腾讯云

Areaki Mesh 在视频直播应用中的服务网格实践

赵化冰

腾讯云技术专家

Aeraki Mesh 开源项目创始人

https://zhaohuabing.com

https://aeraki.net

Service Mesh

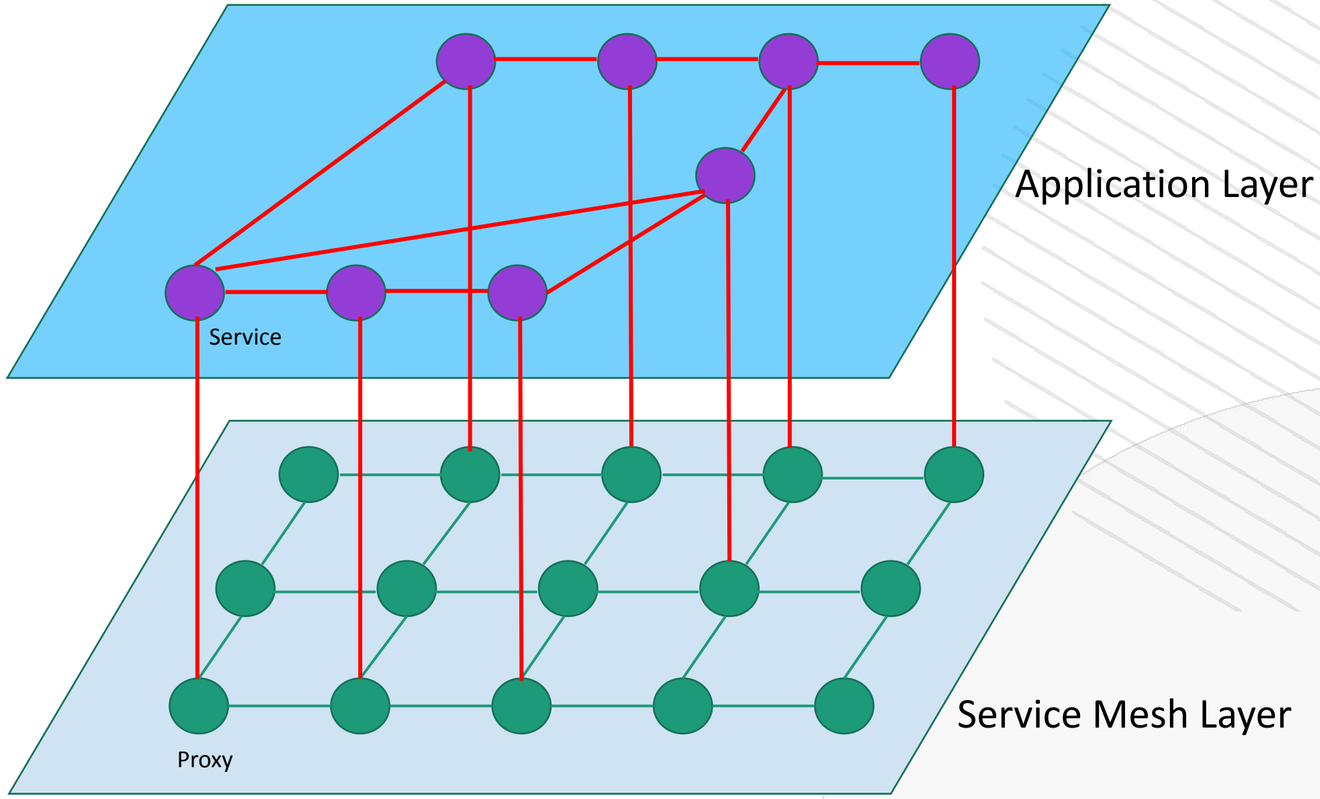
处理服务间通信（主要是七层通信）的云原生基础设施层：

Service Mesh 将各个服务中原来使用 SDK 实现的七层通信相关功能抽象出来，使用一个专用层次来实现，Service Mesh 对应用透明，因此应用可以无需关注分布式架构带来的通信相关问题，而专注于其业务价值。

流量控制：服务发现、请求路由、负载均衡、灰度发布、错误重试、断路器、故障注入

可观察性：遥测数据、调用跟踪、服务拓扑

通信安全：服务身份认证、访问鉴权、通信加密



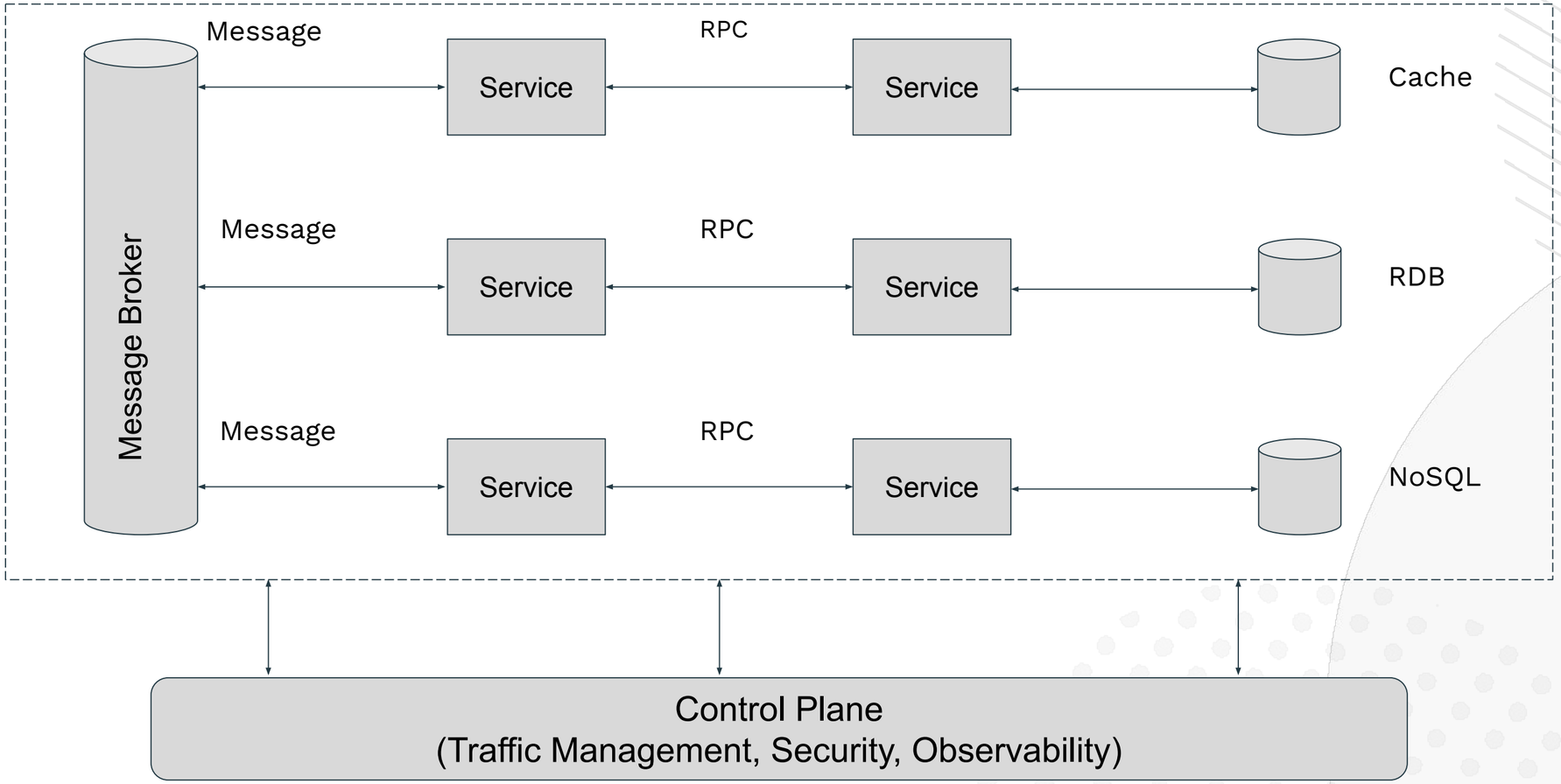
微服务中的常用七层协议

微服务中除 HTTP 之外的的常见七层协议：

- RPC：Thrift, Dubbo, Private RPC Protocols …
- Messaging: Kafka, RabbitMQ …
- Cache: Redis, Memcached …
- Database: mySQL, PostgreSQL, MongoDB …

大多数 Service Mesh 实现都不能在七层上处理这些协议

- 主要关注 HTTP
- 其他协议的流量被作为 TCP 看待



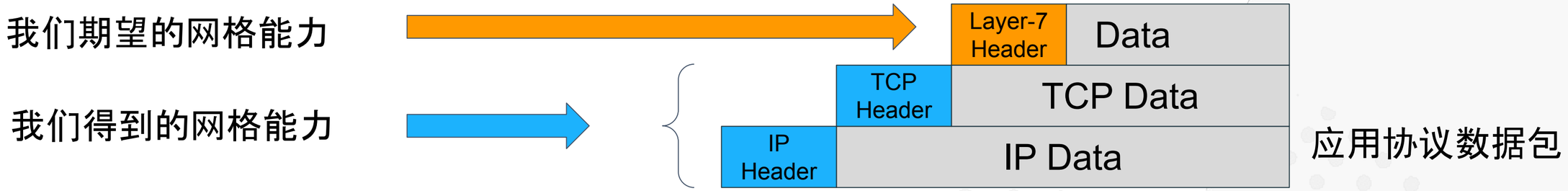
我们希望从服务网格中获得这些协议的哪些治理能力

我们期望的网格能力：七层服务治理

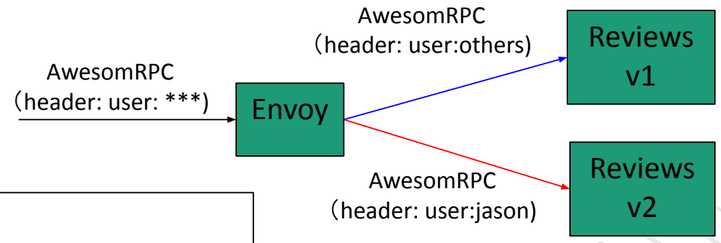
- 服务发现（基于服务的逻辑名称，如 Host，Service）
- 七层负载均衡、基于应用协议的错误码进行重试和熔断
- 基于七层协议头的路由（RPC协议中的调用服务名、方法名等）
- 故障注入（RPC 协议层的错误码）
- 七层请求 Metrics（调用次数，调用失败率等）
- 调用跟踪

我们得到的网格能力：三/四层服务治理

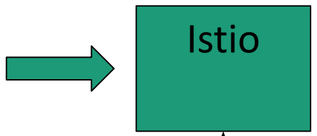
- 服务发现（基于 VIP 或者 Pod IP：DNS 只用于解析得到 IP，不能被 Envoy 感知）
- 四层负载均衡、基于四层链接错误的重试和熔断
- 基于四层的路由（IP + Port）
- 基于四层的 Metrics（TCP收发包数量等）



Istio 协议扩展：控制面和数据面需要进行的改动

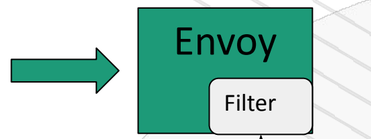



```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews-route
spec:
  hosts:
  - reviews.prod.svc.cluster.local
  awesomeRPC:
  - name: ”canary-route"
    match:
    - headers:
        user:
          exact: jason
    route:
    - destination:
        host: reviews.prod.svc.cluster.local
        subset: v2
  - name: ”default"
    route:
    - destination:
        host: reviews.prod.svc.cluster.local
        subset: v1
```



Istio 代码改动
- 解析 CRD
- 生成 xDS 配置下发

```
{
  "virtual_hosts": [
    {
      "name": "reviews.default.svc.cluster.local:9080",
      "services": [
        "reviews.default.svc.cluster.local",
        "reviews"
      ],
      "routes": [
        {
          "name": ”canary-route"
          "match": {
            "headers":  [
              {
                "name": ":user",
                "exact_match": "jason"
              }
            ],
          },
          "route": {
            "cluster": "outbound|9080||reviews.default.svc.cluster.local | v2",
          },
        },
        {
          "name": ”default"
          "route": {
            "cluster": "outbound|9080||reviews.default.svc.cluster.local | v1",
          },
        }
      ]
    }
  ],
}
```



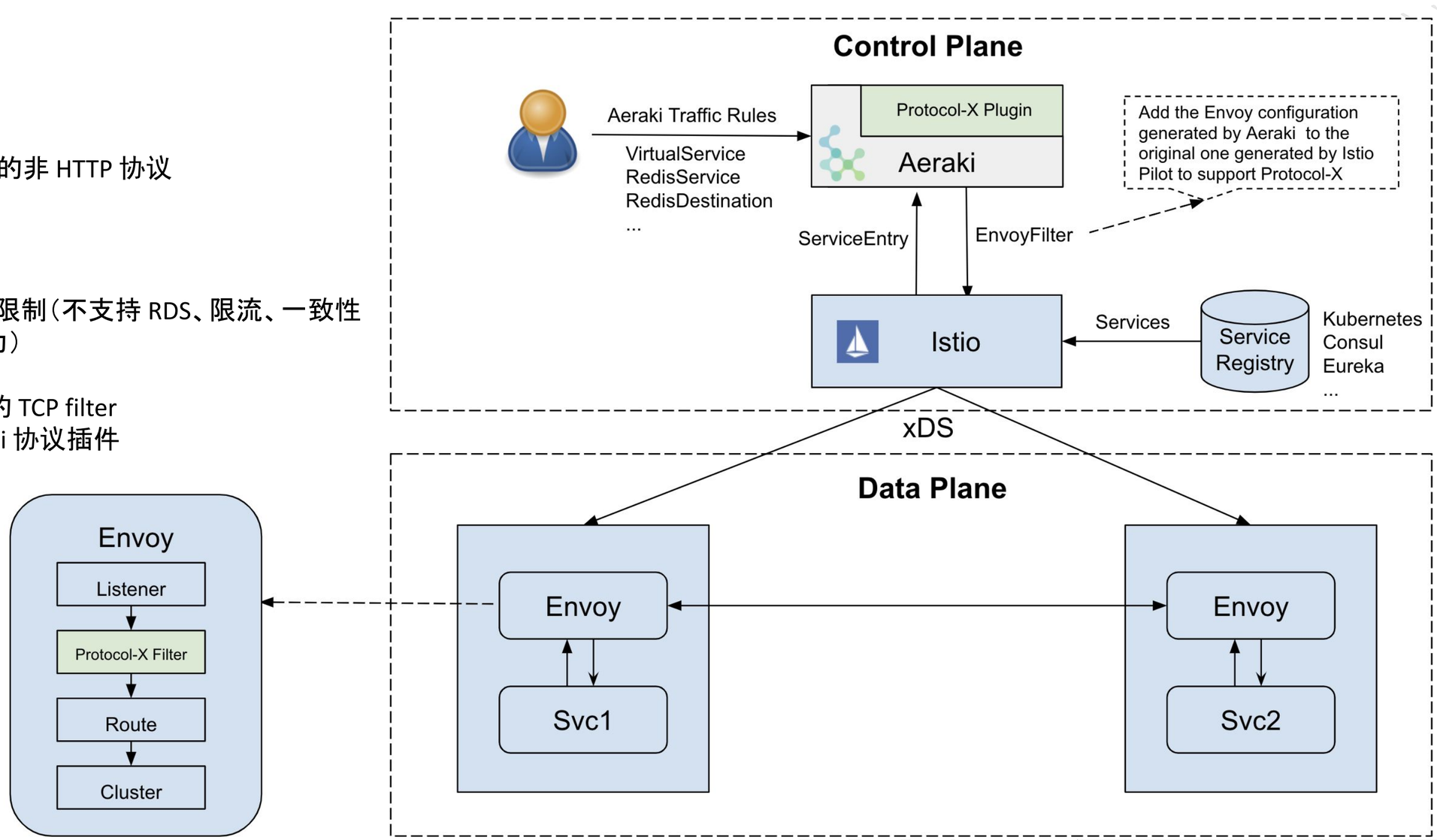
AwesomeRPC Filter
- Decoding/encoding
- Parsing header
- Routing
- Load balancing
- Circuit breaker
- Fault injection
- Telemetry collecting

困难：
- Istio 目前缺少一个良好的协议扩展机制
- Istio 需要理解 Envoy filter 中协议特定的知识
- Istio 代码中维护众多七层协议的代价较大

Aeraki 第一版架构

能力：
- 和 Istio 无缝集成，对 Istio 无侵入
- 可以支持 Envoy 中已经提供实现的非 HTTP 协议（Dubbo，Thrift，Redis等）

问题：
- Envoy Dubbo，Thrift Filter 的功能限制（不支持 RDS、限流、一致性 Hash、流量镜像、调用跟踪等能力）
- 支持一个新的协议工作量很大
  - 数据面需要编写一个完整的 TCP filter
  - 控制面需要开发一个 Aeraki 协议插件

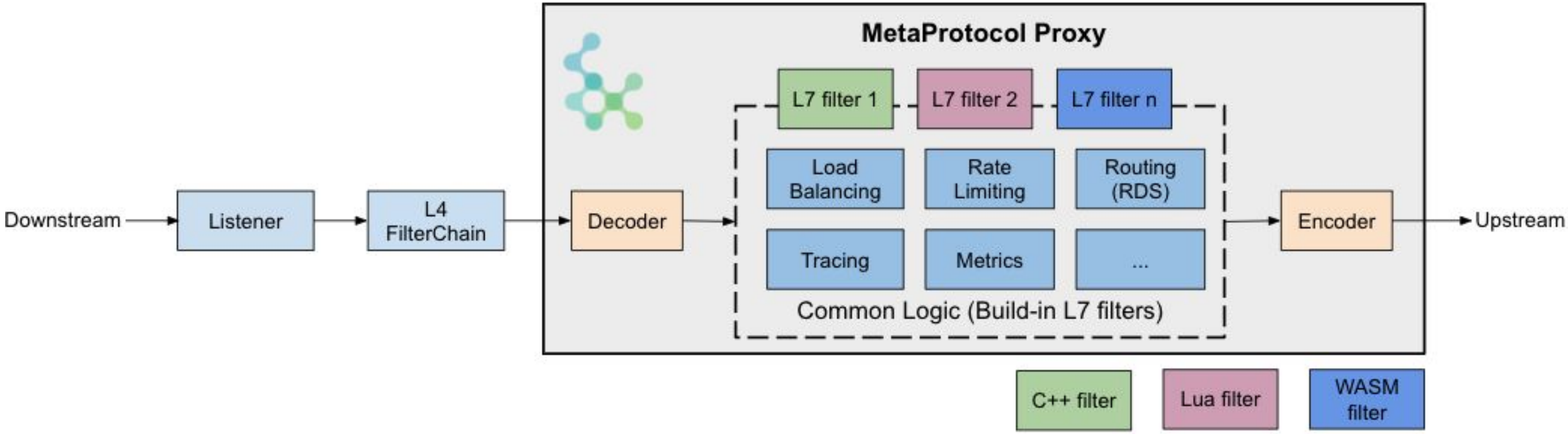


重复造轮子？常见七层协议流量管理的相似之处

Protocol	Destination service	Parameters could be used for routing
HTTP 1.1	host	host, path，method headers
HTTP 2	pseudo header: authority	pseudo header: authority, path，method， headers
gRPC	HTTP 2 path	Request-Headers(Delivered as HTTP2 headers)
TARS	ServantName	ServantName, FuncName, Context
Dubbo	service name	service name, service version, service method
Any RPC Protocol	service name in message header	some key:value pairs in message header

MetaProtocol:基于 Envoy 的七层协议框架

- MetaProtocol Proxy 中实现七层协议的通用逻辑：负载均衡、熔断、动态路由、消息头修改、本地\全局限流、请求指标上报、调用跟踪等。
- 基于 MetaProtocol 实现一个自定义协议时，只需要实现 Decode 和 Encode 扩展点的少量代码（数百行代码）。
- 提供基于 C++、WASM、Lua 的 L7 filter 扩展点，用户可以实现一些灵活的自定义协议处理逻辑，例如认证授权等。



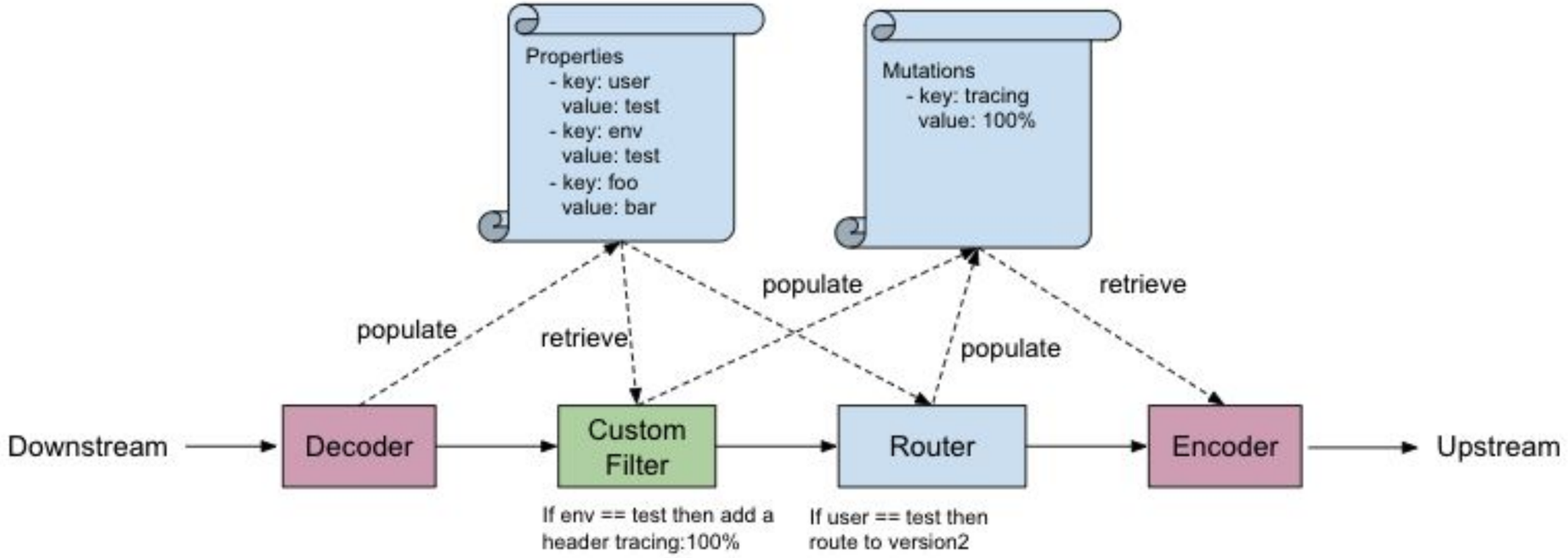
MetaProtocol: 请求处理路径

处理流程：

1. Decoder 解析 Downstream 请求，填充 Metadata
2. L7 filter 从 Metadata 获取所需的数据，进行请求方向的业务处理
3. L7 filter 将需要修改的数据放入 Mutation 结构中
4. Router 根据 RDS 配置的路由规则选择 Upstream Cluster
5. Encoder 根据 Mutation 结构封包
6. 将请求发送给 Upstream

L7 filter 共享数据结构：

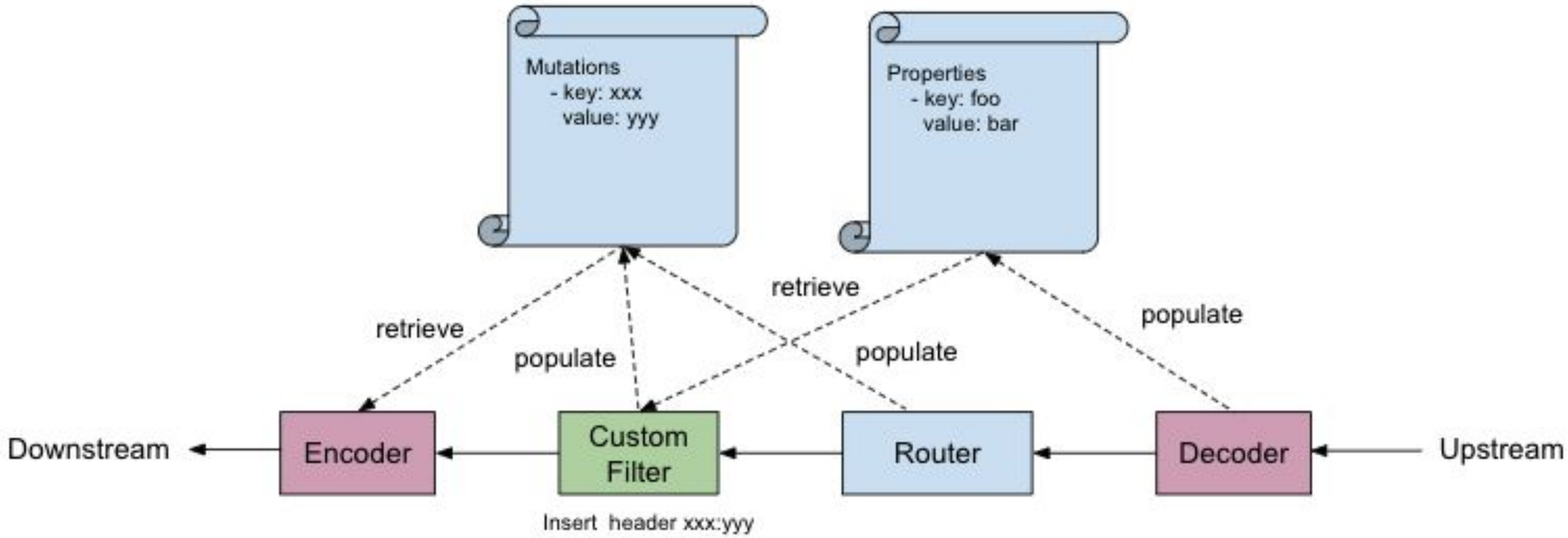
- Metadata： decode 时填充的 key:value 键值对，用于 l7 filter 的处理逻辑中
- Mutation：L7 filter 填充的 key:value 键值对，用于 encode 时修改请求数据包



MetaProtocol: 响应处理路径

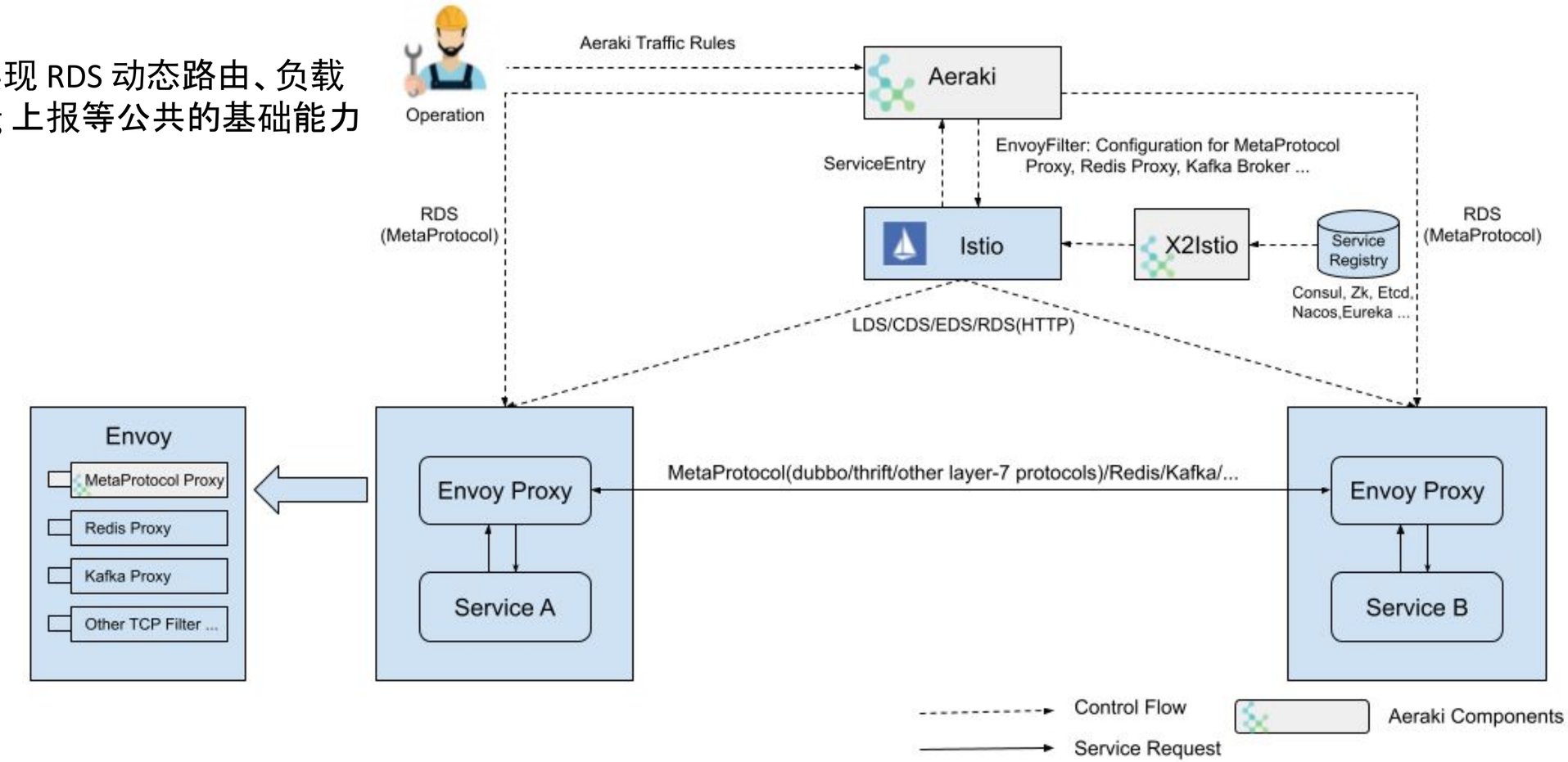
1. 处理流程：
2. Decoder 解析 Upstream 的响应，填充 Metadata
3. Router 根据 connection/stream 对应关系找到响应的 Downstream 连接
4. L7 filter 从 Metadata 获取所需的数据，进行响应方向的业务处理
5. L7 filter 将需要修改的数据放入 Mutation 结构中
6. Encoder 根据 Mutation 结构封包
7. 将响应

- L7 filter 共享数据结构：
- Metadata： decode 时填充的 key:value 键值对，用于 l7 filter 的处理逻辑中
- Mutation：L7 filter 填充的 key:value 键值对，用于 encode 时修改响应数据包



Aeraki 最新架构（Aeraki + MetaProtocol Proxy 双剑合璧 → Aeraki Mesh）

- 控制面：Aeraki + Istio 提供控制面管理，实现按请求 header 路由、灰度发布、地域感知LB、流量镜像等高级流量管理能力。
- 数据面：MetaProtocol Proxy 实现 RDS 动态路由、负载均衡、熔断、Metrics 和 Tracing 上报等公共的基础能力



Aeraki Mesh 项目当前进展

协议支持

- MetaProtocol-Dubbo
- MetaProtocol-Thrift
- MetaProtocol-腾讯音乐私有协议
- MetaProtocol-腾讯融媒体私有协议
- MetaProtocol-腾讯游戏私有协议（接入中）
- Dubbo（Envoy 原生 Filter）
- Thrift （Envoy 原生 Filter）
- Redis （Envoy 原生 Filter）

备注：由于原生 Dubbo 和 Thrift filter 的限 制，不支持 RDS，限流，一致性 Hash LB，调用跟踪等能力。建 议使用 MetaProtocol Proxy 的 Dubbo 和 Thrift 协议能力。

功能特性

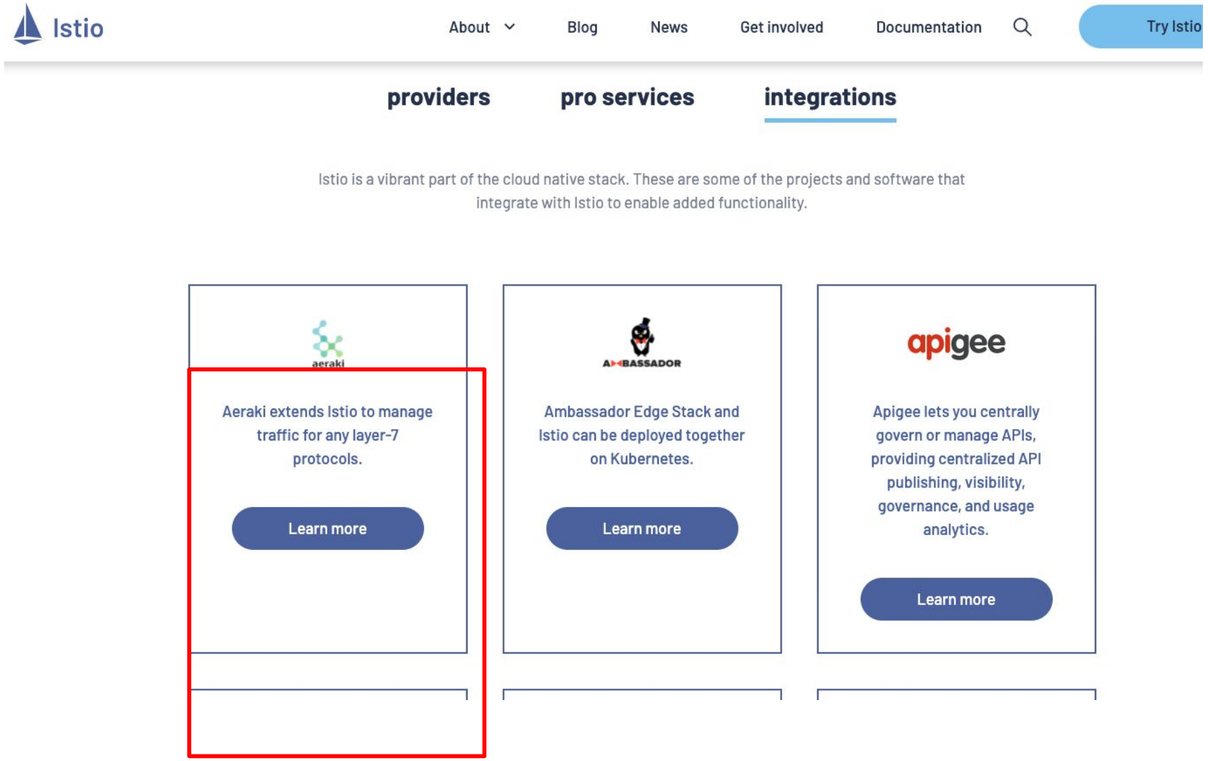
- Request Level Load Balancing/Locality Load Balancing (Supports Consistent Hash/Sticky Session)
- Circuit breaking
- Metadata based flexible Route Match Conditions
- Dynamic route update through Aeraki MetaRDS
- Version Based Routing
- Traffic Splitting
- Local/Global Rate Limit
- Message mutation
- Request level Metrics

产品落地

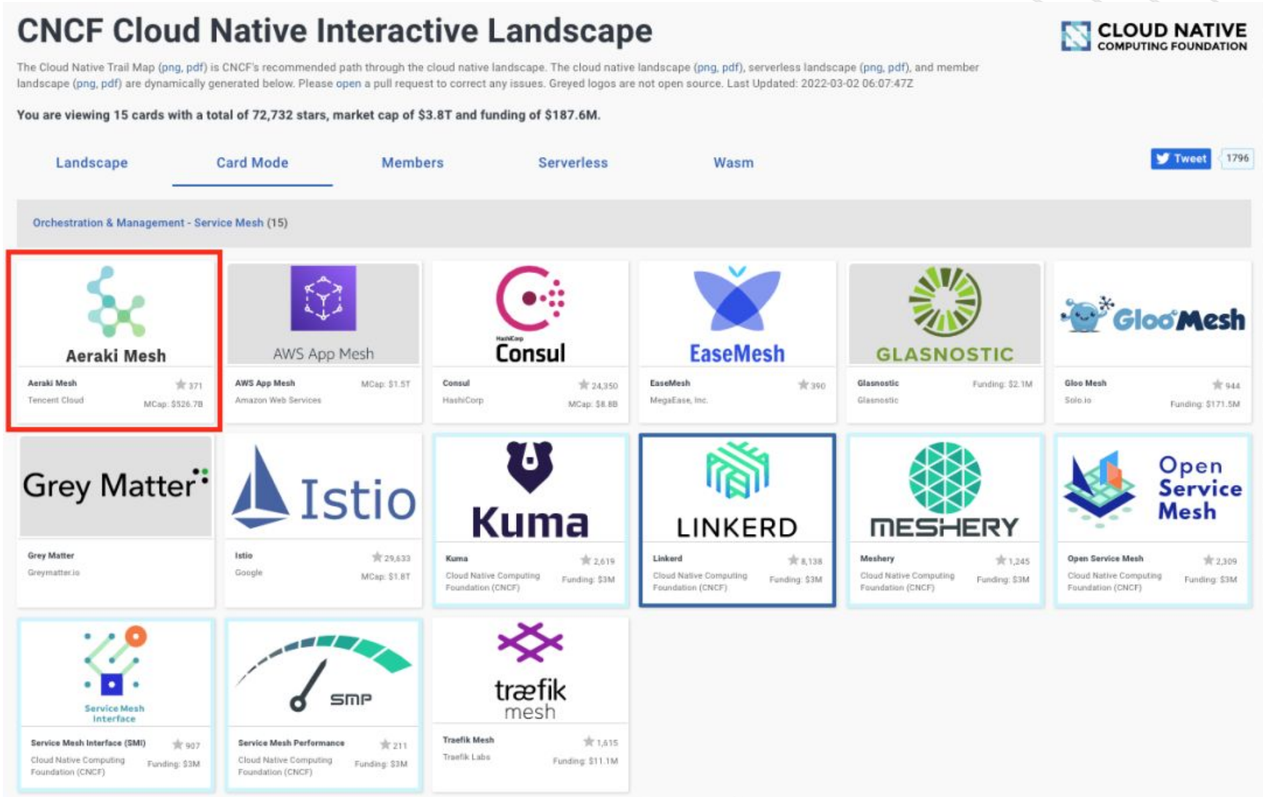
- 腾讯融媒体/央视频 冬奥会视频直播
- 腾讯音乐
- 腾讯游戏王者破晓 (协议接入中）
- 小红书

Aeraki Mesh 开源生态

Istio ecosystem integration 项目



CNCF 云原生全景图 Service Mesh 项目



如何基于 Aeraki Mesh 开发一个自定义协议？

1. 实现编解码接口 codec (约数百行代码)
2. 定义一个 ApplicationProtocol

```
class Codec {
public:
  virtual ~Codec() = default;

  /*
   * decodes the protocol message.
   *
   * @param buffer the currently buffered data.
   * @param metadata saves the meta data of the current message.
   * @return DecodeStatus::DONE if a complete message was successfully consumed,
   * DecodeStatus::WaitForData if more data is required.
   * @throws EnvoyException if the data is not valid for this protocol.
   */
  virtual DecodeStatus decode(Buffer::Instance& buffer, Metadata& metadata) PURE;

  /*
   * encodes the protocol message.
   *
   * @param metadata the meta data produced in the decoding phase.
   * @param mutation the mutation that needs to be encoded to the message.
   * @param buffer save the encoded message.
   * @throws EnvoyException if the metadata or mutation is not valid for this protocol.
   */
  virtual void encode(const Metadata& metadata, const Mutation& mutation,
                      Buffer::Instance& buffer) PURE;

  /*
   * encodes an error message. The encoded error message is us
   * can't find the specified cluster, or there is no healthy
   *
   * @param metadata the meta data produced in the decoding ph
   * @param error the error that needs to be encoded in the me
   * @param buffer save the encoded message.
   * @throws EnvoyException if the metadata is not valid for t
   */
  virtual void onError(const Metadata& metadata, const Error&
};
```

```yaml
apiVersion: metaprotocol.aeraki.io/v1alpha1
kind: ApplicationProtocol
metadata:
  name: my-protocol
  namespace: istio-system
spec:
  protocol: my-protocol
  codec: aeraki.meta_protocol.codec.my_protocol
```

自定义协议项目模板: https://github.com/aeraki-mesh/meta-protocol-awesomerpc

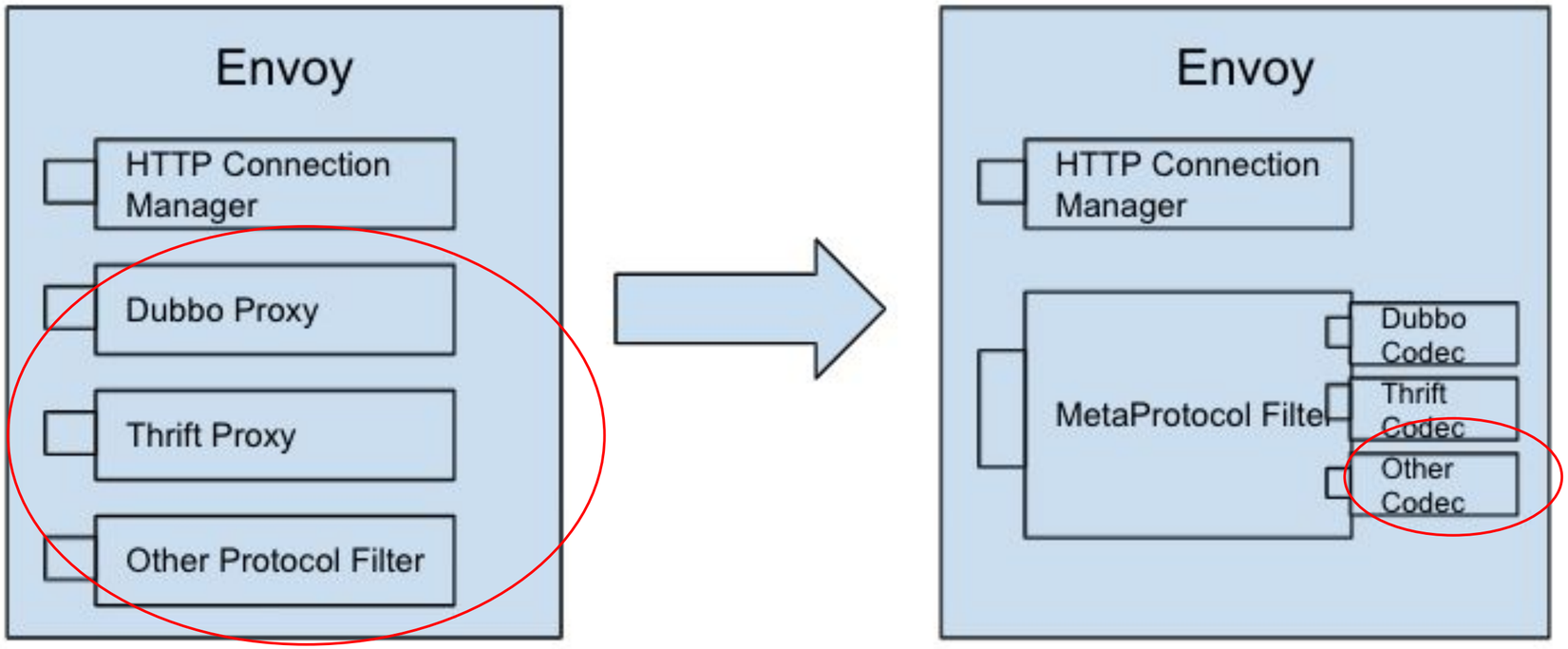
在 Service Mesh 中管理一个私有协议的所需工作量对比(采用/不采用 Aeraki Mesh)

数据面的工作量:

- 不采用 Aeraki Mesh:巨大的工作量,需要编写一个完整的 Envoy L4 filter
- 采用 Aeraki Mesh:很少的工作量:只需要实现 codec 接口(通常数百行代码)

控制面的工作量:

- 不采用 Aeraki Mesh:巨大的工作量,需要专门为该协议编写一个控制面(基于 Istio 魔改或者从头编写)
- 采用 Aeraki Mesh:工作量为零,Aeraki 可作为任何基于 MetaProtocol 的协议的控制面



流量管理示例

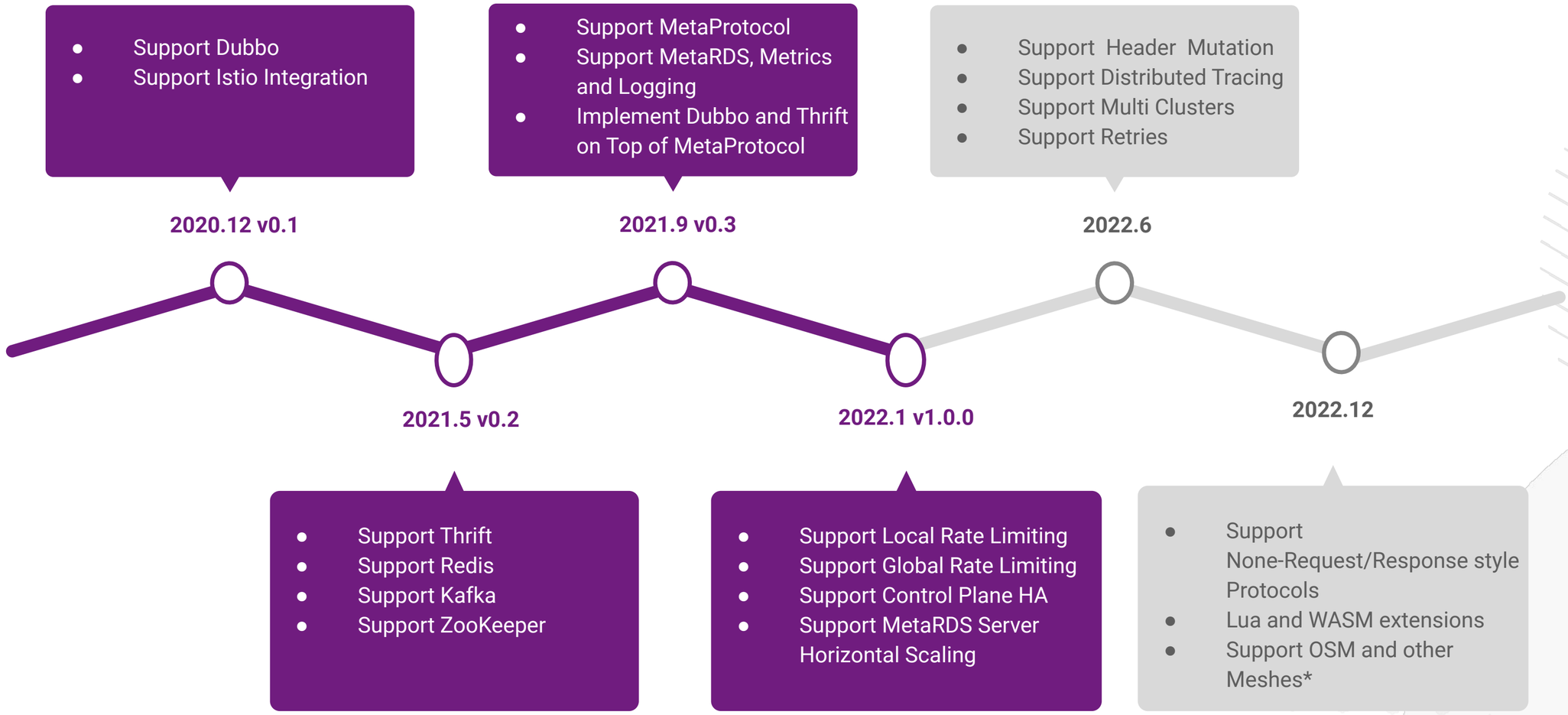
权重路由

```yaml
apiVersion: metaprotocol.aeraki.io/v1alpha1
kind: MetaRouter
metadata:
  name: test-metaprotocol-qza-route
  namespace: meta-qza
spec:
  hosts:
    - qza-sample-server.meta-qza.svc.cluster.local
  routes:
    - name: traffic-split
      match:
        attributes:
          cmd:
            exact: "153"
          sub_cmd:
            regex: "1|2"
      route:
        - destination:
            host: qza-sample-server.meta-qza.svc.cluster.local
            subset: v1
          weight: 20
        - destination:
            host: qza-sample-server.meta-qza.svc.cluster.local
            subset: v2
          weight: 80
```

本地限流

```yaml
apiVersion: metaprotocol.aeraki.io/v1alpha1
kind: MetaRouter
metadata:
  name: test-metaprotocol-thrift-route
  namespace: meta-thrift
spec:
  hosts:
  - thrift-sample-server.meta-thrift.svc.cluster.local
  localRateLimit:
    tokenBucket:
      fillInterval: 60s
      maxTokens: 5
      tokensPerFill: 5
    conditions:
    - tokenBucket:
        fillInterval: 10s
        maxTokens: 2
        tokensPerFill: 2
      match:
        attributes:
          method:
            exact: sayHello
~
~
~
```

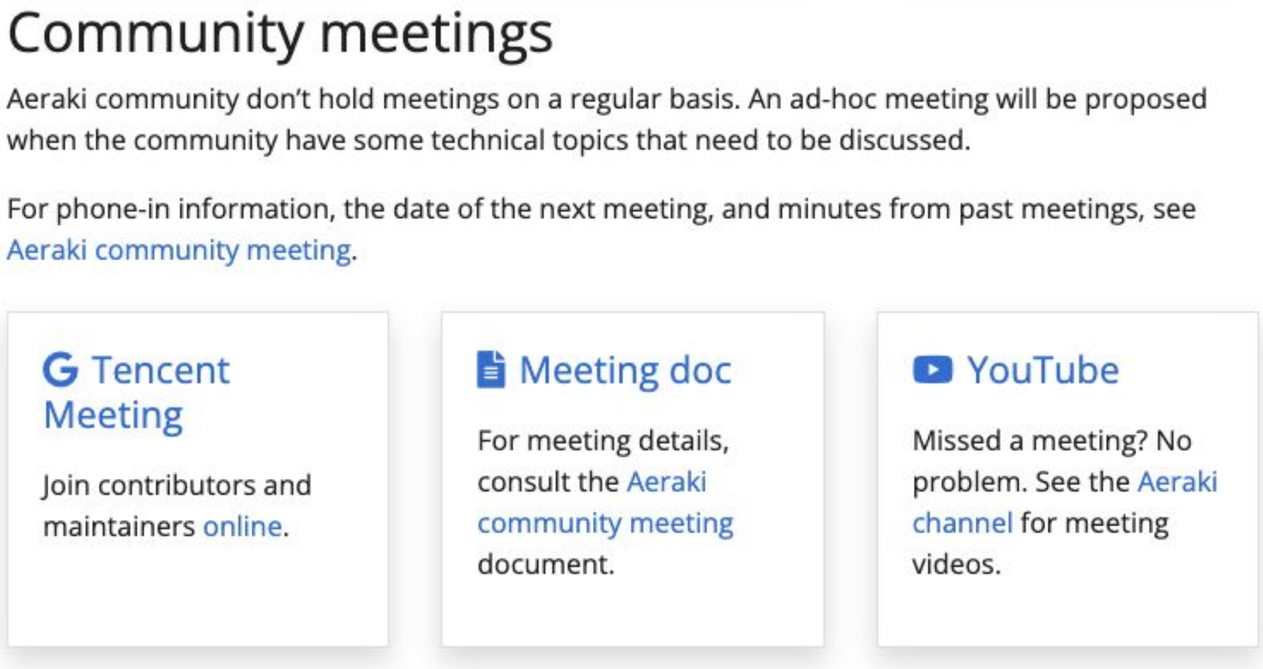
Aeraki Mesh 项目路标



* If they support extensibility mechanisms like Istio EnvoyFilter

如何参与社区？

参与社区会议：https://www.aeraki.net/zh/community/#community-meetings



参与微信群：请微信联系 zhao_huabing 进群

Aeraki Mesh 官网：https://www.aeraki.net
Aeraki Mesh Github: https://github.com/aeraki-mesh

Demo 及产品落地实践

腾讯云

Aeraki Mesh在冬奥会中的落地

腾讯 · 覃士林

个人简介

目前担任腾讯后台开发

推动腾讯融媒体产业全面上云

主导微服务化体系建设，推动传统运维方式向云原生转型

目录

01.官方媒体 流量大受众广

承担国内外大型赛事、政治新闻、明星热点、热播影视剧在移动端向全球网民的稳定输出

02.业务复杂 业务角度

语言、协议、框架、基础组件、开发团队繁多，异构化严重

开发语言：C++/Java/Go/Python/Rust等

协议类型：HTTP/tRPC/Videopacket/rpc等

03.难以维护 运维角度

规模体量大

靠传统运维手段，已经难以支撑如此大的业务体量

业务逻辑不透明

难以支撑对服务状态服务质量的把控

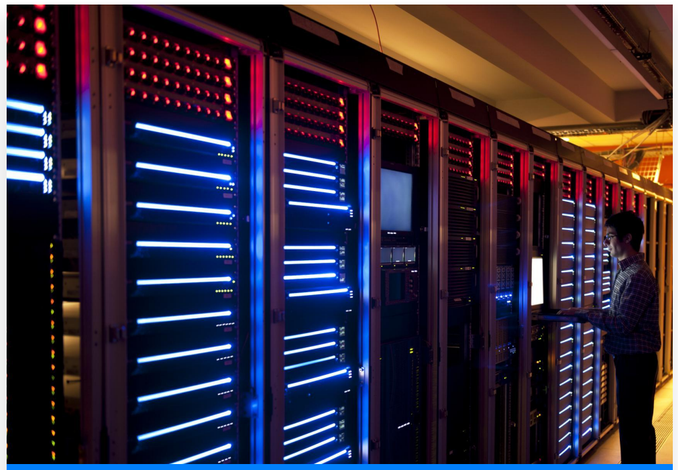
热点事件突发

突发流量存在业务宕机隐患

缺乏合理的流量管控

缺乏分布式场景流量调度、管控能力

04.需求梳理 技术角度



业务类型丰富

无法做到统一语言、协议、框架（业务规模较小的企业可以往这方面考虑）

上云作为基本面

随着上云的持续推进，已经很大程度的解决了因扩缩容等资源、环境问题造成的复杂度。因此全量上云势在必行

补齐治理能力

过载保护（限流、熔断、降级）、合理的请求重试机制、灵活的请求路由策略、基于流量侧的灰度发布能力（金丝雀、蓝绿）等

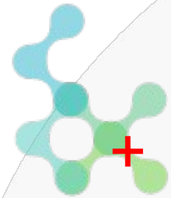
基本设计原则：

1、无侵入业务（硬性要求，开发自身任务重时间紧，不应在此花费更大的精力）

05. 技术选型 落地尝试

云上运行 + 服务治理功能 + 服务状态监测 + 不侵入业务 = ？

云原生体系 + 通用性可扩展性 + 可观测能力 + 开发无感知 =



Aeraki Mesh

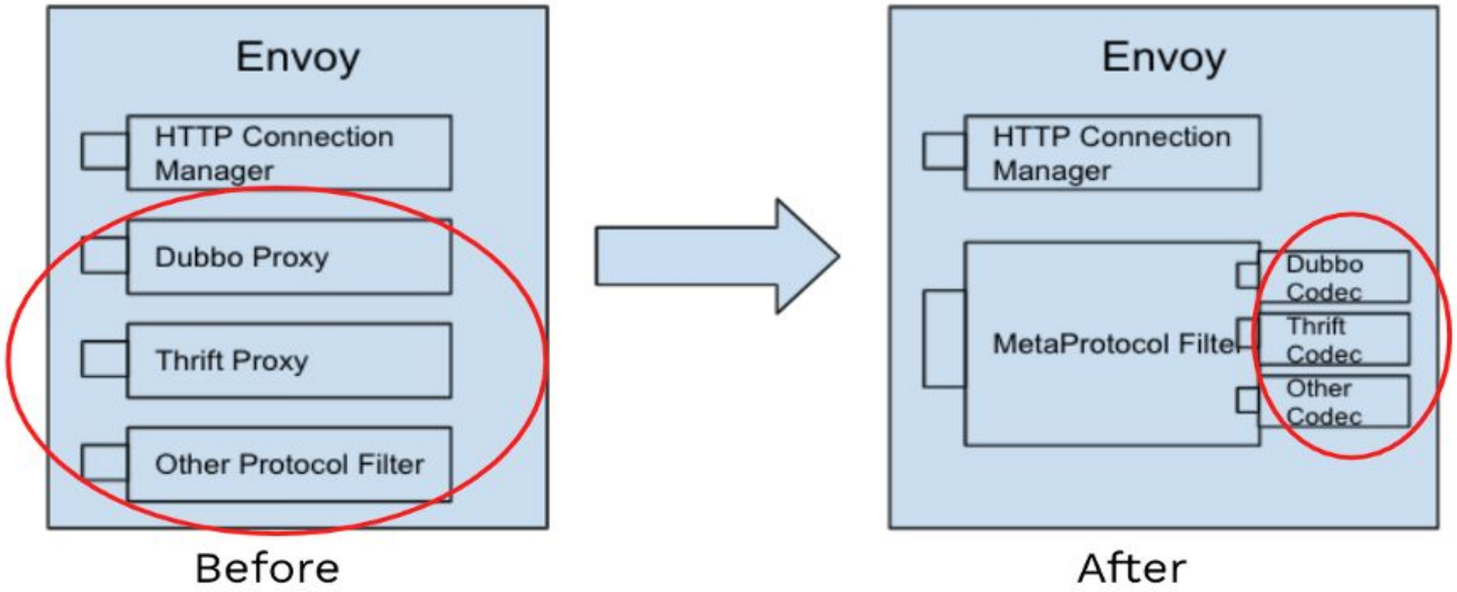
+

Istio: 已成为Service Mesh领域的事实标准

Aeraki：基于Istio对任意七层协议管控的支持

目录

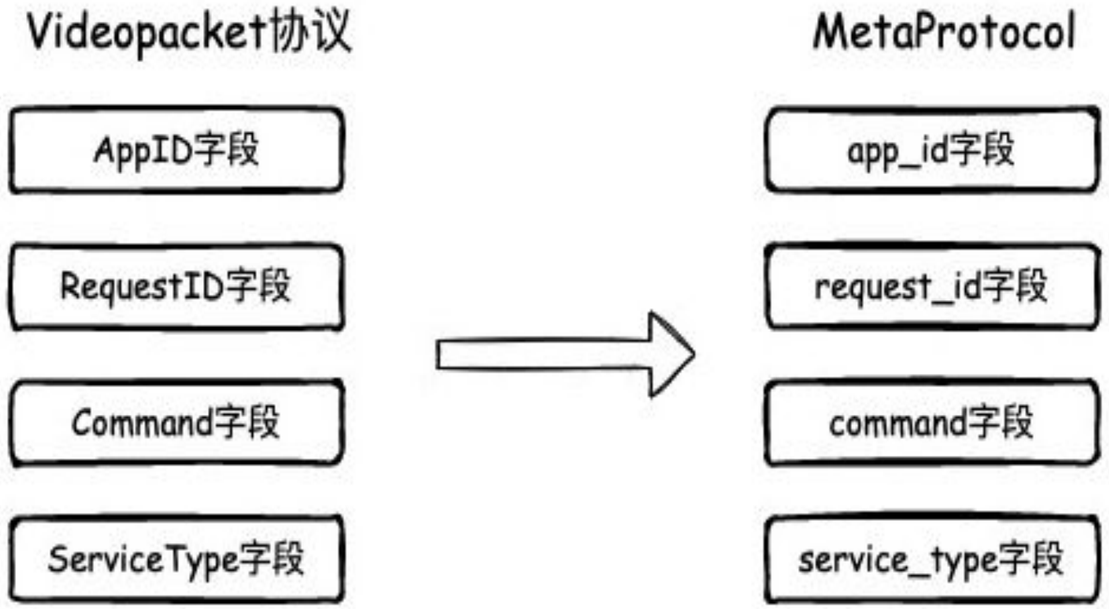
协议扩展

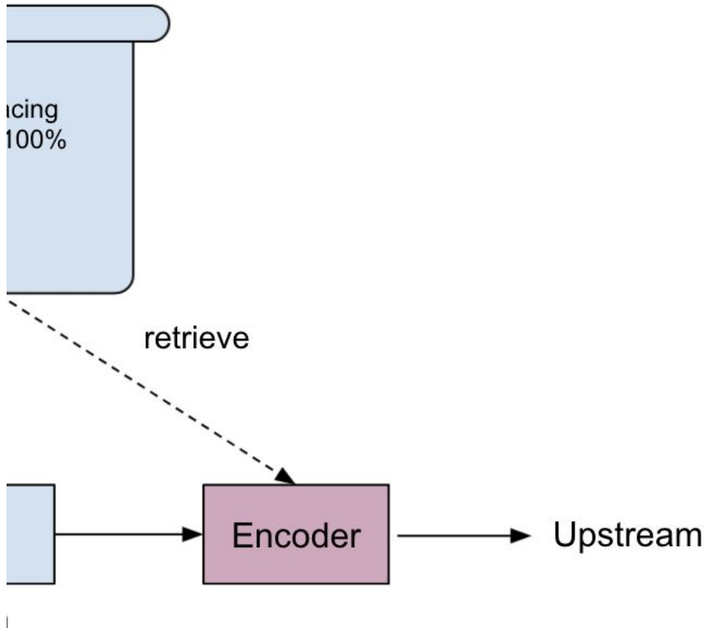


在Aeraki之前，社区的做法是为每种私有协议新增一个过滤器，所有协议的服务治理功能均需要各自实现一遍。

协议扩展

- 对Videopacket协议进行解码
- 将解码字段映射为MetaProtocol字段





协议扩展

```
namespace Envoy {
namespace Extensions {
namespace NetworkFilters {
namespace MetaProtocolProxy {
namespace VideoPacket {

/**
 * Codec for videopacket protocol
 *
 */
class VideoPacketCodec : public MetaProtocolProxy::Codec, public Logger::Loggable<Logger::Id::misc> {
public:
  ~VideoPacketCodec() override = default;

  MetaProtocolProxy::DecodeStatus decode(Buffer::Instance& buffer,
                                         MetaProtocolProxy::Metadata& metadata) override;
  void encode(const MetaProtocolProxy::Metadata& metadata,
              const MetaProtocolProxy::Mutation& mutation, Buffer::Instance& buffer) override;
  void onError(const MetaProtocolProxy::Metadata& metadata, const MetaProtocolProxy::Error& error,
               Buffer::Instance& buffer) override;

private:
 void toMetadata(CVideoPacket& cvp, MetaProtocolProxy::Metadata& metadata, Buffer::Instance& buffer);
private:
 //CVideoPacket req_packet_;
};

}   // namespace VideoPacket
}   // namespace MetaProtocolProxy
}   // namespace NetworkFilters
}   // namespace Extensions
}   // namespace Envoy
```

只需要实现：

1、解码

2、编码

3、请求错误处理

目录

接入场景

过载保护，依托限流、熔断功能。

本地限流接入

100%

本地限流保护服务不被瞬时突增流量击穿，所有服务均已配置。

推荐使用方式：

压测平台压测不同应用 -> 根据容量模型预测容量 -> 配置本地限流阈值

全局限流接入

3条业务线

全局限流保护某条请求链路的整体稳定性。

熔断接入

100%

熔断保护客户端总是拿到健康的服务端实例节点。

接入场景

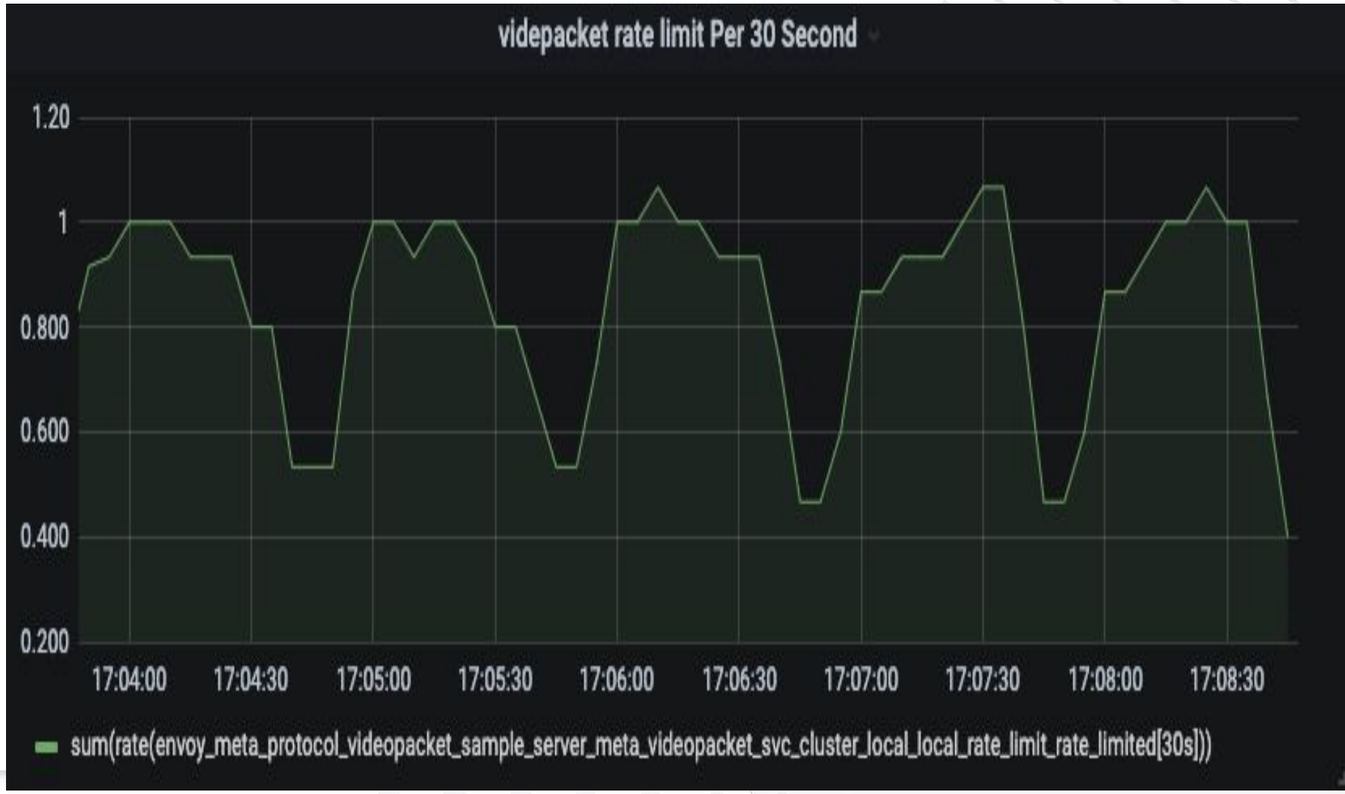
过载保护，本地限流与服务扩容联动。

这样实现的好处：1、与传统负载监控指标相比更贴近于业务本身

2、该值在高并发场景下优先于负载监控上报的发生（感知更灵敏）

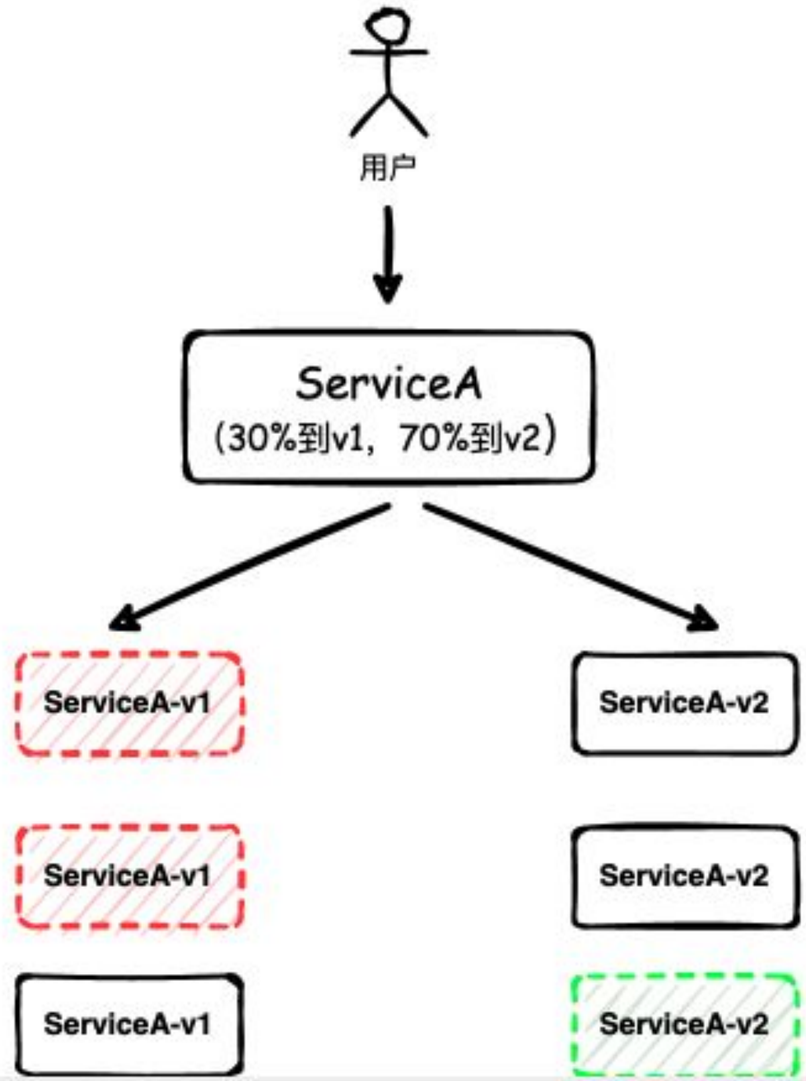
图中展示的是限流与弹性扩缩容组件联动

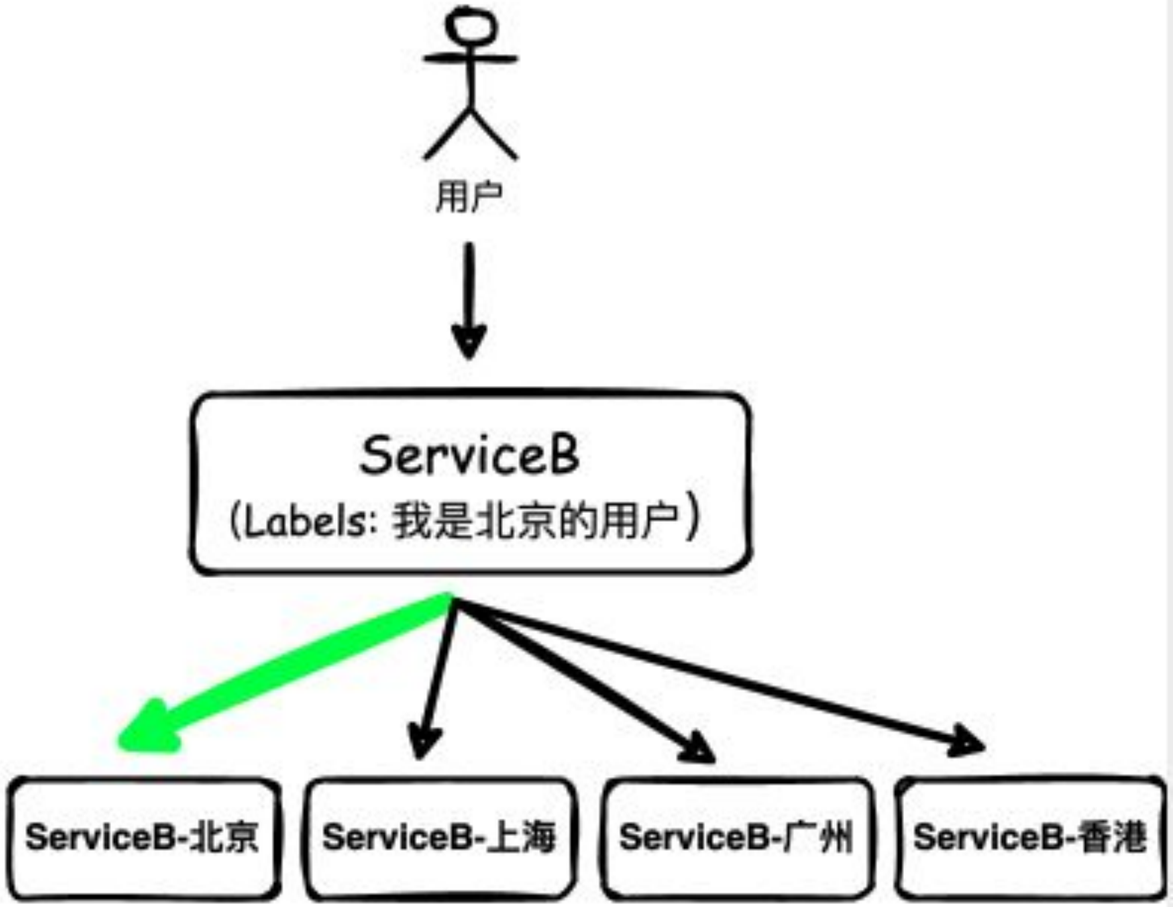
图中展示的是每30秒的限流频次



接入场景

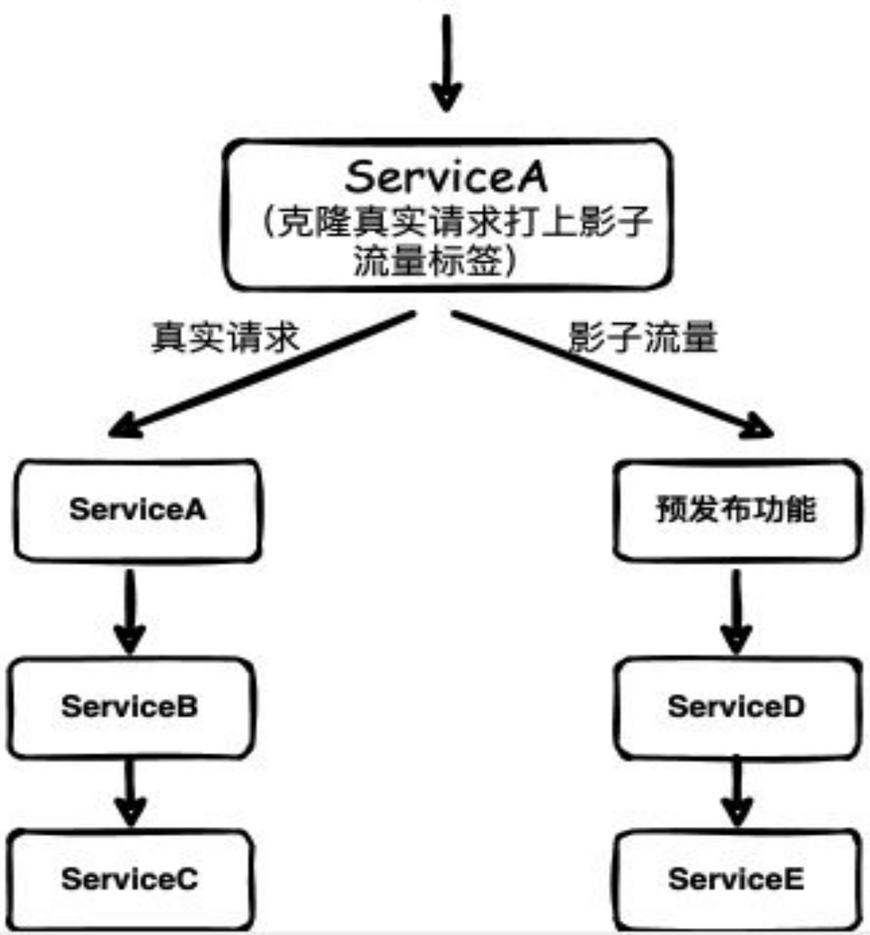
灰度发布、就近访问，依托流量控制功能。





接入场景

基于真实请求灰度未发布的新功能，依托请求头修改、流量镜像功能。



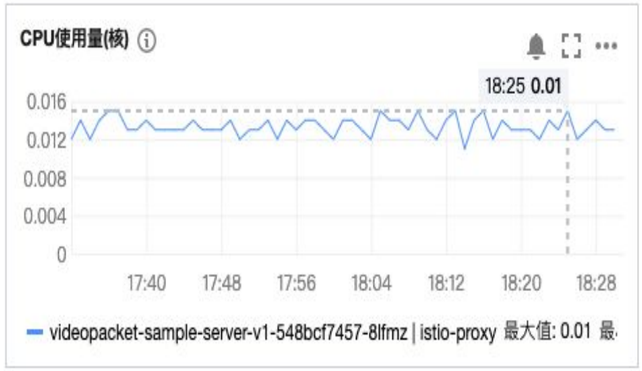
目录

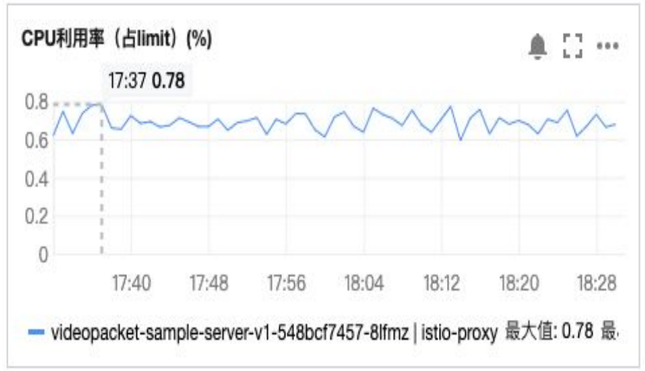
总结及规划

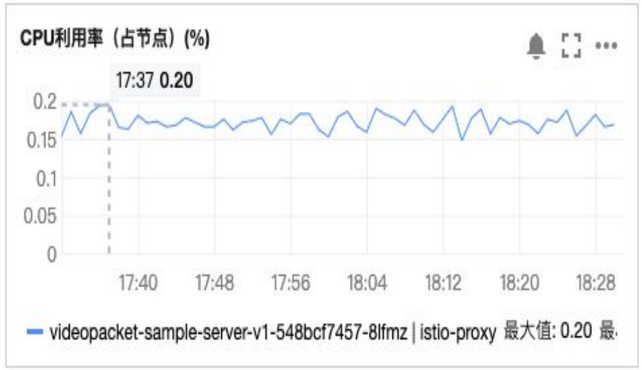
功能层面网格的确收益良多，那业务性能呢？

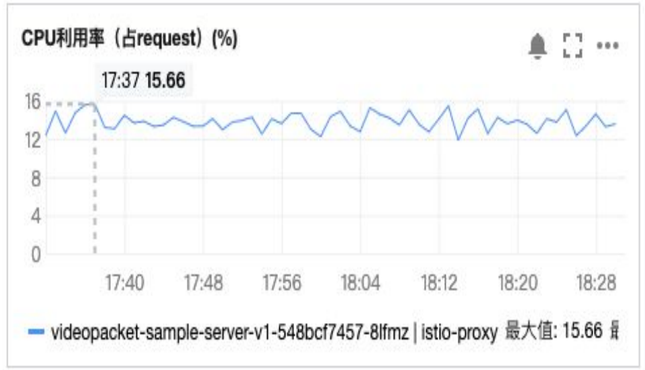
CPU/MEM维度：随着硬件的高速发展硬件资源早已不是瓶颈，价格也已亲民。但开源节流也是人之常情。

CPU

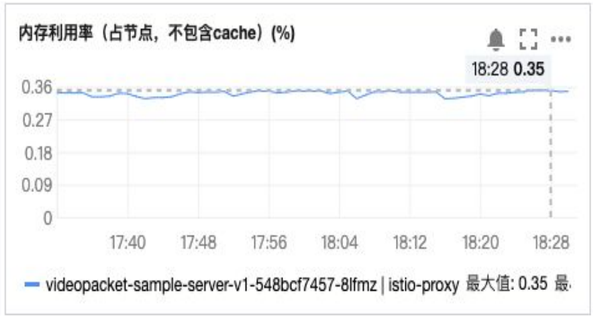


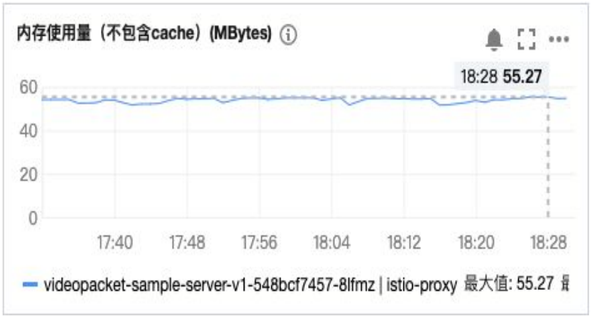


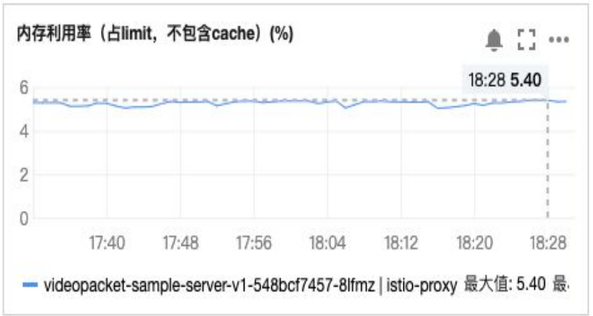




内存



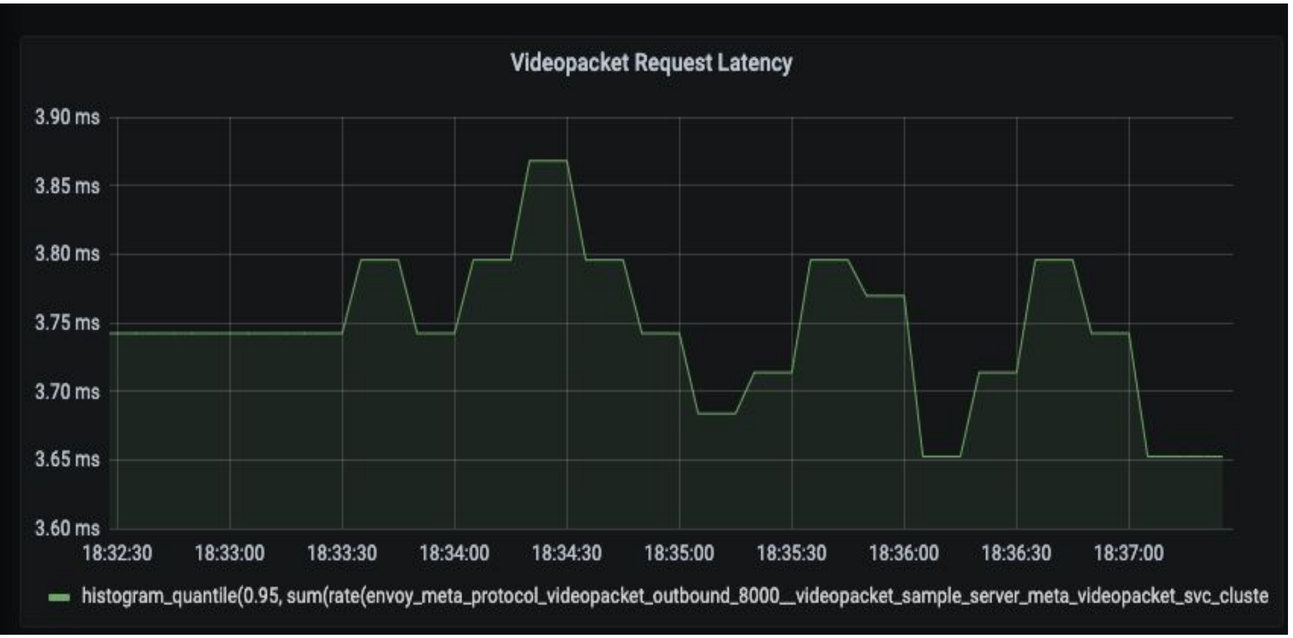




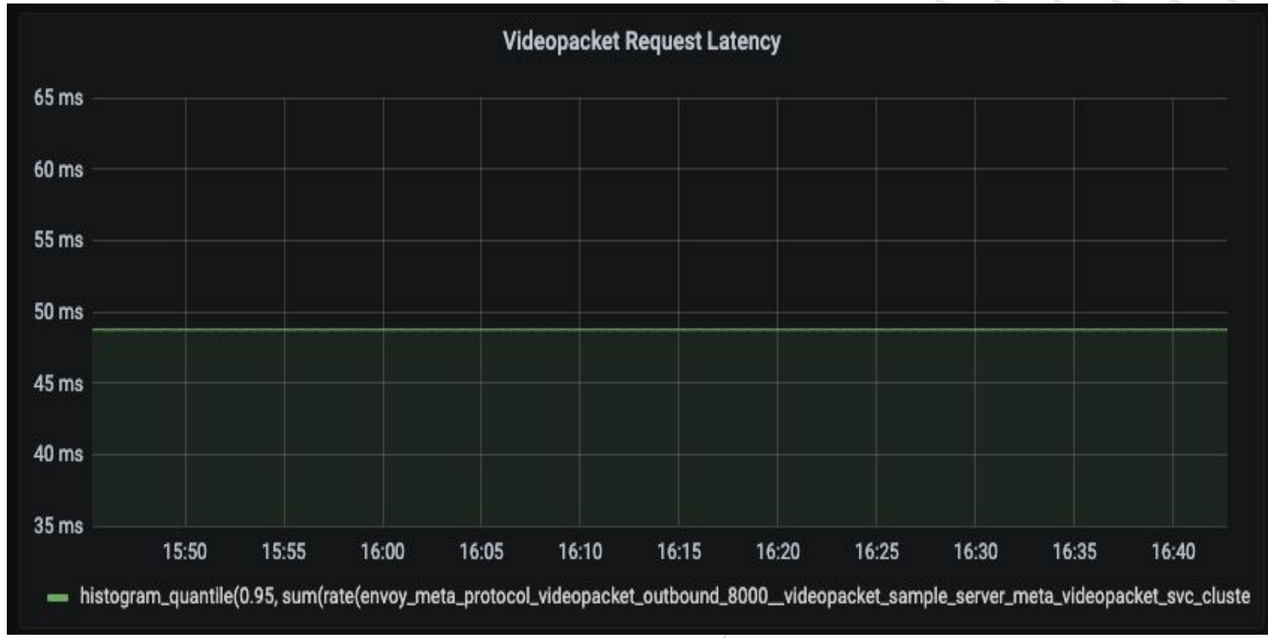
总结及规划

性能层面，大家最关心的是代理层增加的请求耗时？

20字节请求耗时



12万字节请求耗时



Videopacket协议请求耗时：

1、20字节代理层增加耗时：**1~6ms**

2、12万字节代理层增加耗时：**15~30ms**

注意：

1、单纯谈请求耗时并没有意义

2、耗时跟协议的编解码性能有关，性能优的协议耗时相对少

3、跟请求体的大小正相关，这一点从图中也可以看出来

总结及规划

可观测性度量

丰富指标提供观测

Prometheus Alerts Graph Status Help Classic UI

envoy_meta_protocol_**videopacket**_inbound_8000___request_time_ms_sum
envoy_meta_protocol_**videopacket**_inbound_8000___request_twoway
envoy_meta_protocol_**videopacket**_inbound_8000___response
envoy_meta_protocol_**videopacket**_inbound_8000___response_business_exception
envoy_meta_protocol_**videopacket**_inbound_8000___response_decoding_error
envoy_meta_protocol_**videopacket**_inbound_8000___response_decoding_success
envoy_meta_protocol_**videopacket**_inbound_8000___response_error
envoy_meta_protocol_**videopacket**_inbound_8000___response_error_caused_connection_close
envoy_meta_protocol_**videopacket**_inbound_8000___response_success
envoy_meta_protocol_**videopacket**_outbound_8000__videopacket_sample_server_foo_svc_cluster_local_cx_destroy_local_with_active_rq
envoy_meta_protocol_**videopacket**_outbound_8000__videopacket_sample_server_foo_svc_cluster_local_cx_destroy_remote_with_active_rq
envoy_meta_protocol_**videopacket**_outbound_8000__videopacket_sample_server_foo_svc_cluster_local_local_response_business_exception
envoy_meta_protocol_**videopacket**_outbound_8000__videopacket_sample_server_foo_svc_cluster_local_local_response_error
envoy_meta_protocol_**videopacket**_outbound_8000__videopacket_sample_server_foo_svc_cluster_local_local_response_success
envoy_meta_protocol_**videopacket**_outbound_8000__videopacket_sample_server_foo_svc_cluster_local_rds_videopacket_sample_server_foo_svc_cluster_local_8000_config_reload
envoy_meta_protocol_**videopacket**_outbound_8000__videopacket_sample_server_foo_svc_cluster_local_rds_videopacket_sample_server_foo_svc_cluster_local_8000_config_reload_time_ms
envoy_meta_protocol_**videopacket**_outbound_8000__videopacket_sample_server_foo_svc_cluster_local_rds_videopacket_sample_server_foo_svc_cluster_local_8000_control_plane_connected_state
envoy_meta_protocol_**videopacket**_outbound_8000__videopacket_sample_server_foo_svc_cluster_local_rds_videopacket_sample_server_foo_svc_cluster_local_8000_control_plane_pending_requests
envoy_meta_protocol_**videopacket**_outbound_8000__videopacket_sample_server_foo_svc_cluster_local_rds_videopacket_sample_server_foo_svc_cluster_local_8000_control_plane_rate_limit_enforced
envoy_meta_protocol_**videopacket**_outbound_8000__videopacket_sample_server_foo_svc_cluster_local_rds_videopacket_sample_server_foo_svc_cluster_local_8000_init_fetch_timeout
envoy_meta_protocol_**videopacket**_outbound_8000__videopacket_sample_server_foo_svc_cluster_local_rds_videopacket_sample_server_foo_svc_cluster_local_8000_update_attempt
envoy_meta_protocol_**videopacket**_outbound_8000__videopacket_sample_server_foo_svc_cluster_local_rds_videopacket_sample_server_foo_svc_cluster_local_8000_update_duration_bucket
envoy_meta_protocol_**videopacket**_outbound_8000__videopacket_sample_server_foo_svc_cluster_local_rds_videopacket_sample_server_foo_svc_cluster_local_8000_update_duration_count
envoy_meta_protocol_**videopacket**_outbound_8000__videopacket_sample_server_foo_svc_cluster_local_rds_videopacket_sample_server_foo_svc_cluster_local_8000_update_duration_sum
envoy_meta_protocol_**videopacket**_outbound_8000__videopacket_sample_server_foo_svc_cluster_local_rds_videopacket_sample_server_foo_svc_cluster_local_8000_update_empty
envoy_meta_protocol_**videopacket**_outbound_8000__videopacket_sample_server_foo_svc_cluster_local_rds_videopacket_sample_server_foo_svc_cluster_local_8000_update_failure
envoy_meta_protocol_**videopacket**_outbound_8000__videopacket_sample_server_foo_svc_cluster_local_rds_videopacket_sample_server_foo_svc_cluster_local_8000_update_rejected
envoy_meta_protocol_**videopacket**_outbound_8000__videopacket_sample_server_foo_svc_cluster_local_rds_videopacket_sample_server_foo_svc_cluster_local_8000_update_success
envoy_meta_protocol_**videopacket**_outbound_8000__videopacket_sample_server_foo_svc_cluster_local_rds_videopacket_sample_server_foo_svc_cluster_local_8000_update_time

总结及规划

后续规划

核心业务完全铺开

100%

基于网格tracing接入

基于应用安全的接入

基于网格Web化能力的开发

我是腾小云

一起感受腾讯云容器魅力

加我进入云原生技术交流群

扫描左边👈二维码

继续交流

扫描右边👉二维码

互动答疑

【云原生正发声】互动提问区

扫一扫二维码打开或分享给好友

- 腾讯文档 -

可多人实时在线编辑，权限安全可控

THANK YOU!

感谢聆

听！