Kmesh特性验证与补充和文档编写

项目编号：25ed70261

项目导师：李振诚

曹雨森

yusencao@outlook.com

项目背景

1. 项目简述

Kmesh 是基于 eBPF 和可编程内核的高性能、低开销服务网格数据平面。去年完成捐赠CNCF。流量治理相关的功能特性也已经基本完善。但Kmesh dual-engine mode下的流量治理特性缺少相对应的验证测试与文档。因此我们希望就以下几个特性完成验证测试与验证文档的编写。

- **locality load balance**
- **circuit breaker**
- **rate limit**
- **Ingress**
- **engress**

其中关于Ingress和Egress经过先行测试，欠缺DNS解析的逻辑，需要补充开发。

2. 项目要求

技术要求

- go
- 服务网格

产出要求

1. egress特性针对serverEntry的开发
   - 1.1. DNS解析模块逻辑的补充
   - 1.2. 针对serverEntry的STATIC和NONE处理逻辑的补充
   - 1.3. UT测试覆盖

- 1.4. e2e测试的覆盖

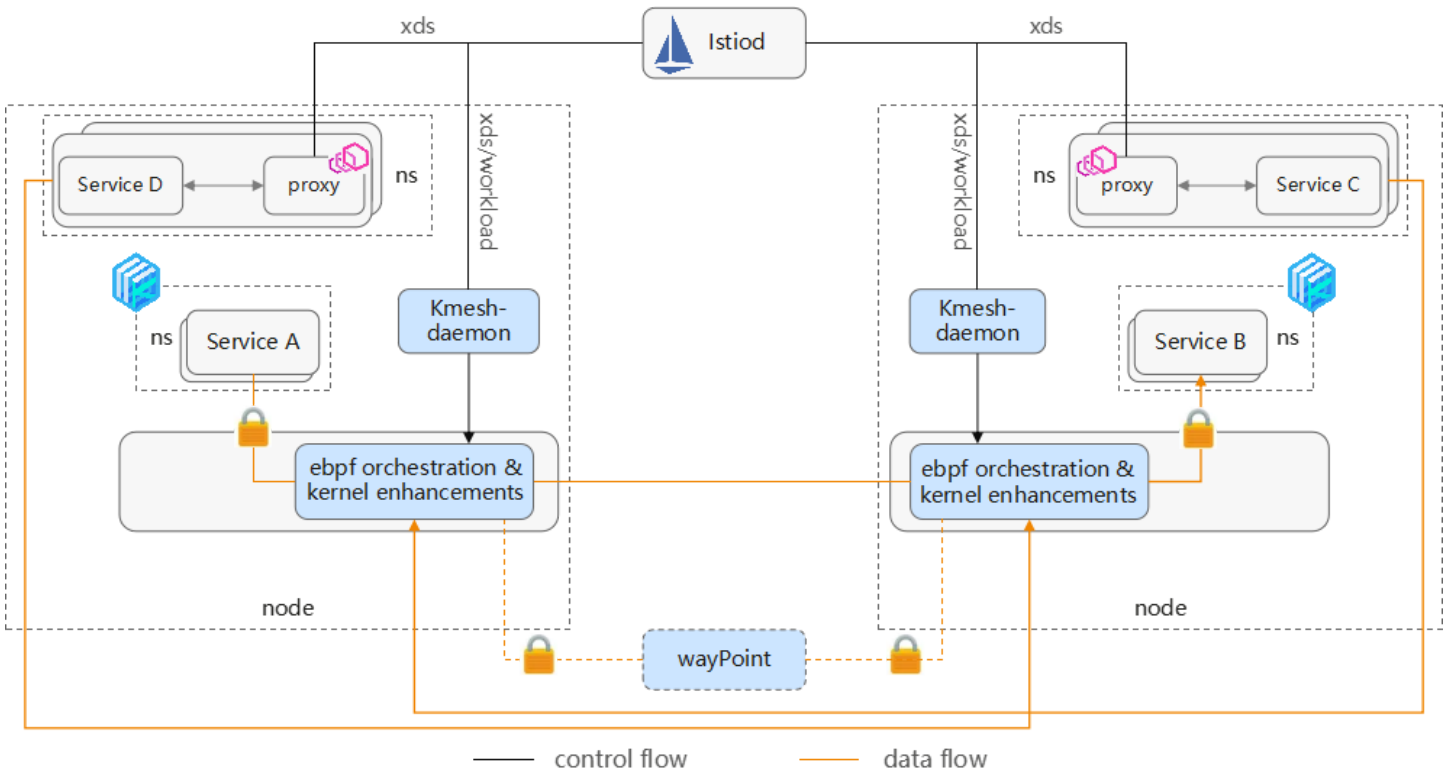
2. 其他功能的验证文档
   - 2.1. locality load balance
   - 2.2. circuit breaker
   - 2.3. rate limit
   - 2.4. Ingress

技术调研

Kmesh 的软件架构由以下核心组件构成：

组件	描述
Kmesh-daemon	负责 eBPF 编排生命周期管理、xDS 协议集成、可观察性等功能的守护进程
eBPF Orchestration	使用 eBPF 实现的流量编排，包括动态路由、授权、负载均衡
Waypoint	基于 istio 的 waypoint 适配 Kmesh 协议，负责 L7 流量管理

Kmesh 的双引擎（Dual-engine）模式使用 eBPF 在内核空间截获流量，同时部署 Waypoint 代理来处理复杂的 L7 流量管理，从而实现内核空间（eBPF）和用户空间（Waypoint）间的 L4 与 L7 分离治理。与 Istio Ambient Mesh 相比，它降低了约 30% 的延迟；与内核原生模式相比，双引擎模式不需要内核增强，具有更广泛的适用性。

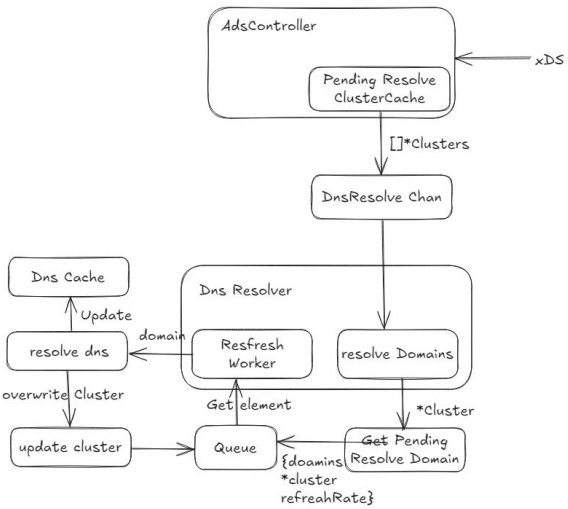


目前，ASM 支持将 Kmesh 的双引擎模式作为服务网格的数据面之一，从而实现更高效的服务管理。具体来说，ASM 可作为控制面使用，而 Kmesh 则可作为数据面部署在阿里云容器服务 Kubernetes（ACK）集群中。

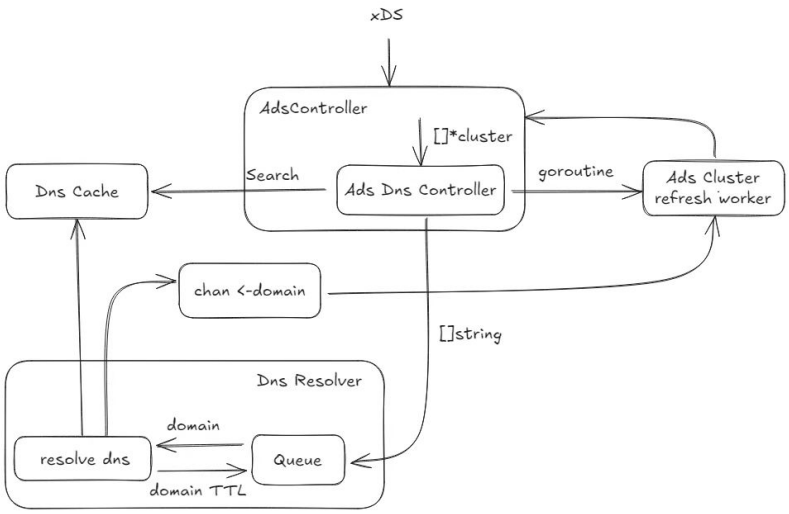
课题概览

先前的 DNS 处理过程中包含了 CDS 刷新的过程，因而 DNS 模块与 kernel-native 模式的耦合度过高，不能再 dual-engine 模式下复用

在Release 1.1版本中，Kmesh 重构了DNS模块。现在，DNS刷新队列中循环的数据是域名，而不是一个包含CDS的结构体。因此，DNS模块不再关心Kmesh的模式，仅提供需要解析的主机名。具体而言，Ads controller 的 DNS 逻辑单独的抽取为了 Ads Dns Controller。



重构前，DNS 模块与 kernel-native 的耦合度过高



重构后，DNS 的逻辑单独抽取为了旁路式的 Dns Resolver

因而 `pkg/controller/workload/workload_processor.go` 中 `handleServicesAndWorkloads` 所缺失的 `ServiceEntry` 相关的逻辑需要补全。

代码块

```
1   func (p *Processor) handleServicesAndWorkloads(services
    []*workloadapi.Service, workloads []*workloadapi.Workload) {
2     var servicesToRefresh []*workloadapi.Service
3     for _, service := range services {
4       if err := p.handleService(service); err != nil {
5         log.Errorf("handle service %v failed, err: %v", service.ResourceName(),
    err)
6       }
7       svcs, wls := p.WaypointCache.Refresh(service)
8       servicesToRefresh = append(servicesToRefresh, svcs...)
9       // Directly add deferred workload to workloads.
10      workloads = append(workloads, wls...)
```

```
    }

    // Handle services that are deferred due to waypoint hostname resolution.
    for _, service := range servicesToRefresh {
      if err := p.handleService(service); err != nil {
        log.Errorf("handle deferred service %v failed, err: %v",
service.ResourceName(), err)
      }
    }

    for _, workload := range workloads {
      // TODO: Kmesh supports ServiceEntry
      if workload.GetAddresses() == nil {
        log.Warnf("workload: %s/%s addresses is nil", workload.Namespace,
workload.Name)
        continue
      }

      if err := p.handleWorkload(workload); err != nil {
        log.Errorf("handle workload %s failed, err: %v",
workload.ResourceName(), err)
      }
    }
  }
```

考虑到需要兼容在 dual-engine 模式下使用的 Workload，因而需要为 workload controller 补全 DNS 逻辑

代码块

```
func NewXdsClient(mode string, bpfAds *bpfads.BpfAds, bpfWorkload
*bpfwl.BpfWorkload, enableMonitoring, enableProfiling bool) *XdsClient {
  client := &XdsClient{
    mode:      mode,
    xdsConfig: config.GetConfig(mode),
  }

  if mode == constants.DualEngineMode {
    client.WorkloadController = workload.NewController(bpfWorkload,
enableMonitoring, enableProfiling)
  } else if mode == constants.KernelNativeMode {
    client.AdsController = ads.NewController(bpfAds)
  }

  client.ctx, client.cancel = context.WithCancel(context.Background())
```

```
14    client.ctx = metadata.AppendToOutgoingContext(client.ctx, "ClusterID",
    client.xdsConfig.Metadata.ClusterID.String())
15    return client
16  }
```

开发计划

- 第一阶段 (7月——8月中旬)
  - 和导师讨论具体的需求和实现要求
  - 对项目进行架构设计，确定技术方案
  - 初步实现 Fleet 支持配置集群网络
- 第二阶段(8月中旬——9月底)
  - 对之前的实现进行进一步完善并进行相关的测试
  - 总结工作内容并输出文档
  - 思考可以改进或者补充的地方