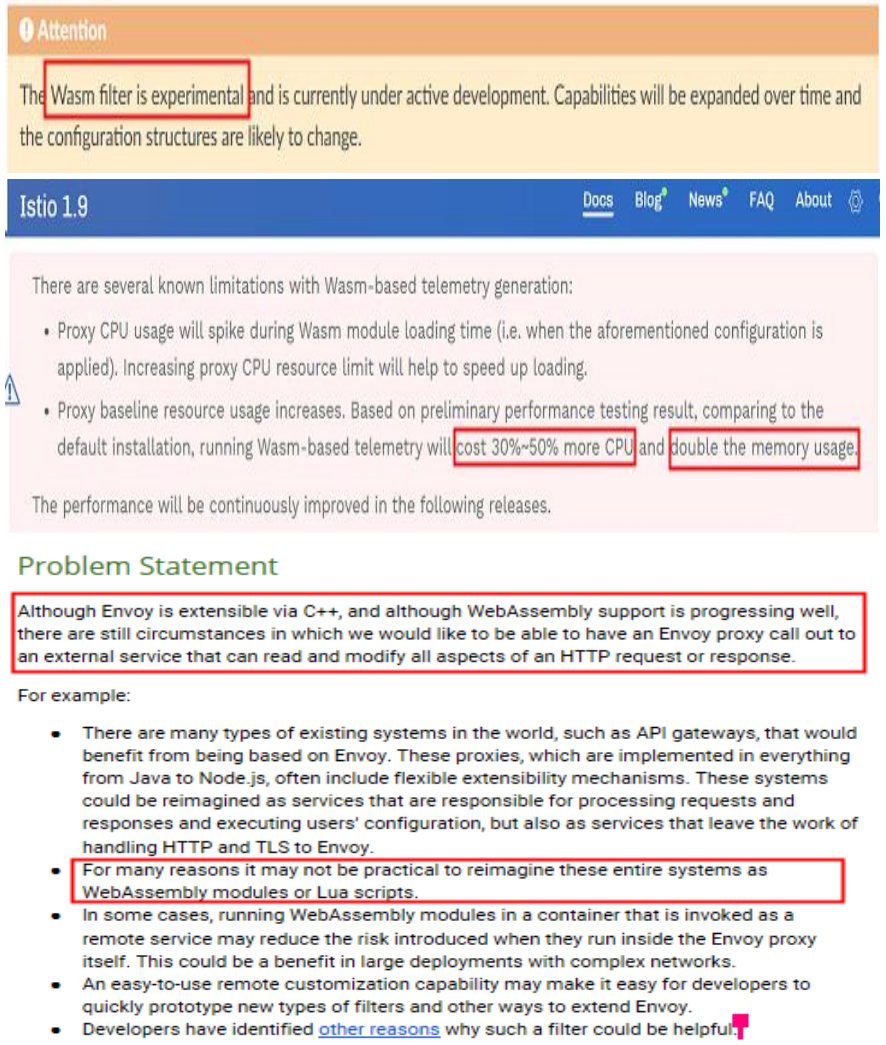
价值意义

- 技术共享，融合 Envoy 和 MOSN 优势
- 增强 Envoy 和 GoLang 社区生态粘性

Envoy 和 GoLong 生态打通

MoE 背景介绍 — 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	适合治理能力已经是一个远程服务，集成进 Envoy	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.

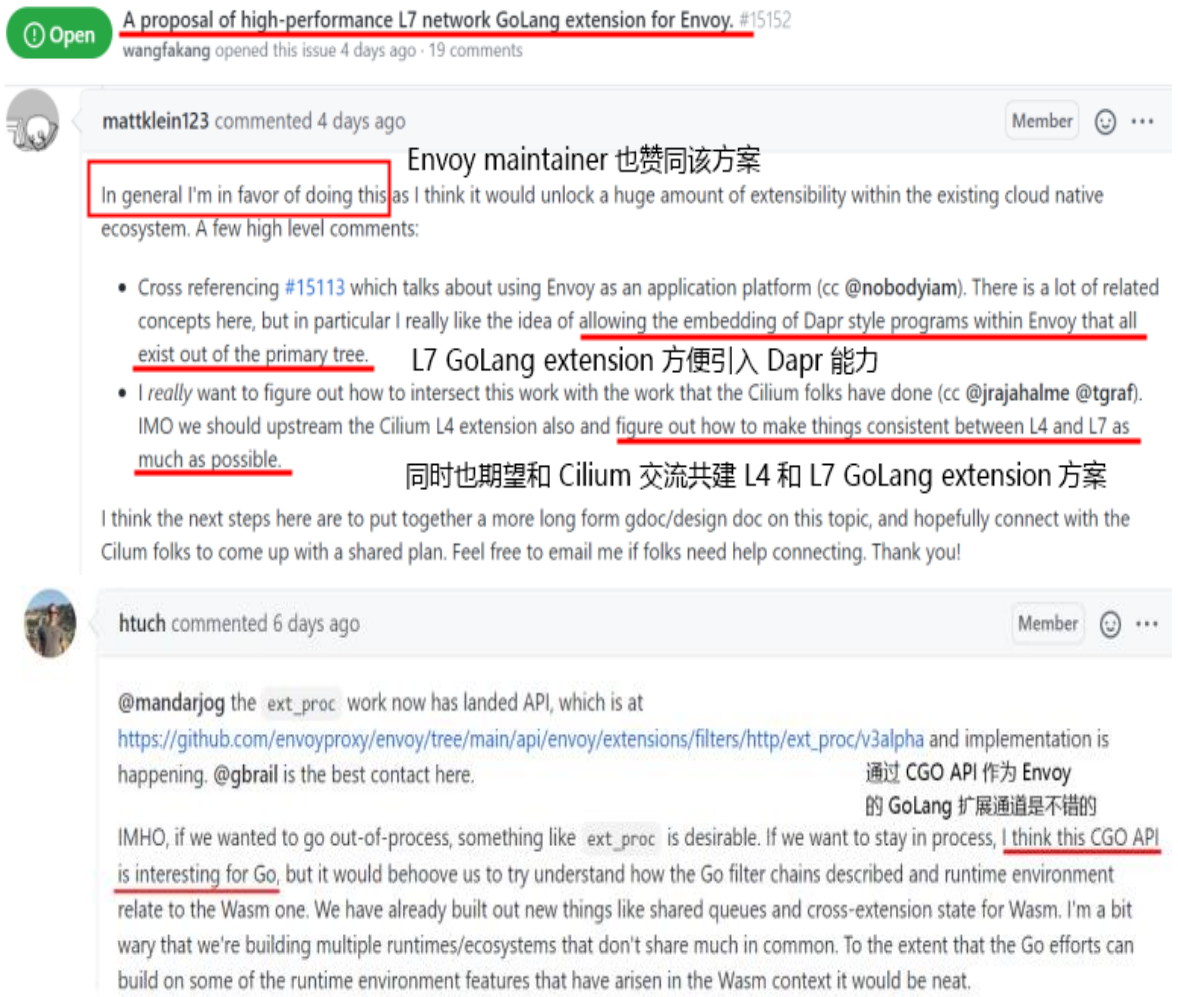
MoE 背景介绍 — 方案分析

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

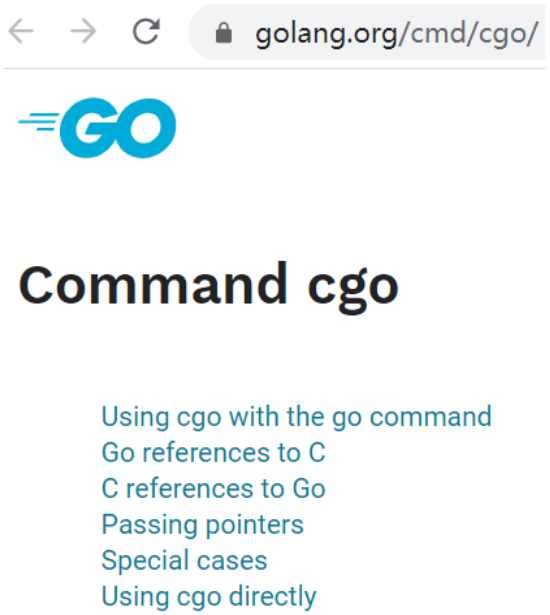
对比：MOE 相比 ext-proc 无需跨进程 gRPC，性能高，易管理；相比 WASM 无需网络 IO 操作转换成本；相比 Lua 生态好、能复用现有的 SDK，对于处理上层业务更合适

Envoy 社区讨论



MOE 背景介绍 — 方案评估

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

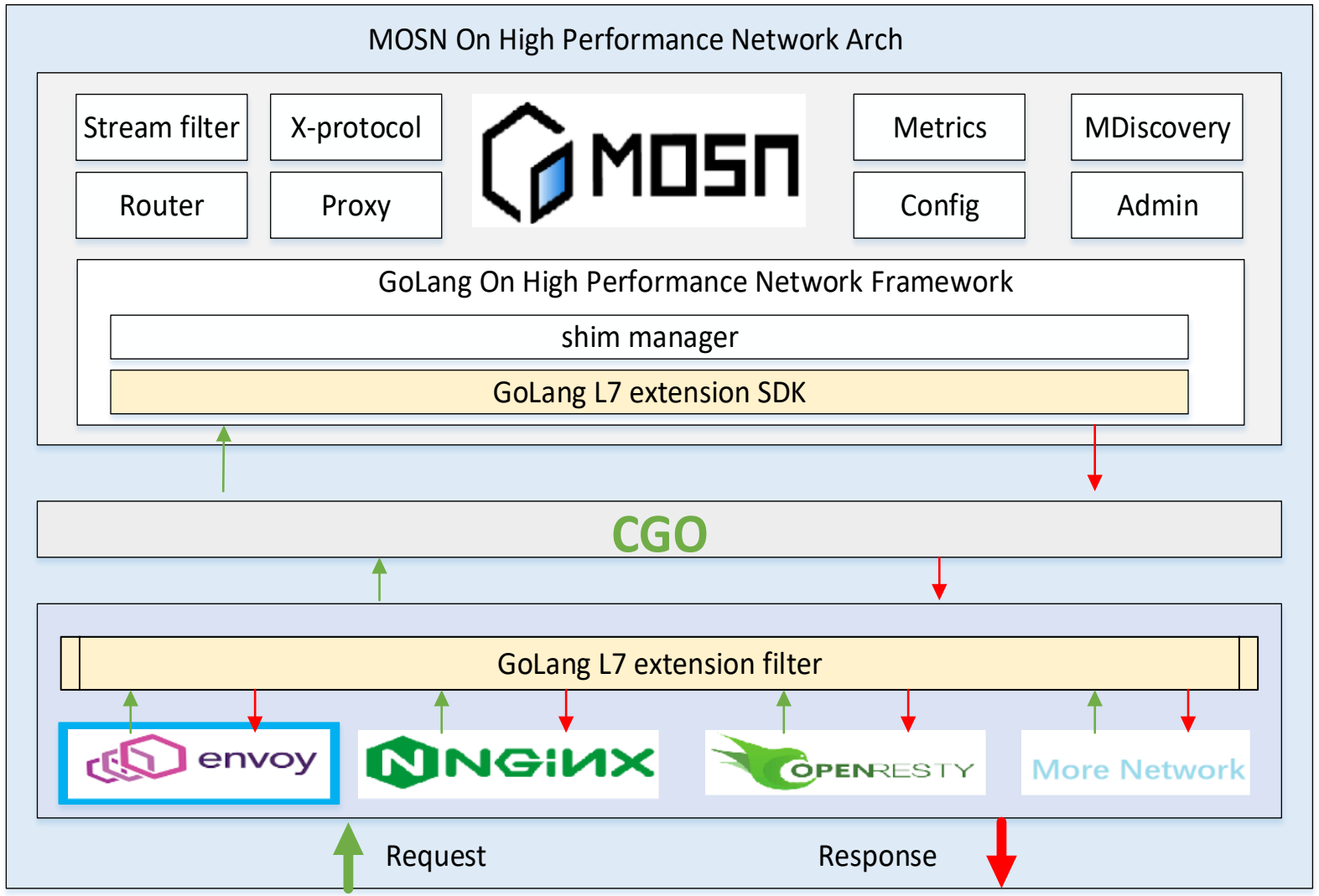
CGO 一次调用开销在 0.08 ~ 1.626 us，另外 CGO 调用开销呈线性增长；CGO 中增加 Go 自身计算逻辑时,其 Go 的计算消耗也呈线性增长

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
解决方案及实践介绍

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**

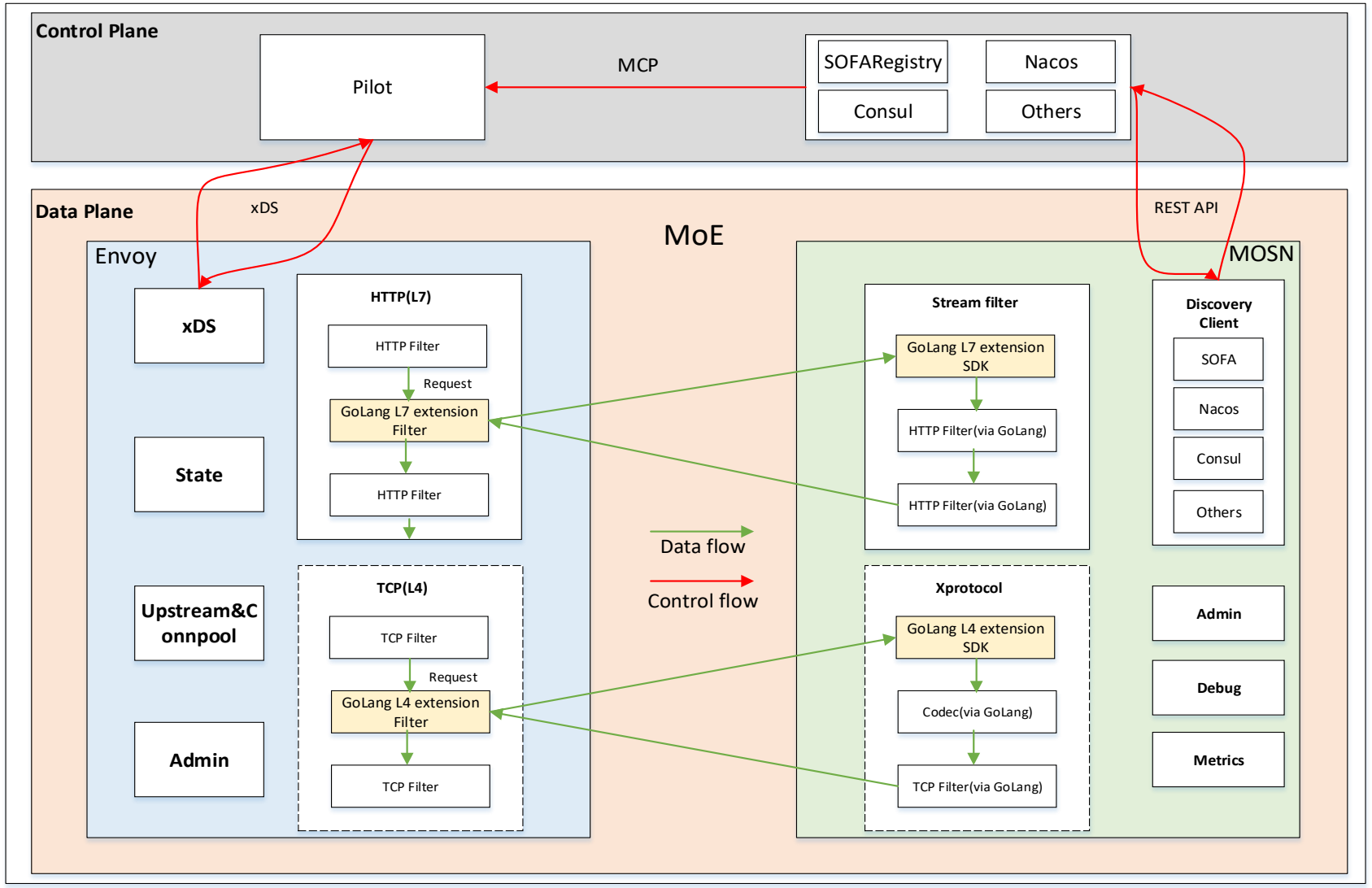
  headers
  body
  trailers

- **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

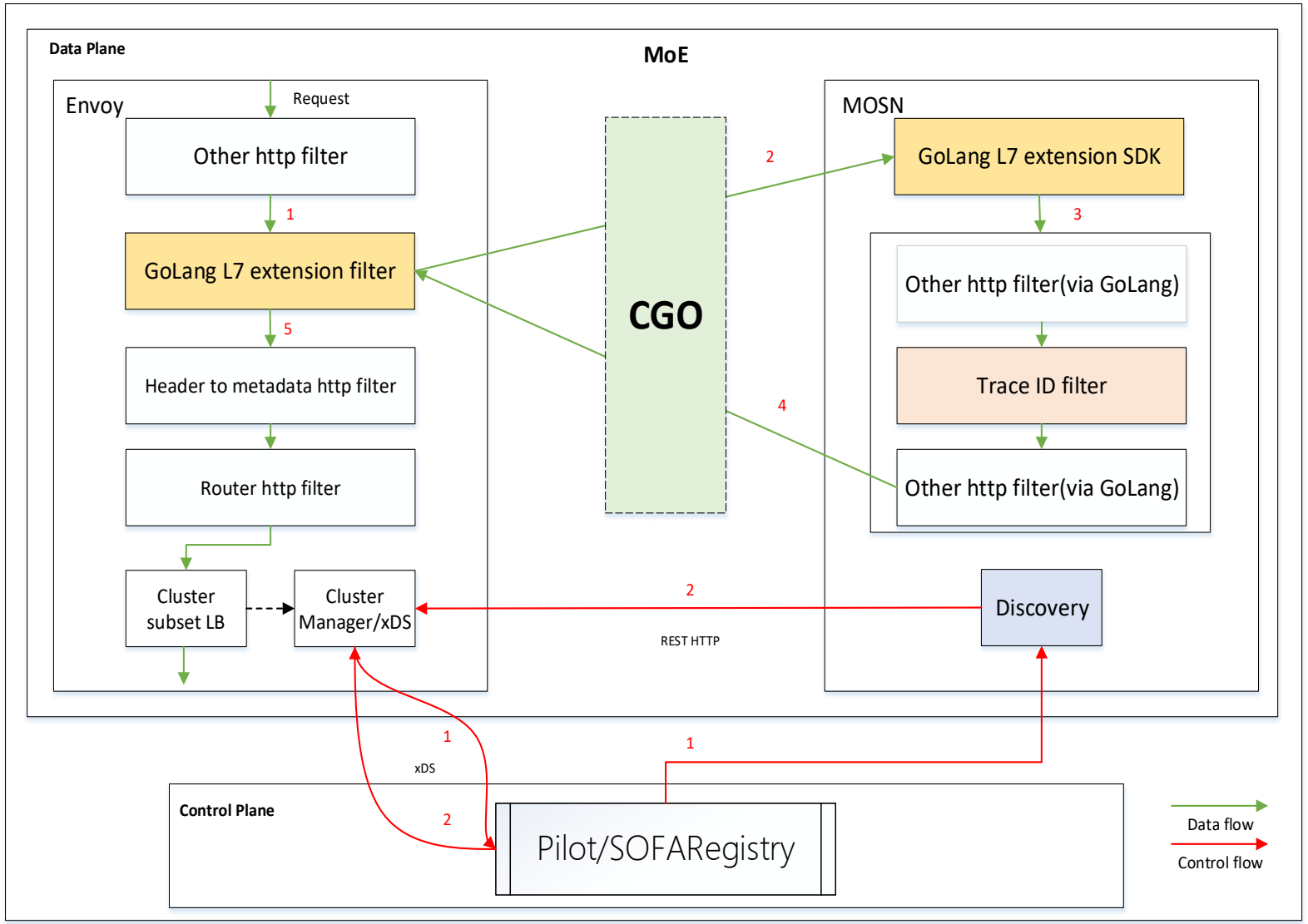
Envoy 复用基础能力

- 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

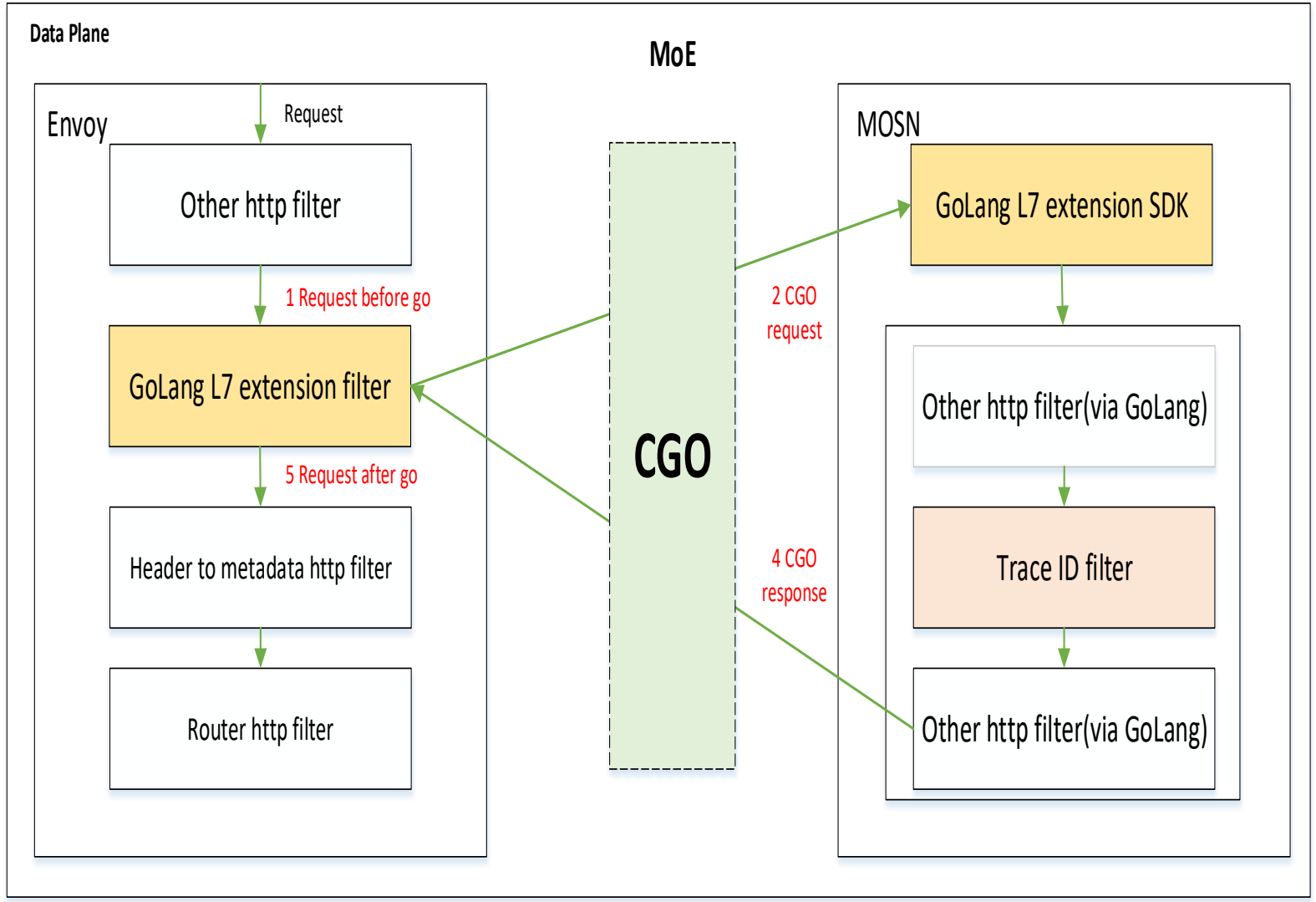


相关问题点

- 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**

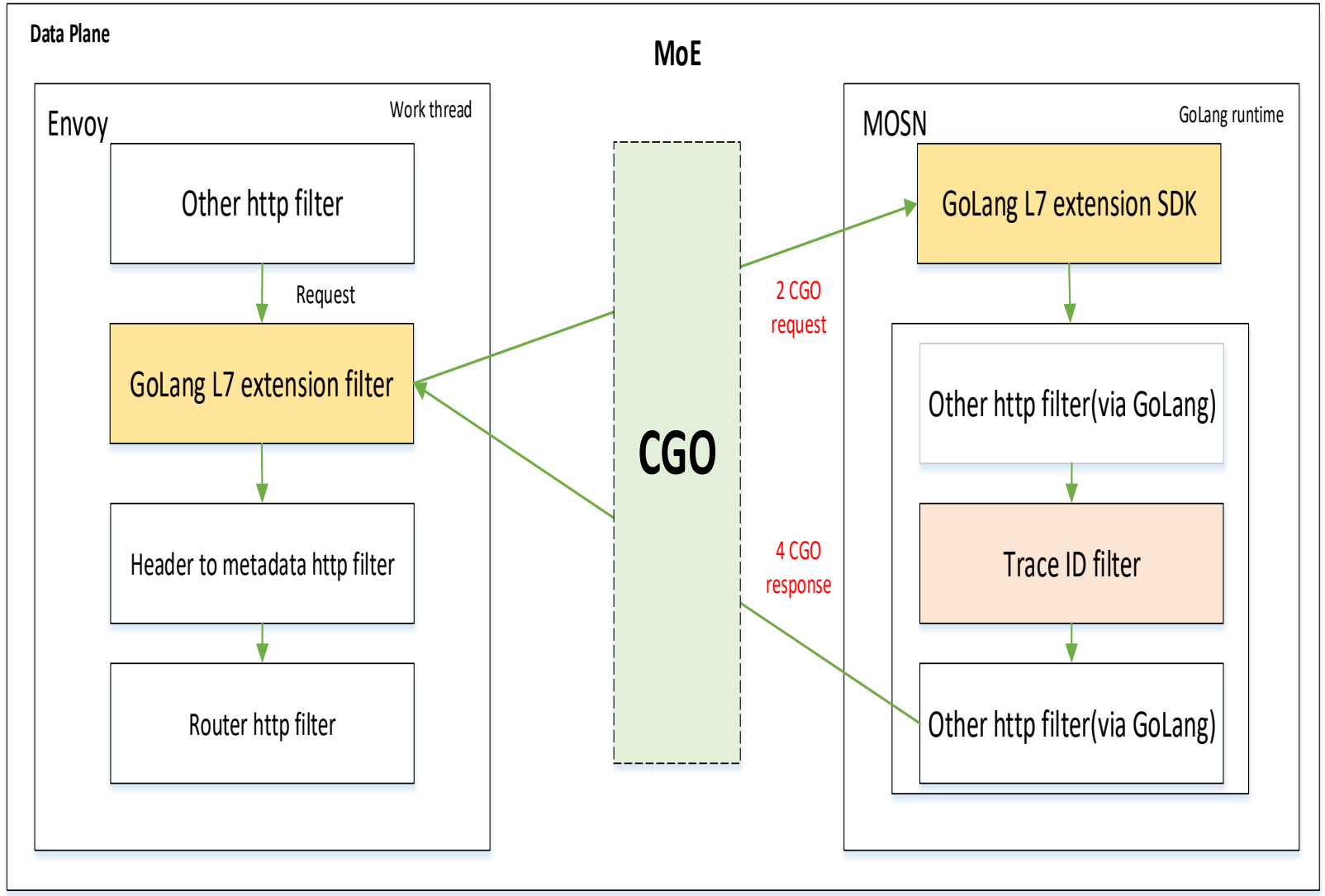
  headers
  body
  trailers

- **returns**

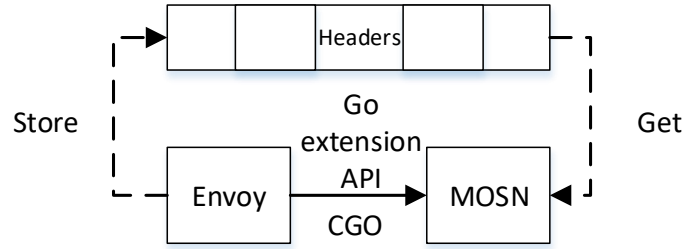
  C.Response{
  headers,
  body,
  trailers,
  optionFlags}

……

MOE 方案介绍 — MOSN 和 Envoy 内存如何管理

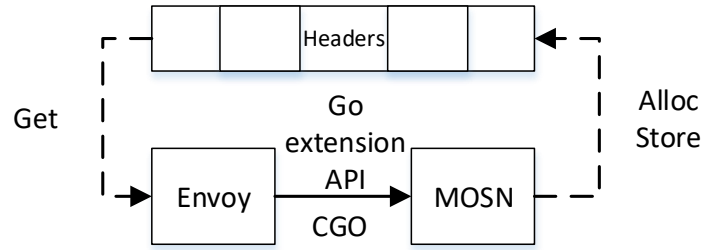


【请求链路】将 Envoy 中的请求信息拷贝一份传递给 MOSN？需要额外内存分配和拷贝？

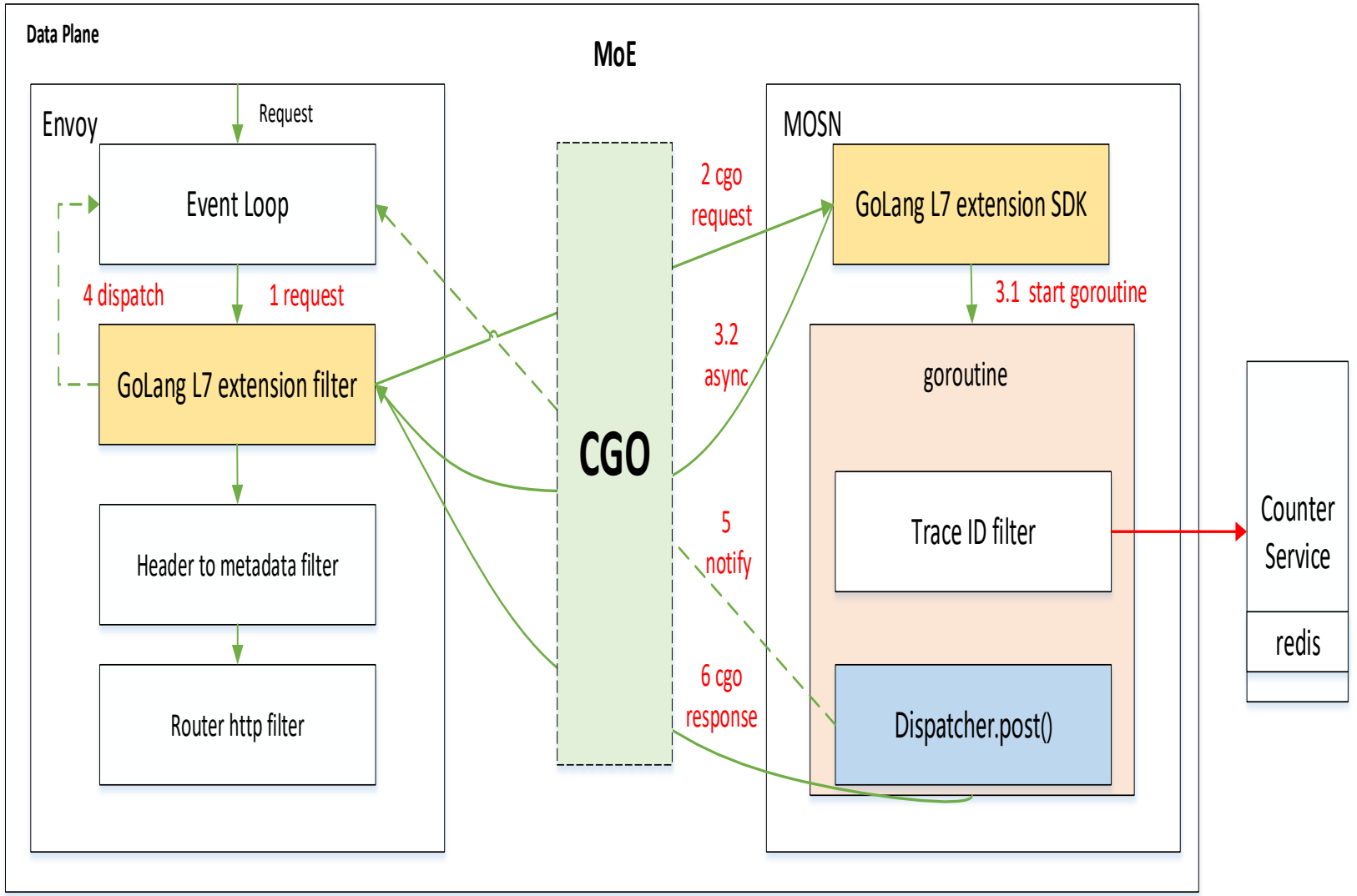


相比 WASM、Lua，MoE 内存 Zero Copy

【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



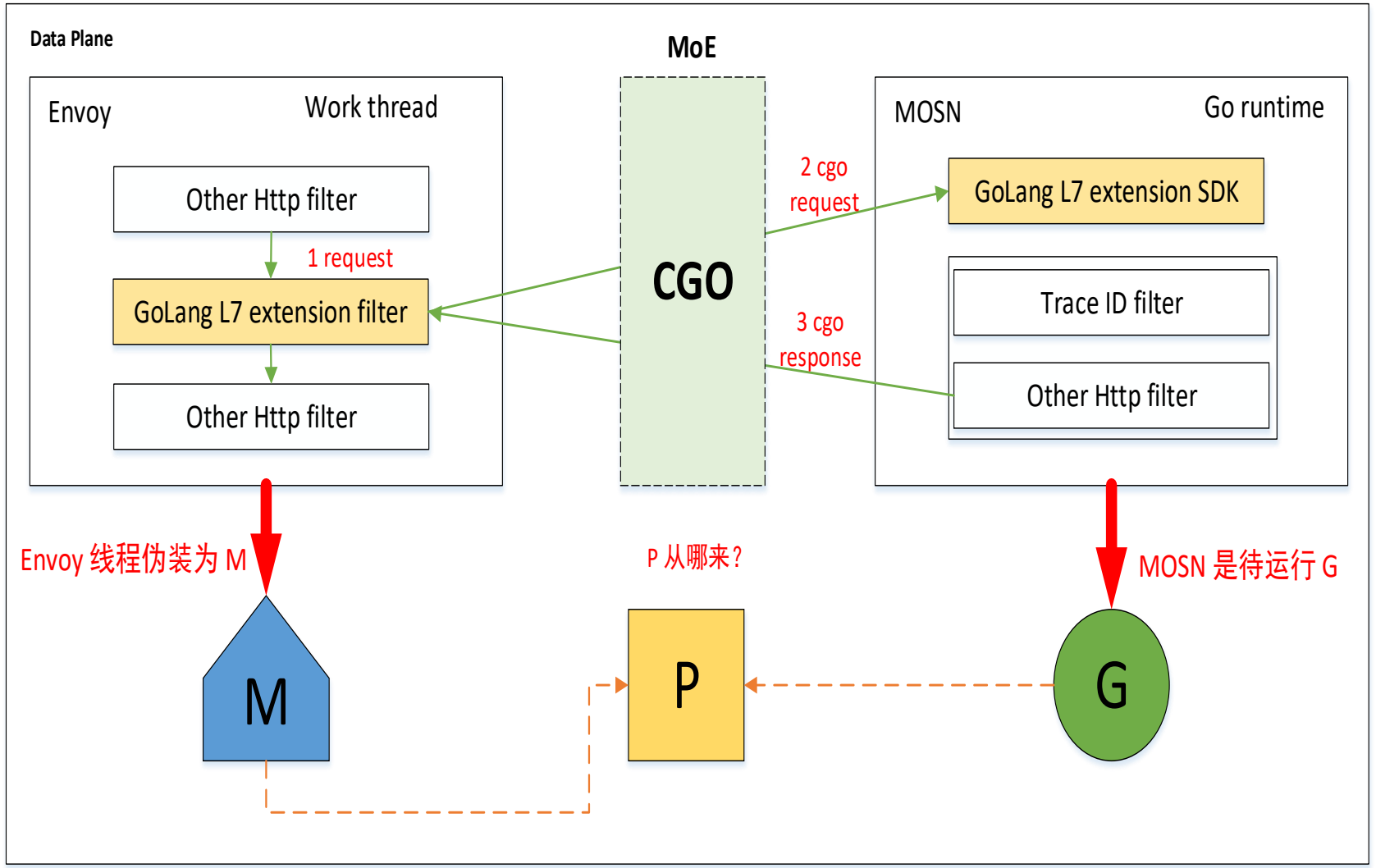
Envoy 是异步非阻塞事件模型，那 MOSN(GoLang) filter 中存在阻塞操作需要如何处理？

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

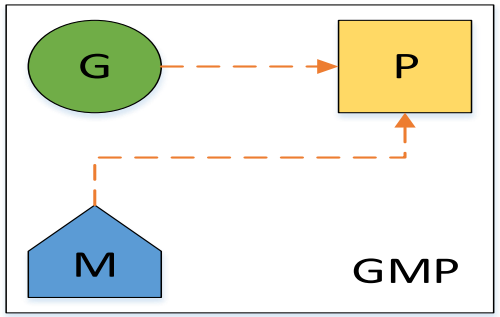
对于阻塞操作，通过 GoLang 的 groutine（协程） 结合 Envoy 的 event loop callback 机制：

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

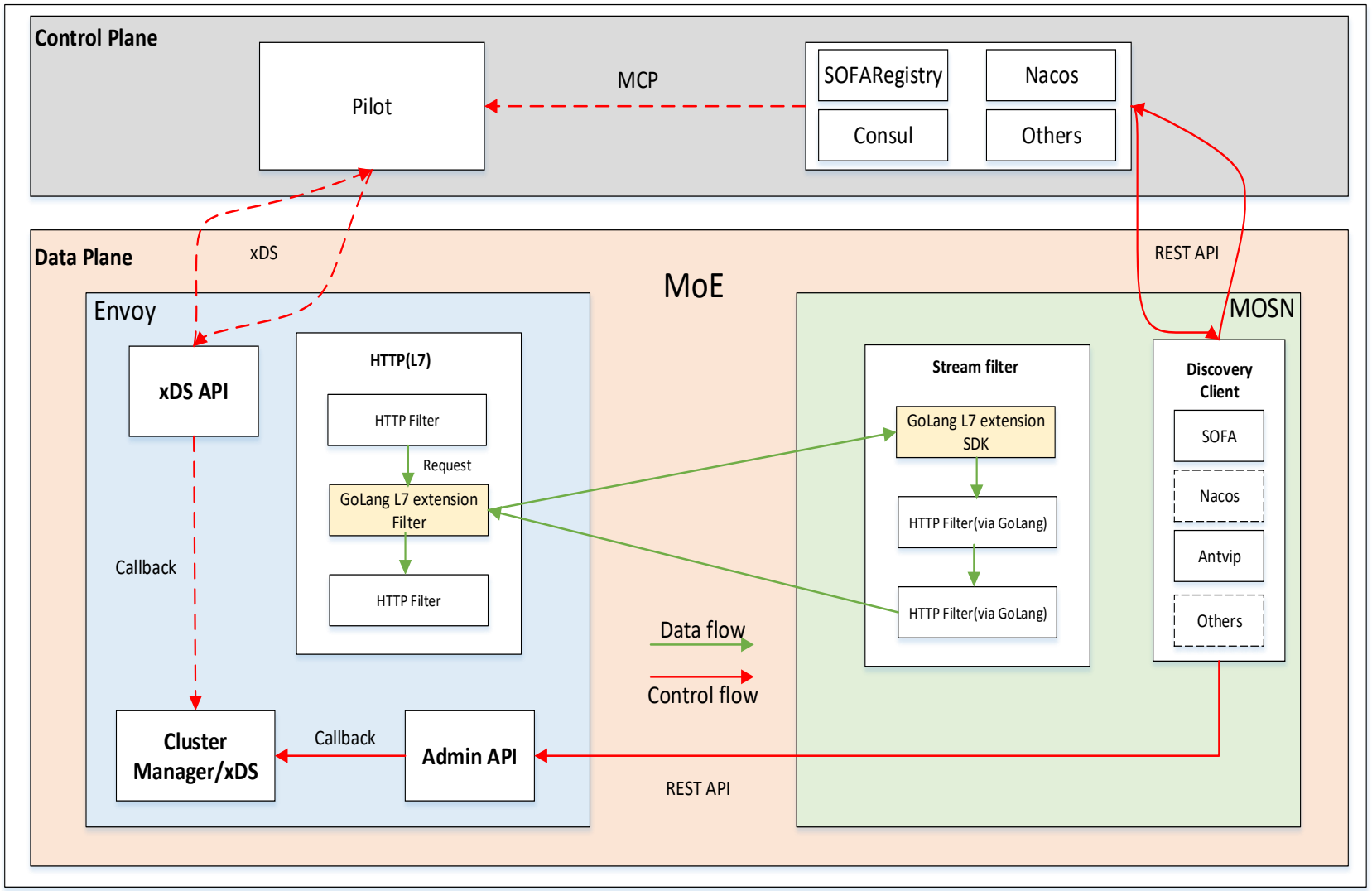


Envoy 通过 CGO 执行 MOSN(GoLang)，此时 P 的数量如何管理？M 从哪来？



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 — 如何 Debug

Envoy

- Admin API
- Debug log
- Request/Connection metrics

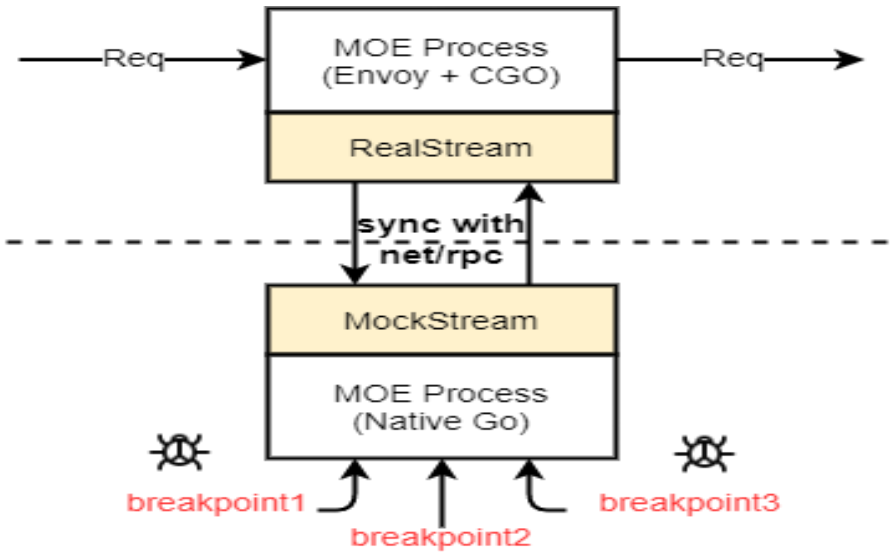
MOSN(GoLang)

- Admin API
- Debug log
- GoLang runtime 指标

Envoy 和 MOSN 交互层

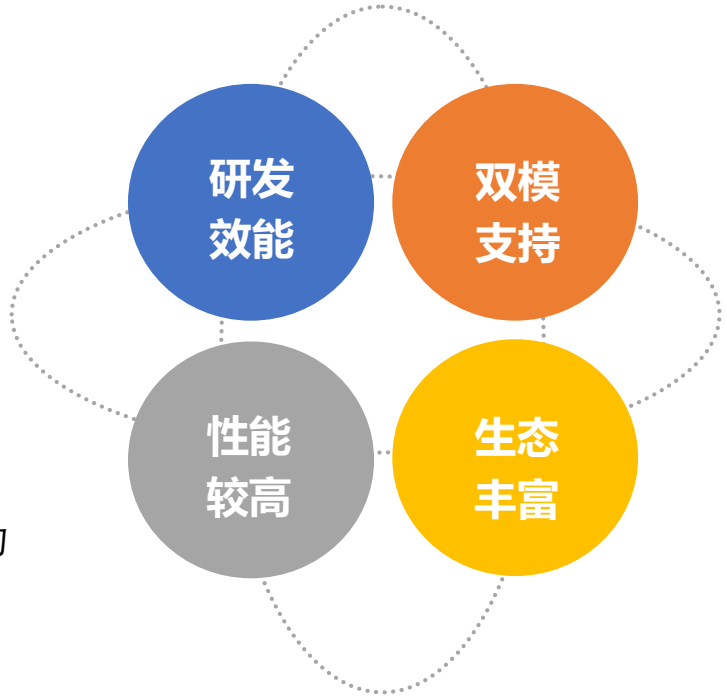
- MOSN（GoLang） 侧耗时统计
- 交互异常数统计
- GoLang 程序异常场景下的容灾处理

CGO 断点调试支持



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

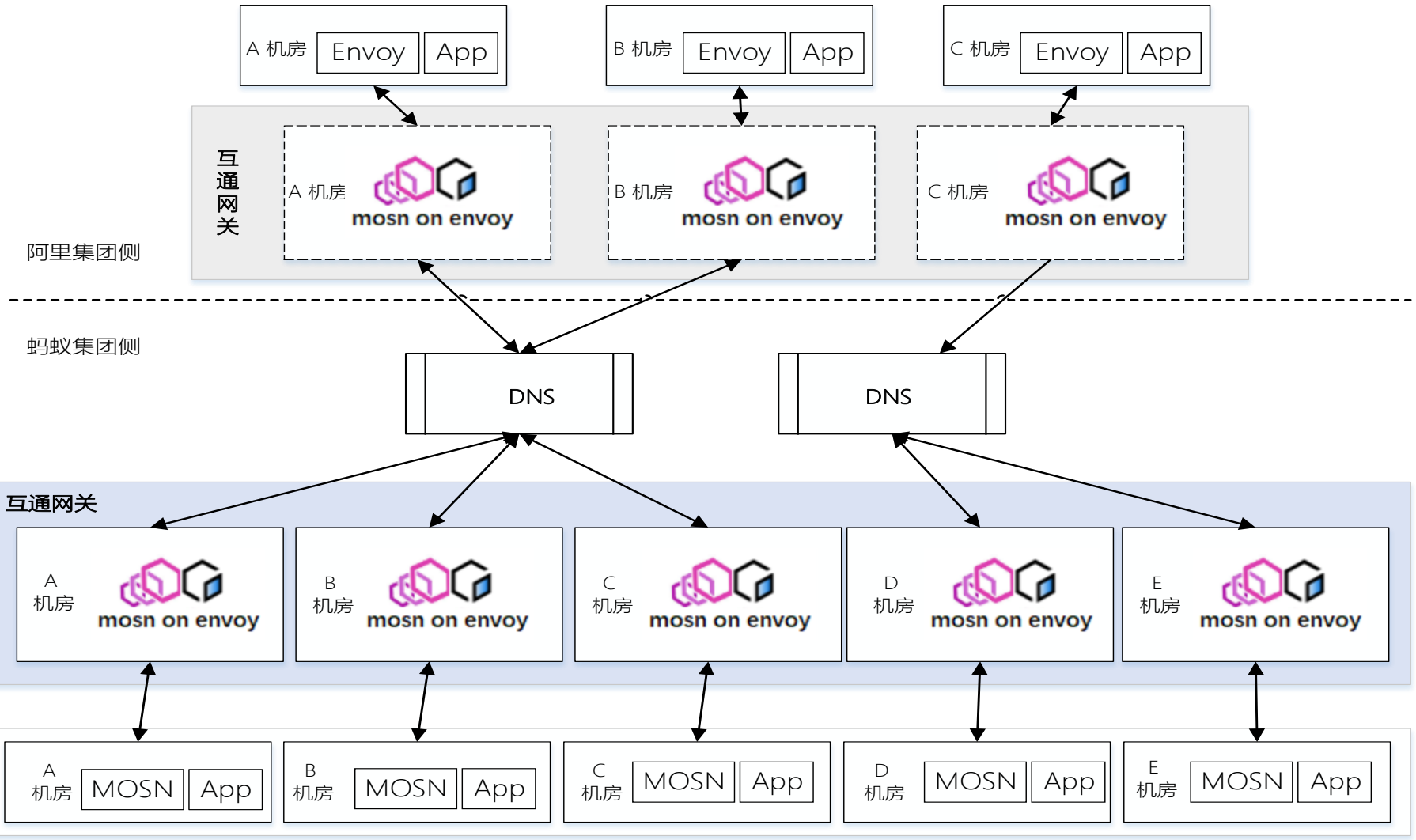


- 同时具备云原生 xDS 、REST API服务元数据管理通道能力

- 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 实践介绍 — 部署架构



MOE 实践介绍 — 相关采坑记录

```
$go run  gogo_stack.go
0xc000032770
runtime: bad pointer in frame main.main at 0xc000045f70: 0x13
fatal error: invalid pointer found on stack

runtime stack:
runtime.throw(0x47e512, 0x
        /home/fakang.wfk/w
runtime.adjustpointers(0xc
        /home/fakang.wfk/w
runtime.adjustframe(0x7fff
        /home/fakang.wfk/w
runtime.gentraceback(0xfff
        /home/fakang.wfk/w
runtime.copystack(0xc00000
        /home/fakang.wfk/w
runtime.newstack()
        /home/fakang.wfk/w
runtime.morestack()
        /home/fakang.wfk/w

goroutine 1 [copystack]:
runtime.gwrite(0xc000045bf
        /home/fakang.wfk/w
runtime.printhex(0xc000032438)
        /home/fakang.wfk/work/github/go/src/runtime/print.go:234 +0xd8 fp=0xc00004
runtime.printpointer(0xc000032438)
        /home/fakang.wfk/work/github/go/src/runtime/print.go:238 +0x2b fp=0xc00004
main.test(0x13)
        /home/fakang.wfk/work/github/go_test/go_stack/gogo_stack.go:9 +0x60 fp=0xc
main.main()
```

```go
package main

import "unsafe"
import "C"

//go:noinline
func test(p *C.char) {
        var y [100]int
        println(&y)
}

func main() {
        var p0 *C.char
        p0 = (*C.char)(unsafe.Pointer(uintptr(19)))

        println(&p0)
        test(p0)
        println(&p0)
}
```

```go
// Adjust any pointers contained therein.
func adjustpointers(scanp unsafe.Pointer, bv *bitvector, adjinfo *adjustinfo, f funcInfo) {
        minp := adjinfo.old.lo
        maxp := adjinfo.old.hi
        delta := adjinfo.delta
        num := uintptr(bv.n)
        // If this frame might contain channel receive slots, use CAS
        // to adjust pointers. If the slot hasn't been received into
        // yet, it may contain stack pointers and a concurrent send
        // could race with adjusting those pointers. (The sent value
        // itself can never contain stack pointers.)
        useCAS := uintptr(scanp) < adjinfo.sghi
        for i := uintptr(0); i < num; i += 8 {
                if stackDebug >= 4 {
                        for j := uintptr(0); j < 8; j++ {
                                print("        ", add(scanp, (i+j)*sys.PtrSize), ":", ptrnames[bv.ptrbit(i+j)], ":", he
x(*(*uintptr)(add(scanp, (i+j)*sys.PtrSize))), " # ", i, " ", *addb(bv.bytedata, i/8), "\n")
                        }
                }
                b := *(addb(bv.bytedata, i/8))
                for b != 0 {
                        j := uintptr(sys.Ctz8(b))
                        b &= b - 1
                        pp := (*uintptr)(add(scanp, (i+j)*sys.PtrSize))
                retry:
                        p := *pp
                        if f.valid() && 0 < p && p < minLegalPointer && debug.invalidptr != 0 {
                                // Looks like a junk value in a pointer slot.
                                // Live analysis wrong?
                                getg().m.traceback = 2
                                print("runtime: bad pointer in frame ", funcname(f), " at ", pp, ": ", hex(p), "\n")
                                throw("invalid pointer found on stack")
                        }
                        if minp <= p && p < maxp {
                                if stackDebug >= 3 {
                                        print("adjust ptr ", hex(p), " ", funcname(f), "\n")
                                }
                                if useCAS {
                                        ppu := (*unsafe.Pointer)(unsafe.Pointer(pp))
                                        if !atomic.Casp1(ppu, unsafe.Pointer(p), unsafe.Pointer(p+delta)) {
                                                goto retry
                                        }
                                } else {
                                        *pp = p + delta
                                }
                        }
                }
        }
}
```

615,1-8 44%

MOE 实践介绍 — 相关采坑记录

GoLang 相关

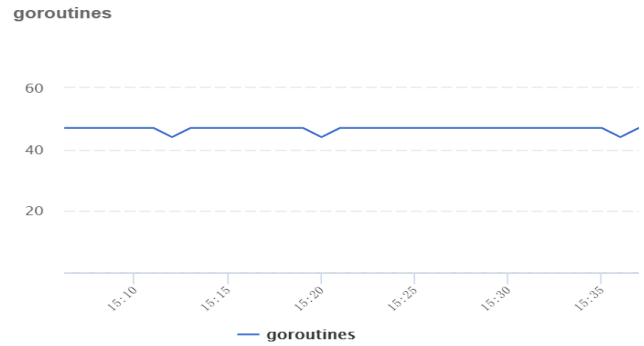
- GoLang recover 失效
- GoLang 返回的字符串被截断
- export function type 关联出错
- 不同方式加载 CGO 程序，则 GoLang runtime 运行时机可能不一样
- CGO 交互内存生命周期管理
- 结构体内存对齐
- GoLang runtime invalidptr check

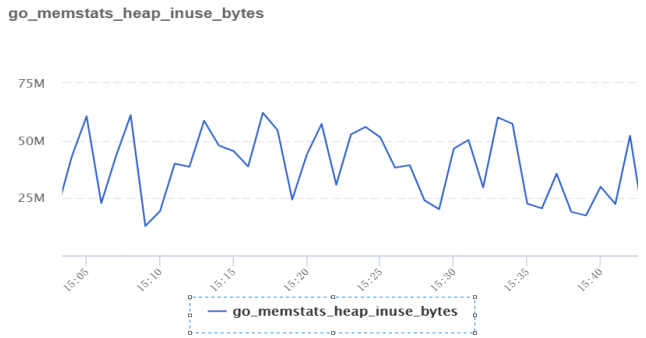
Envoy 相关

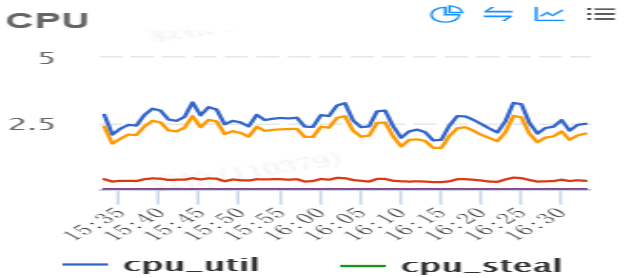
- 默认配置不支持 HTTP1.0
- Envoy 时间模块使用的是 UTC
- upstream 支持 HTTP2 需要显示配置
- 访问日志换行需要自行配置 format 支持
- 异常场景下响应状态码不标准
- 各个 worker 处理请求均衡性问题
- access_log handler 执行顺序不合理
- etc

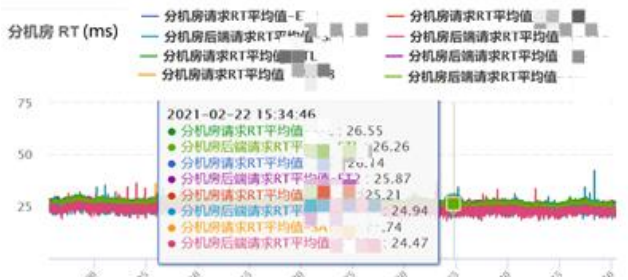
MOE 实践介绍 — 运行效果

- MOE 中 MOSN RT 消耗在 0.05 ms 左右
- MOE 相比于纯 Go 实现的网关处理能力具有 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









开源进展同步

高性能网络扩展层 — MOE

L7 network GoLang extension for Envoy

Open **A proposal of high-performance L7 network GoLang extension for Envoy.** #15152
wangfakang opened this issue 4 days ago · 19 comments

mattklein123 commented 4 days ago — Member

Envoy maintainer 也赞同该方案

In general I'm in favor of doing this as I think it would unlock a huge amount of extensibility within the existing cloud native ecosystem. A few high level comments:

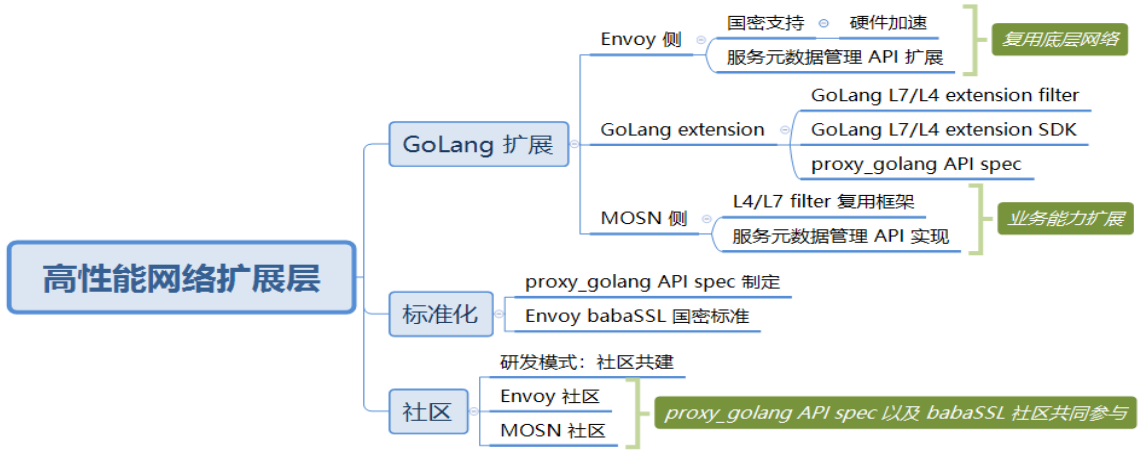
- Cross referencing #15113 which talks about using Envoy as an application platform (cc @nobodyiam). There is a lot of related concepts here, but in particular I really like the idea of allowing the embedding of Dapr style programs within Envoy that all exist out of the primary tree.

L7 GoLang extension 方便引入 Dapr 能力

- I *really* want to figure out how to intersect this work with the work that the Cilium folks have done (cc @jrajahalme @tgraf). IMO we should upstream the Cilium L4 extension also and figure out how to make things consistent between L4 and L7 as much as possible.

同时也期望和 Cilium 交流共建 L4 和 L7 GoLang extension 方案

I think the next steps here are to put together a more long form gdoc/design doc on this topic, and hopefully connect with the Cilum folks to come up with a shared plan. Feel free to email me if folks need help connecting. Thank you!



GoLang extension proposal

Envoy's GoLang extension proposal

version	date	comment	author
v1.0	2021/04/07	draft	fakangwang@gmail.com linuxty@gmail.com yeyeyongjie@gmail.com xiaohanhoho@gmail.com

Background

Envoy is an excellent proxy, designed for Cloud-Native applications. The use of modern C++ guarantees its high performance, but also increases the difficulty for developers. Although Envoy also supports Lua and WASM to enhance its scalability, there are some shortcomings in some scenarios; With the growth of GoLang's cloud native language ecology and popularity, We introduce a proposal that allows Envoy to support GoLang extension capabilities.

Problem Statement

Although Envoy's filter features are already rich. Service Mesh or APIGateway scenes need more customization capabilities. However, development of Envoy's filter has the following problems:

- Envoy uses C++ implementation, which brings the benefit of high performance, but also increases the development difficulty of its extensions for developer
- Although Envoy supports WASM extensions, but GoLang runtime doesn't support WASM very well, and involves network io need to do transformation
- The Lua extension is easier for handling some simple scenarios, but not for more complex business processing, such as Dapr scenarios
- With the growth of GoLang's cloud native language ecology and popularity, there are a large number of SDKs can not be used directly in Envoy
- Developers have identified other reasons why such a filter could be helpful

已支持将 MOSN 部分 filter 运行在 Envoy 中， 欢迎试用
https://github.com/mosn/mosn/blob/master/pkg/networkextention/README-cn.md

Application Runtime — Layotto

背景

Service Mesh 解决了微服务治理的痛点，但在实际业务开发中，**缓存**、**数据库**、**消息队列**、**配置管理等**， 我们仍然需要维护一套重量级的 SDK 并且**侵入应用代码**。

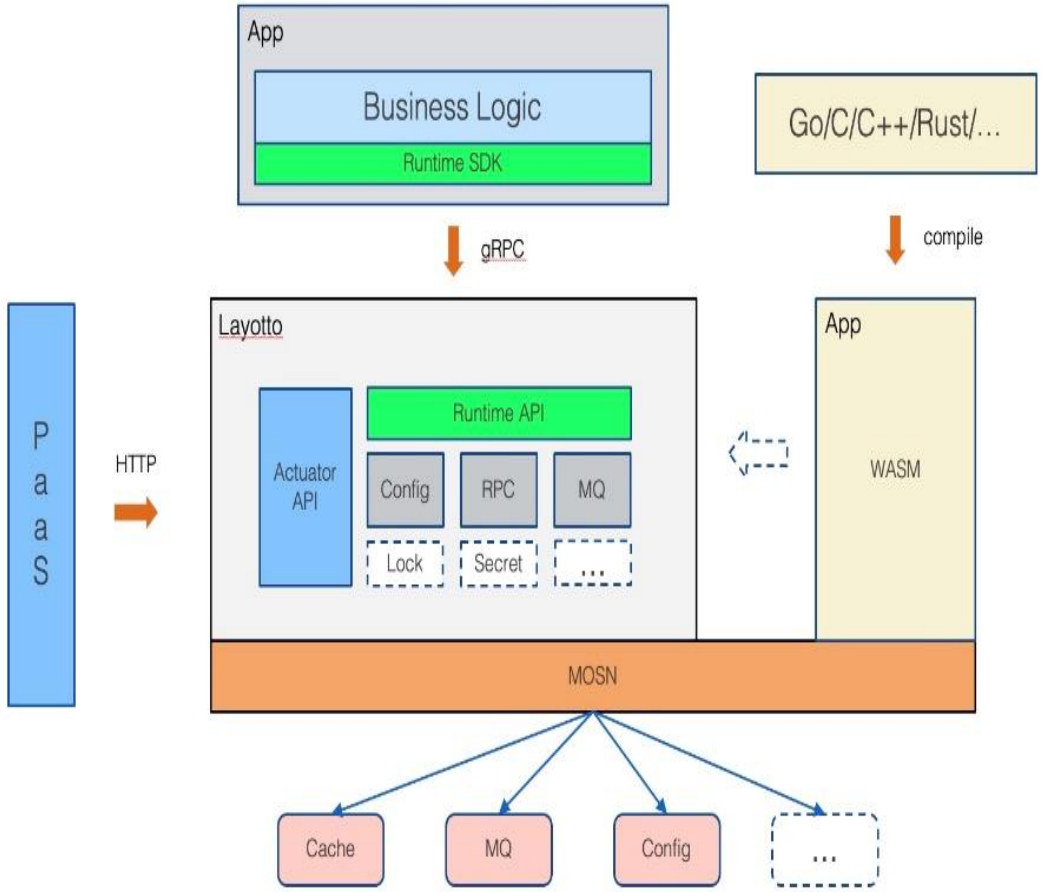
方案

提供 **API 抽象层**，应用程序中只针对这套标准的 API 编程，无需考虑实际运行时的后端服务形态。

优点

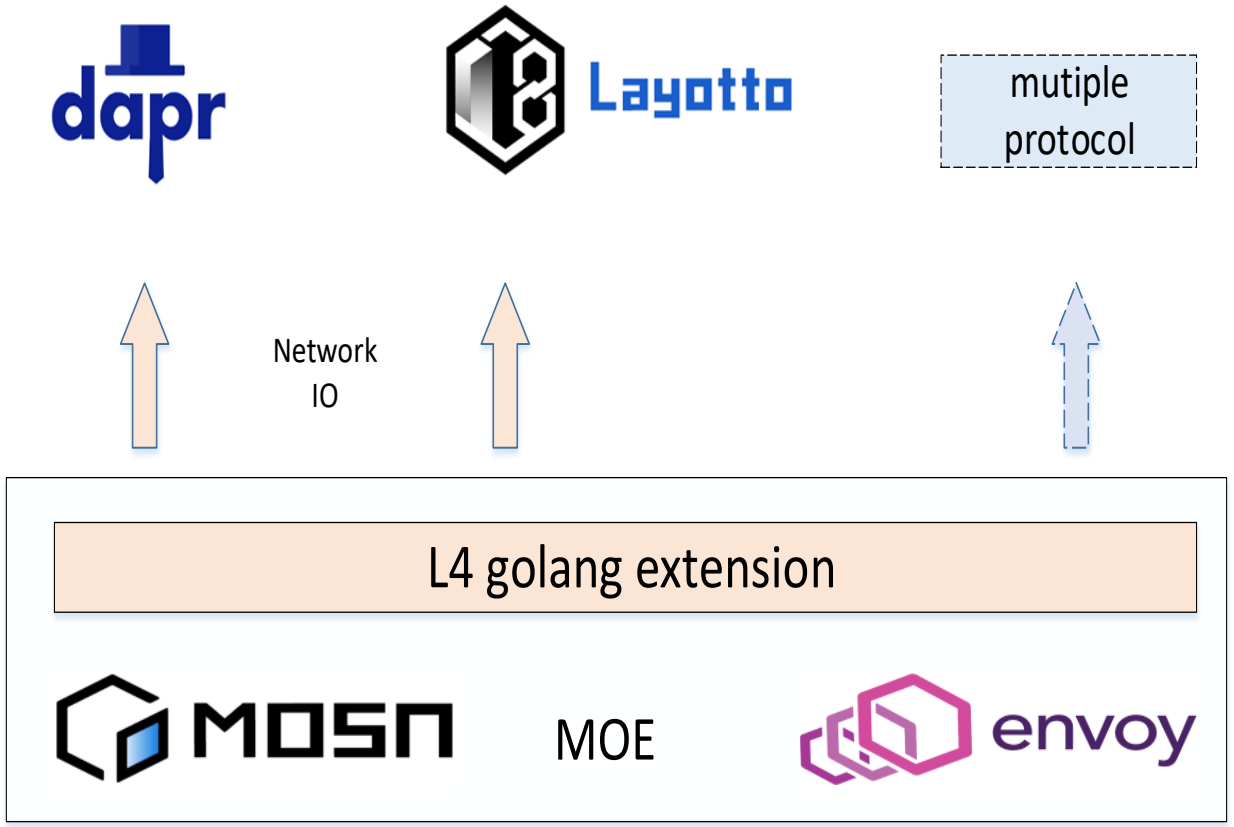
- 多语言友好
- 标准化，无厂商绑定
- 真正实现 Write once, Run anywhere

https://github.com/mosn/layotto



MOE — L4 扩展支持

在 MOE 支持 L4 扩展后，可方便使得 Envoy 集成 Layotto 或 Dapr 能力，从而同时具备 service mesh 与 application runtime 能力。



envoyproxy / envoy

Code | Issues 1k | Pull requests 153 | Actions | Projects | Wiki | Security | Insights

Does Envoy intend to evolve to an application runtime? #15113

Open nobodyiam opened this issue Feb 19, 2021 · 1 comment

nobodyiam commented Feb 19, 2021

Envoy is a great service proxy and is now the default data plane for many service mesh products, which works well to solve many companys' service connectivity challenges.

Yet, in real world cases, a typical company may face much more challenges in areas like cache, database, message queue, configuration management, etc. Which means we still need to maintain a set of heavy weight SDKs along with the application code.

With the release of dapr 1.0.0, we see another possiblity: the sidecar becomes an application runtime that abstracts away the details of backend servcies, so that the application code could safely program against one set of standard api and forget about the issues such as how to handle database failover, how to work with different cloud providers, etc.

mattklein123 commented Feb 19, 2021 Member

This is something I have thought about and would be interested in discussing. I think with WASM this becomes tractable in some pretty interesting ways. Can you reach out to me and we can discuss further (contact info on my website if you can't otherwise find it)?

mattklein123 mentioned this issue Feb 23, 2021

A proposal of high-performance L7 network GoLang extension for Envoy. #15152 Open

来吧，提个 PR 就是 gopher

- [doc] Add a dsl router samples kind/docs
  #1689 opened Jun 18, 2021 by wangfakang
- [Feature] Add `ResponseCodeDetails` and `RequestTerminationDetails` variables kind/feature
  #1688 opened Jun 18, 2021 by wangfakang
- [Feature] cluster name support variable for `RouterActionConfig` kind/feature
  #1687 opened Jun 18, 2021 by wangfakang
- [Feature] MOSN Support Only one server in config kind/enhancement
  #1686 opened Jun 17, 2021 by nejisama
- [Feature] Enhance Router Weight Cluster Configuartion area/router kind/enhancement
  #1685 opened Jun 17, 2021 by nejisama
- [Feature] XProtocol Tracelog Extension kind/enhancement kind/feature
  #1684 opened Jun 17, 2021 by nejisama
- [Feature] MOSN support holmes kind/feature
  #1683 opened Jun 17, 2021 by taoyuanyuan
- [Performance] Variable support reuse mem by sync.Pool area/performance
  #1682 opened Jun 17, 2021 by taoyuanyuan
- [Feature] connpool supoort "max-requests-per-connection" kind/feature
  #1681 opened Jun 16, 2021 by taoyuanyuan
- [Feature] enhance startup parameter kind/enhancement kind/feature
  #1680 opened Jun 16, 2021 by taoyuanyuan
- [Feature] Support websocket protocol kind/feature
  #1679 opened Jun 16, 2021 by taoyuanyuan
- [Feature] Support Dubbo3.0 Protocol kind/feature
  #1678 opened Jun 16, 2021 by taoyuanyuan
- [Feature] LB support session_sticky area/loadbalancer kind/feature
  #1677 opened Jun 16, 2021 by taoyuanyuan

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 meetup https://github.com/mosn/meetup
MOSN github http://github.com/mosn/mosn

MOSN 开源交流群2

欢迎技术交流

Thanks