

华为云服务网格数据面演进与实践

张伟 华为云 ASM服务网格架构师

华为云服务网格数据面演进 与实践

讲师简介



张伟

ASM服务网格架构师

“

《Istio权威指南》第二版核心作者。18年开发经验，先后就职于亿阳信通、北电、甲骨文、Polycom、阿里巴巴及华为等公司，作为核心开发人员开发过传输网管系统、Tuxedo交易中间件、ts-server多媒体转码服务、GTS（seata）高性能事务云服务、SC高性能注册中心、ASM数据面等多个产品，有丰富的架构设计及开发经验，现主要负责华为云ASM服务网格数据面代理产品的方案设计及开发工作。

”

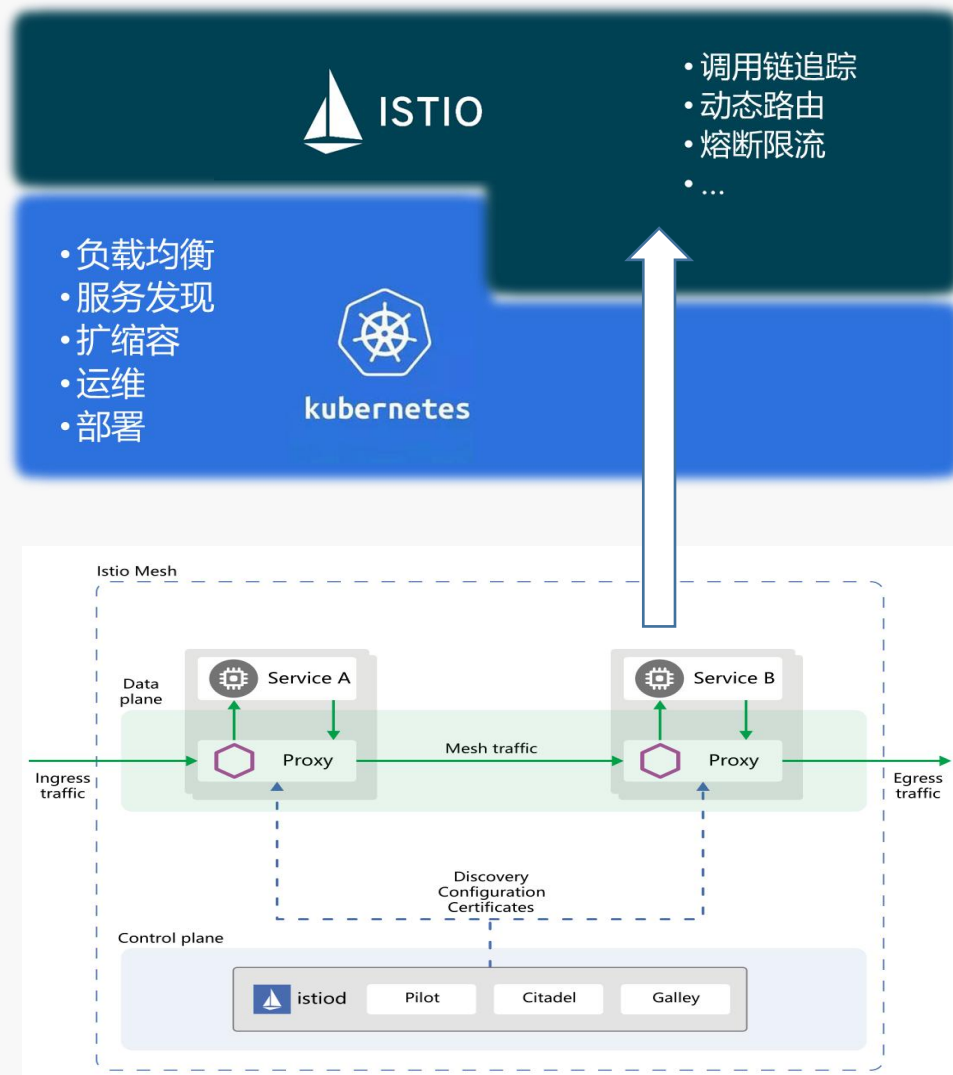
目录

- 网格数据面运维增强功能介绍
- ASM网格数据面的演进路线

Istio + Kubernetes: 云原生应用治理 + 云原生应用设施

服务治理

部署运维



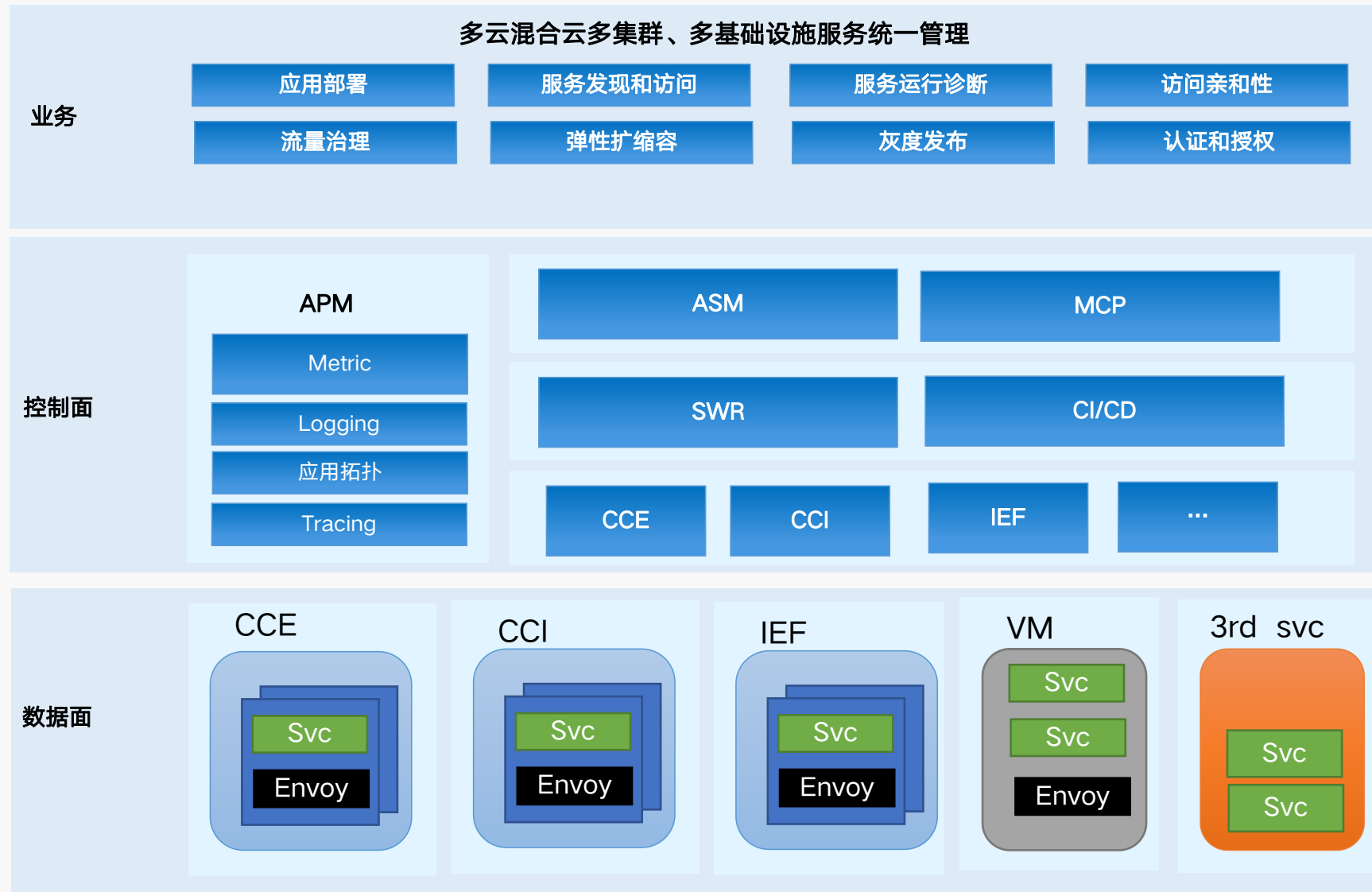
核心理念:

1. 非侵入式Sidecar注入技术，将数据面组件注入到应用所在的容器，通过劫持应用流量来进行功能实现，应用无感知。
2. 北向API基于K8s CRD实现，完全声明式，标准化。
3. 数据面与控制面通过xDS gRPC标准化协议通信，支持订阅模式。

核心特性:

1. 服务&流量治理：熔断，故障注入，丰富的负载均衡算法，限流，健康检查，灰度发布，蓝绿部署等
2. 流量与访问可视化：提供应用级别的监控，分布式调用链，访问日志等
3. 安全连接：通过mTLS、认证、鉴权等安全措施帮助企业在零信任的网络中运行应用

华为云ASM，全托管的应用服务网格基础设施

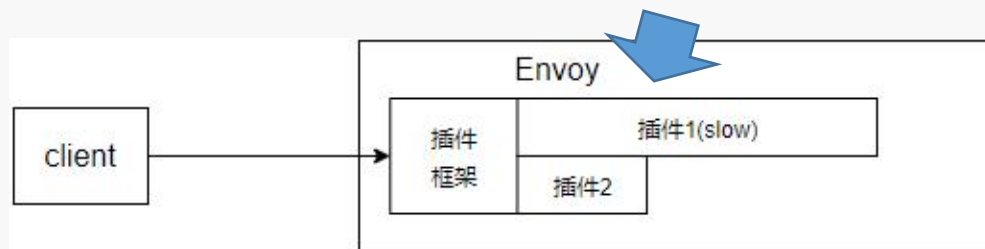
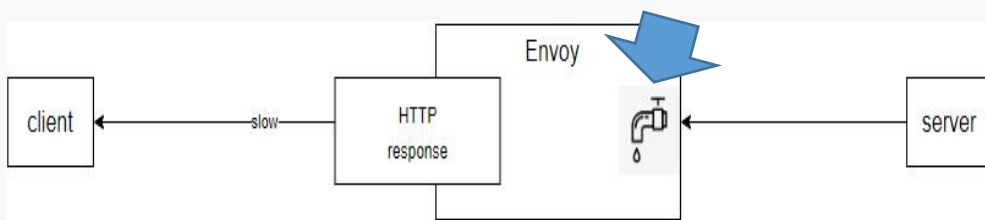
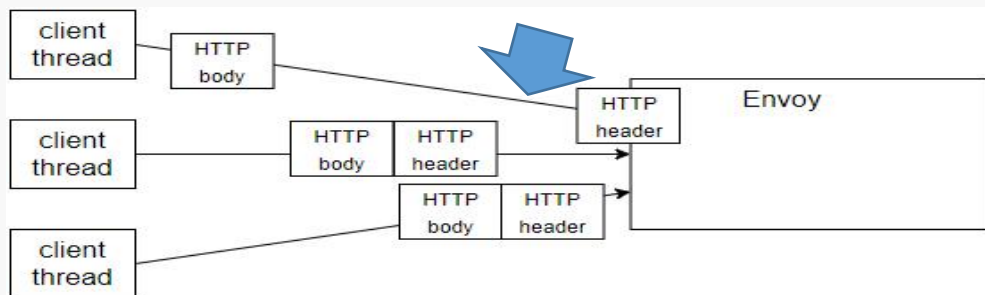


- 覆盖全应用形态，支持容器、虚机、边缘、传统微服务、第三方服务等多种应用平滑接入、统一治理。支持多云、混合云复杂场景多种网络条件下服务跨集群流量混合管理；大规模网格；提供智能运维、智能扩缩容，帮助用户自动透明的管理应用访问
- 高性能、低损耗、轻量、多形态网格数据面，支持每POD、每Node形态，加快Sidecar转发效率，优化网格控制面资源。

亮点介绍

- ASM产品在运维能力上针对现有Envoy进行了增强，可以对一些普遍由于引入代理的时延增加类问题快速定位。
- ASM产品的发展路线从Sidecar和节点模式NodeProxy逐渐演进到下一代L4/L7分离式架构。这一演进旨在提升用户资源利用率的同时，实现更快的L4层快速路径数据治理以及更具弹性的L7层远程处理能力。

案例背景：常见的时延问题



问题基本描述：

1. 多个客户线程同时访问Envoy，TCP分片或应用发送消息分片使得同一个应用消息处理时间拉长，而且不一定可以稳定重现。
2. 客户端处理响应较慢，导致Envoy上游读服务端响应水位被关闭，造成整体请求处理时间较长。
3. Envoy内用户自定义插件编写不当，导致请求整体处理时间较长。

案例背景

1. 客户线上服务支持用户使用HTTP请求上传文档，文档本身10k左右，有时延达到秒级。通过Envoy AccessLog只能看出Envoy内时延较高，但不知道在哪个阶段引起。
2. 此为线上偶现问题，再尝试上传相同文档则在几十毫秒内完成传输。
3. 从OS层面观察内核数据发送无延迟，HTTP数据量本身不大，不应在不同调用时呈现较大的时延差异。
4. 测试环境下无法稳定重现，此问题定位最终耗时几周。
5. 最后判断由于Istio ingressgateway网关处理较大并发时，用户消息被分片处理，且Envoy事件处理不及时导致完整HTTP完成处理较慢。

问题与挑战

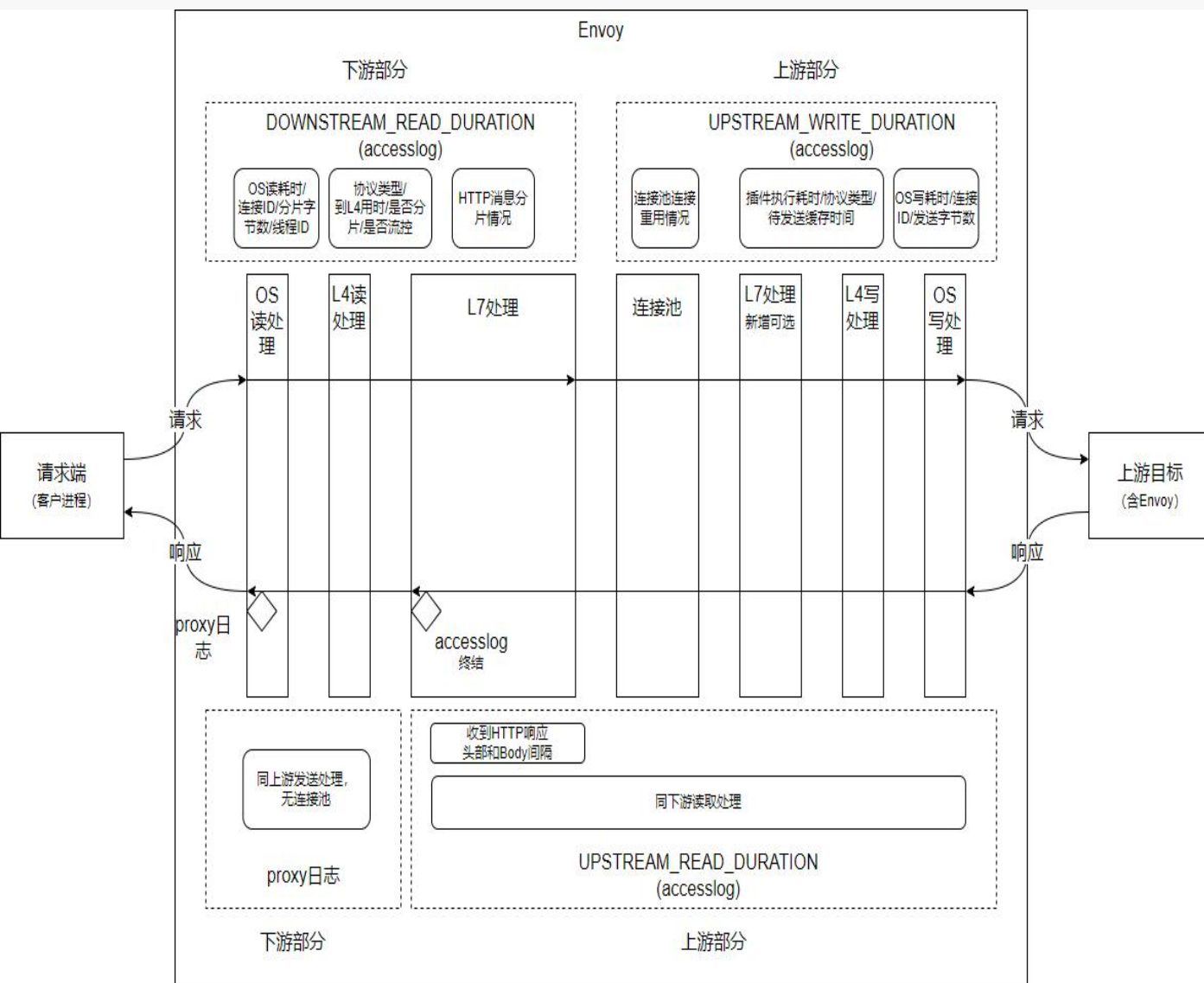
- 线上问题、测试环境无法稳定重现、即使重现在Envoy现有日志也没法直接判断。

破题思路

1. 时延类问题需要在请求整个梳理路径中，对L4数据收发、插件执行、连接池观察等多个方面进行综合分析。
2. 可以结合tcpdump抓包查看客户端到Envoy、Envoy之间、Envoy到目标服务之间包时间戳的差异（需要NTP校准各个部分的系统时间）。
3. 需要将更详细的观测状态记录到AccessLog中，作为日常观测手段，可以快速对不同时延类问题进行判断。

解决问题思路：简要但不简单；经验->能力

核心实践：增强Envoy运维日志输出信息



打通 Envoy请求处理链路中L4~L7自身运行状态观察能力

1. 增强 Envoy 的请求级别信息，提供更详细的信息，包括：

- 调用操作系统读写操作的耗时
- 是否使用安全连接
- 处理连接的线程 ID及连接ID
- 是否出现 HTTP 消息**分片**
- 是否出现由于连接内缓存限制导致的**流控**
- **插件**框架执行时间
- 上游连接池在 HTTP 协议中是否被**重用**
- 处理 HTTP 响应时收到的 HTTP 头部和数据部分是否存在较大时间间隔等信息

2. 日志增强增强信息可以帮助快速定位时延问题，例如系统调用阻塞、消息分片、Envoy 自身插件处理时间过长、到达缓存水位暂停接收等。

3. 日志增强功能不会影响现有的 Envoy 内的请求级别服务维度日志，而是作为日常运维手段的补充。


4. 可以提供更全面的请求级别信息，帮助运维人员更好地分析和调优系统性能，快速定位潜在的性能瓶颈和延迟问题。

成果展示1

判断HTTP大小，以及Envoy L4接收及发送的数据量和Envoy处理中自身增加的HTTP头部大小：

只有HTTP Header部分的下游请求日志：

```
[2023-11-28T02:21:36.106Z] "GET / HTTP/1.1" 200 - via_upstream - "-" 0 12 1089 1033 "-" "fortio.org/fortio-1.61.0" "cfff585a-52c9-97a6-8ccd-1c3269a15d64" "http-server-service.default:9000" "10.244.1.19:9000" outbound|9000||http-server-service.default.svc.cluster.local 10.244.2.16:42688 10.96.45.119:9000 10.244.2.16:49218 - default : 1089, 0, 136, 1089, 0, 1033 : (42) "[C56:R:<21:36.106>0,0,96]H" "[C58~C58:S:0,<21:36.242>0,1271]d" "[C58:S:<21:37.195>0,1124]H,D,E" 1089 0
```



1. 上面例子DOWNSTREAM_READ_DURATION 部分[C56:R:<21:36.106>0,0,96]H中：

- <21:36.106>0 表示OS接收开始的时间戳，以及接收用时ms，如果耗时较长需要考虑调整内核参数提升网络接收性能。
- 96 表示本次接收到的数据字节数
- H 表示本地接收到的数据经过HTTP解码器作为HTTP 头部进行解码。
- Envoy内HTTP添加部分：1271-96 = 1175字节


2. 如果观察到日志中只有头部而没有数据部分，并且整个接收字节数较大，那么可以推断存在较大的 HTTP 头部。在这种情况下，可能会触发一些 HTTP 服务器处理框架中的 HTTP 头部长度的限制。管理员可以将这个问题反馈给开发，建议他们将某些头部参数移动到数据部分。通过将一部分头部参数放入数据部分，可以减少整体头部的长度，从而避免触发服务器处理框架的限制。

成果展示2

判断Envoy工作线程数量及各个线程处理新连接情况是否均衡：


Envoy启动参数：concurrency参数启动4个工作线程

```
info    Envoy command: [-c etc/istio/proxy/envoy-rev.json --drain-time-s 45 --drain-strategy immediate --parent-shutdown-time-s 60 -  
t %Y-%m-%dT%T.%fZ    %l    envoy %n    %v -l warning --component-log-level misc:error --concurrency 4]
```



单行日志：

```
[2023-11-28T02:21:36.106Z] "GET / HTTP/1.1" 200 - via_upstream - "-" 0 12 1089 1033 "-" "fortio.org/fortio-1.61.0" "cfff585a-52c9-97a6-8ccd-1c3269a15d64" "http-server-service.default:9000" "10.244.1.19:9000" outbound|9000||http-s  
erver-service.default.svc.cluster.local 10.244.2.16:42688 10.96.45.119:9000 10.244.2.16:49218 - default : 1089, 0, 136, 1089, 0, 1033 : (42) "[C56:R:<21:36.106>0,0,96]H" "[C58~C58:S:0,<21:36.242>0,1271]d" "[C58:S:<21:37.195>0,0,1  
124]H,D,E" 1089 0
```



使用命令：awk ‘{print \$29}’ 4.log | sort | uniq -c 统计每个线程处理请求的数量了解各个线程处理连接是否均衡：

```
10929 (40)  
13327 (41)  
11334 (42)  
10967 (45)
```

1. 在 Istio 1.8 版本之前，存在在多线程环境下，outbound 方向的 Envoy 可能出现每个线程处理连接数差异较大的情况的问题，这可能导致端到端处理的平均时延不均衡。
2. 运维人员可判断是否需要启用连接均衡选项来提升吞吐。从而在多线程环境下更均衡地分配连接，提高系统性能并改善响应时间。

成果展示3

判断是否单个消息太大导致TCP分片，或HTTP响应头和数据部分之间产生延迟：

a. 由于HTTP请求及响应消息较大，导致出现同一个HTTP消息出现分片：

```
[2023-11-28T02:56:20.815Z] "POST /large?length=100000 HTTP/1.1" 200 - via_upstream - "-" 262144 100001 436 436 "-" "fortio.org/fortio-1.61.0" "5c679327-928a-97c3-bd50-2fb9d4cd121a" "http-server-service.default:9000" "10.244.1.19:9000" outbound|9000||http-server-service.default.svc.cluster.local 10.244.2.18:35432 10.96.45.119:9000 10.244.2.18:51004 - default : 436, 0, 0, 436, 0, 436 : (43) "[C752:R:<56:20.815>0,0,14480]H,D-[C752:R:<56:20.815>0,0,247836]D,E" "[C0~C234:S:0,<56:20.815>0,263491]d" "[C234:S:<56:21.251>0,0,16384]H,D-[C234:S:<56:21.251>0,0,84758]D,E" 436 0

[2023-11-28T02:56:21.004Z] "POST /large?length=100000 HTTP/1.1" 200 - via_upstream - "-" 262144 100001 247 247 "-" "fortio.org/fortio-1.61.0" "bb543594-d8a7-959f-9e77-bc0ac041dd91" "http-server-service.default:9000" "10.244.1.19:9000" outbound|9000||http-server-service.default.svc.cluster.local 10.244.2.18:35370 10.96.45.119:9000 10.244.2.18:51002 - default : 247, 0, 0, 247, 0, 247 : (44) "[C769:R:<56:21.004>0,0,14480]H,D-[C769:R:<56:21.004>0,0,247836]D,E" "[C0~C229:S:0,<56:21.004>0,263491]d" "[C229:S:<56:21.252>0,0,32768]H,D-[C229:S:<56:21.252>0,0,68374]D,E" 247 0
```

b. HTTP响应头部与数据部分存在延时19ms：

```
[2023-11-28T03:31:48.379Z] "POST /split?delay=19 HTTP/1.1" 200 - via_upstream - "-" 100000 1001 22 2 "-" "curl/8.4.0" "8d1ba2c4-cfad-91cb-8c45-9040d3836aa1" "http-server-service.default:9000" "10.244.1.19:9000" outbound|9000||http-server-service.default.svc.cluster.local 10.244.2.13:46584 10.96.45.119:9000 10.244.2.13:43210 - default : 22, 0, 1, 2, 19, 2 : (39) "[C3950:R:<31:48.379>0,0,100183]H,D,E" "[C3951~C3951:S:0,<31:48.380>0,101352]d" "[C3951:S:<31:48.382>0,0,1077]H-[C3951:S:<31:48.401>0,0,1013]D,E" 2 19
```

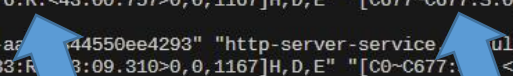
上面例子中：

1. Envoy 的处理过程中，HTTP 头部和数据部分为独立回调，并且每个回调在完成后会单独发送到 L4 连接的发送缓冲区中，当多个客户端频繁发送请求时，可能会导致 HTTP 消息的头部和数据部分被分别发送，增加某HTTP 消息时延。
2. 通过分析分片日志，可以观察到同一个 HTTP 消息的每个分片的接收时间，以及在服务器端发送响应时是否分开发送HTTP 头部和数据部分，增加了客户端的端到端处理时延。
3. 对于这种情况，可以考虑以下优化措施：
 - 减小请求端单个HTTP请求的发送的数据量大小。
 - 优化服务器端的处理逻辑，以减少处理时间，确保 HTTP 头部和数据部分能够尽快被一起发送给客户端。

成果展示4

判断Envoy之间是否启动了TLS，以及是否执行了连接复用

```
[2023-11-28T01:43:00.757Z] "POST / HTTP/1.1" 200 - via_upstream - "-" 1000 12 2 1 "-" "curl/8.4.0" "dc79cc4c-9324-9f81-bcc5-26bdf104dda9" "http-server-service.default:9000" "10.244.1.19:9000" outbound|9000||http-server-service.default.svc.cluster.local 10.244.2.13:43934 10.96.45.119:9000 10.244.2.13:46598 - default : 2, 0, 1, 2, 0, 1 : (39) "[C676:R:<43:00.757>0,0,1167]H,D,E" "[C677~C677:S:0,<43:00.759>0,2336]d" "[C677:S:<43:00.759>0,0,1122]H,D,E" 2 0
[2023-11-28T01:43:09.310Z] "POST / HTTP/1.1" 200 - via_upstream - "-" 1000 12 1 1 "-" "curl/8.4.0" "26586eab-1323-9ba1-a44550ee4293" "http-server-service.default:9000" "10.244.1.19:9000" outbound|9000||http-server-service.default.svc.cluster.local 10.244.2.13:43934 10.96.45.119:9000 10.244.2.13:43366 - default : 1, 0, 0, 1, 0, 1 : (39) "[C683:R:<43:09.310>0,0,1167]H,D,E" "[C0~C677:S:0,<43:09.311>0,2336]d" "[C677:S:<43:09.312>0,0,1122]H,D,E" 1 0
```



1. 在线上配置过程中，如网格默认没有启用 mTLS，导致新增的服务没有配置 mTLS，从而导致明文传输，增加了安全风险。可以通过监控 Envoy 日志来发现此类情况。
2. 在下面的日志中（例如2023-11-28T01:43:00.757Z），可以找到以下信息：
 - DOWNSTREAM_READ_DURATION中：[C676:R:<43:00.757>0,0,1167]H,D,E，表示客户端发起的请求是明文内容，其中的 R 表示明文请求。
 - UPSTREAM_WRITE_DURATION中：[C677~C677:S:0,<43:00.759>0,2336]d中，C677~C677表示上游连接池在收到客户请求后新建了一个 C677 连接，并且实际使用的也是 C677 连接。S 表示该连接使用了 TLS。
 - 另外在 Istio 1.8 版本之前，存在一个问题，即创建的上游连接并不一定是实际请求所使用的连接。这可能导致消息接收端的错乱。您也可以通过上面提到的日志信息中连接ID来判断。
3. 在接下来的2023-11-28T01:43:09.310Z日志中：
 - UPSTREAM_WRITE_DURATION中：[C0~C677:S:0,<43:09.311>0,2336]d中，表示此时没有创建新的上游连接，而是重用了已有的 C677 连接。可以判断在某段时间内有多少活跃的上游连接被重用。根据重用情况，可以决定是否需要调整连接池连接参数，扩大或缩减连接池内的活动连接容量。这样可以优化连接池的性能和资源利用情况。

成果展示5

判断是否由于连接缓冲区水位大小导致接收关闭：

a.上游接收响应触发连接流控日志：

```
[2023-11-28T06:18:00.592Z] "POST /large?length=10000 HTTP/1.1" 200 - via_upstream - "-" 10000 10001 82 81 "-" "fortio.org/fortio-1.61.0" "94e650bf-bd50-9738-805b-8bd34001c363" "http-server-service.default:9000" "10.244.1.19:9000"
outbound|9000||http-server-service.default.svc.cluster.local 10.244.2.21:55632 10.96.45.119:9000 10.244.2.21:34836 - default : 82, 0, 41, 82, 0, 81 : (42) "[C43:R:<18:00.592>0,0,10170]H!1,D!1,E!1" "[C44~C44:S:0,<18:00.633>0,1134
5]d" "[C44:S:<18:00.674>0,0,11137]H,D,E!1" 82 0
```

b.上游接收响应水位恢复后日志：

```
[2023-11-28T06:18:00.633Z] "POST /large?length=10000 HTTP/1.1" 200 - via_upstream - "-" 10000 10001 109 72 "-" "fortio.org/fortio-1.61.0" "14ffed64-a22e-9139-8316-0b906f471d7e" "http-server-service.default:9000" "10.244.1.19:9000"
outbound|9000||http-server-service.default.svc.cluster.local 10.244.2.21:55632 10.96.45.119:9000 10.244.2.21:34880 - default : 109, 37, 47, 109, 0, 72 : (42) "[C99:R:<18:00.633>0,0,10170]H!1,D!1,E!1" "[C114~C44:S:0,<18:00.680>0,
11345]d" "[C44:S:<18:00.743>0,0,11133]H+5<18:00.679>,D,E!1" 109 0
```

1. Envoy 每个连接都可以配置缓存的水位大小。默认情况下，低水位线为512K，高水位线1M。当前连接内的请求未完成处理时，如果新的请求数据量超过了缓存水位，将会触发水位关闭。一旦待处理的数据发送完毕，水位将恢复。
2. 下游连接和上游连接是独立的连接，它们分别具有各自的缓存配置。这意味着在处理下游请求时使用的缓存配置可能与处理上游请求时使用的缓存配置不同
3. 从上面的例子中可以看出：
 - （42）线程上游连接日志UPSTREAM_READ_DURATION显示在<18:00.647>0,0,11137]H,D,E!1时接收服务端响应时，由于上游发送处理较慢无法发送到后端服务时，触发水位关闭，连续关闭次数为1。
 - 在（42）线程的下一个请求处理中，上游连接日志UPSTREAM_READ_DURATION显示C[114~C44:S:0,<18:00.680>0,11345]d”
“[C44:S:<18:00.743>0,0,11133]H+5<18:00.679>,D,E!1时，在经过5ms（对应前一条日志关闭时间<18:00.647>）后水位在<18:00.679>重新开放，因此上游发送可以在<18:00.680>时继续发送数据。此时可以判断出是由于Envoy上游连接水位被限制导致服务端接收数据变慢。管理员可以配置下游监听器或上游Cluster的perConnectionBufferLimitBytes参数控制连接水位。
 - 根因推测为下游写response较慢导致上游的读出现了水位限制。

成果展示6


判断Envoy插件执行时间：

用例：增加根据头部x-delay参数进行延时的lua插件：

```
1 apiVersion: networking.istio.io/v1alpha3
2 kind: EnvoyFilter
3 metadata:
4   name: http-server-lua-delay-filter
5 spec:
6   #workloadSelector:
7   # labels:
8   #   app: http-server-outer-service
9   configPatches:
10    - applyTo: HTTP_FILTER
11      match:
12        context: SIDECAR_OUTBOUND
13        listener:
14          filterChain:
15            filter:
16              name: "envoy.filters.network.http_connection_manager"
17              subFilter:
18                name: "envoy.filters.http.router"
19      patch:
20        operation: INSERT_BEFORE
21        value:
22          name: envoy.lua
23          typed_config:
24            "@type": "type.googleapis.com/envoy.extensions.filters.http.lua.v3.Lua"
25            inlineCode: |
26              function sleep(n)
27                os.execute("sleep " .. tonumber(n))
28              end
29
30              function envoy_on_request(request_handle)
31                local delay = request_handle:headers():get("x-delay")
32                request_handle:logWarn("get request")
33                if delay then
34                  request_handle:logWarn("Delaying request for " .. delay .. " seconds")
35                  request_handle:logWarn("Adding delay header to response")
36                  local response_headers = request_handle:headers():add("x-delayed-response", delay)
37                  local delay_seconds = tonumber(delay)
38                  if delay_seconds then
39                    request_handle:logWarn("Sleeping for " .. delay .. " seconds")
40                    sleep(delay_seconds)
41                  end
42                end
43              end
```

访问日志：

```
cf3-a04d-24ad6b053232" "http-server-service.default:9000" "10.244.1.19:9000" outbound|9000||http-server-service.def
: (40) "[C10001:R:<53:08.802>0,0,106]H" "[C10004~C10004:S:0,<53:17.805>0,1299]d" "[C10004:S:<53:17.806>0,0,1122]H,
```



使用命令：

curl -v -Hx-delay:9 <http://http-server-service.default:9000> 访问服务

根据以上日志：

1. 下游接收DOWNSTREAM_READ_DURATION时间点<53:08.802>到上游发送UPSTREAM_WRITE_DURATION时间点<53:17.806>之间有9s的时间差，排除Envoy基本处理流程耗时外，相差较大的时间可以猜测为耗时的HTTP插件引入。
2. 接下来帮助管理员排查是否由于lua或耗时的用户自定义插件引入时延问题。

非时延类问题

有万分之一的概率，在WAF上看到502报错：

分析方法：

1. 在WAF及ingressgateway侧分别抓包，发现都有FIN报文，Envoy主动关闭连接。

344204	2021-08-26 02:19:18.879070	172.16.0.177	172.16.0.187	TCP	16701 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
344205	2021-08-26 02:19:18.879193	172.16.0.187	172.16.6.177	TCP	443 → 16701 [SYN, ACK] Seq=0 Ack=1 Win=28040 Len=0 MSS=1402 SACK_PERM=1 WS=512
344206	2021-08-26 02:19:18.883770	172.16.0.177	172.16.0.187	TCP	16701 → 443 [ACK] Seq=1 Ack=1 Win=29312 Len=0
344210	2021-08-26 02:19:18.894135	172.16.0.187	172.16.6.177	TCP	443 → 16701 [FIN, ACK] Seq=1 Ack=1 Win=28160 Len=0
344211	2021-08-26 02:19:18.894589	172.16.0.177	172.16.0.187	TLSv1	Client Hello
344212	2021-08-26 02:19:18.894649	172.16.0.187	172.16.6.177	TCP	443 → 16701 [RST] Seq=1 Win=0 Len=0
344213	2021-08-26 02:19:18.899159	172.16.0.177	172.16.0.187	TCP	16701 → 443 [ACK] Seq=228 Ack=2 Win=29312 Len=0
344214	2021-08-26 02:19:18.899217	172.16.0.187	172.16.6.177	TCP	443 → 16701 [RST] Seq=2 Win=0 Len=0
344222	2021-08-26 02:19:18.903364	172.16.0.177	172.16.0.187	TCP	16701 → 443 [FIN, ACK] Seq=228 Ack=2 Win=29312 Len=0
344223	2021-08-26 02:19:18.903433	172.16.0.187	172.16.6.177	TCP	443 → 16701 [RST] Seq=2 Win=0 Len=0

2. 继续抓包，分析正常报文与异常报文区别，502异常报文的ACK报文与客户端Client Hello报文时间间隔超过10ms，而正常ACK报文与Client Hello报文间隔小于10ms。

173	2021-08-26 02:11:31.206269	172.16.0.177	172.16.0.187	TCP	28569 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
174	2021-08-26 02:11:31.206379	172.16.0.187	172.16.6.177	TCP	443 → 28569 [SYN, ACK] Seq=0 Ack=1 Win=28040 Len=0 MSS=1402 SACK_PERM=1 WS=512
180	2021-08-26 02:11:31.210386	172.16.0.177	172.16.0.187	TCP	28569 → 443 [ACK] Seq=1 Ack=1 Win=29312 Len=0
181	2021-08-26 02:11:31.210480	172.16.0.177	172.16.0.187	TLSv1.2	Client Hello
182	2021-08-26 02:11:31.210609	172.16.0.187	172.16.6.177	TCP	443 → 28569 [ACK] Seq=1 Ack=228 Win=29184 Len=0
183	2021-08-26 02:11:31.211518	172.16.0.187	172.16.6.177	TLSv1.2	Server Hello, Certificate, Server Key Exchange, Server Hello Done
185	2021-08-26 02:11:31.215531	172.16.0.187	172.16.0.187	TCP	28569 → 443 [ACK] Seq=228 Ack=2805 Win=34816 Len=0
186	2021-08-26 02:11:31.215555	172.16.0.177	172.16.0.187	TCP	28569 → 443 [ACK] Seq=228 Ack=3253 Win=37632 Len=0
190	2021-08-26 02:11:31.216526	172.16.0.177	172.16.0.187	TLSv1.2	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
191	2021-08-26 02:11:31.216741	172.16.0.187	172.16.6.177	TLSv1.2	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
199	2021-08-26 02:11:31.220864	172.16.0.177	172.16.0.187	TCP	28569 → 443 [ACK] Seq=313 Ack=3503 Win=40448 Len=1394 [TCP segment of a reassembled PDU]
200	2021-08-26 02:11:31.220881	172.16.0.177	172.16.0.187	TLSv1.2	Application Data
201	2021-08-26 02:11:31.220973	172.16.0.187	172.16.6.177	TCP	443 → 28569 [ACK] Seq=3503 Ack=1964 Win=34816 Len=0
209	2021-08-26 02:11:31.222710	172.16.0.187	172.16.6.177	TLSv1.2	Application Data



结论：

1. Istio 1.3老版本默认开启协议嗅探，可自动检测客户端是否开启TLS协议，并会根据客户端发送的第一个Client Hello报文判断协议种类。默认参数protocolDetectionTimeout: 10ms。
2. 如果在嗅探配置时间内未收到客户端报文，Envoy会主动断开连接。Istio在1.3之后版本默认关闭了协议嗅探功能。
3. kubectl edit cm istio -nistio-system可将protocolDetectionTimeout修改为”0s”关闭协议嗅探。

非时延类问题

上传较大图片，网关偶现413报错：

分析方法：

1. 使用tcpdump在ingressgateway抓包，结合报错信息“request data too large watermark exceeded”，初步判断和buffer limit相关。

```
2023-08-09T09:43:21.703242Z    91 warning    envoy lua      script log: ...lua-filter-header already has instance-id: smartovms
2023-08-09T09:43:21.706488Z    91 debug     envoy http     [C1199037646][82140685243370012332] request data too large watermark exceeded
2023-08-09T09:43:21.706505Z    91 debug     envoy http     [C1199037646][82140685243370012332] Sending local reply with details request payload too large
2023-08-09T09:43:21.706625Z    91 debug     envoy http     [C1199037646][82140685243370012332] encoding headers via codec (end_stream=false);
```

2. 分析Envoy代码，发现相关参数为maxRequestBytes和per_connection_buffer_limit_bytes。且经过分析per_connection_buffer_limit_bytes不会对请求造成中断，只会影响请求处理速度。

```
2023-08-12T07:42:47.075146Z    29 debug     envoy http     [C1241728][S8366719042739883616] Read-enabling downstream stream due to filter callbacks.
2023-08-12T07:42:47.075152Z    29 debug     envoy connection [C1241728] onAboveReadBufferHighWatermark
2023-08-12T07:42:47.075156Z    29 debug     envoy http     [C1241728][S8366719042739883616] Read-disabling downstream stream due to filter callbacks.
2023-08-12T07:42:47.075158Z    29 debug     envoy http     [C1241728][S8366719042739883616] Read-enabling downstream stream due to filter callbacks.
2023-08-12T07:42:47.075159Z    29 debug     envoy connection [C1239251] onAboveWriteBufferHighWatermark
2023-08-12T07:42:47.075160Z    29 debug     envoy http     [C1241728][S8366719042739883616] Read-disabling downstream stream due to filter callbacks.
2023-08-12T07:42:47.075162Z    29 debug     envoy connection [C1241728] onBelowReadBufferLowWatermark
2023-08-12T07:42:47.075172Z    29 debug     envoy connection [C1239251] onBelowWriteBufferLowWatermark
2023-08-12T07:42:47.075174Z    29 debug     envoy http     [C1241728][S8366719042739883616] Read-enabling downstream stream due to filter callbacks.
2023-08-12T07:42:47.075184Z    29 debug     envoy connection [C1241728] onAboveReadBufferHighWatermark
2023-08-12T07:42:47.075189Z    29 debug     envoy http     [C1241728][S8366719042739883616] Read-disabling downstream stream due to filter callbacks.
2023-08-12T07:42:47.075190Z    29 debug     envoy http     [C1241728][S8366719042739883616] Read-enabling downstream stream due to filter callbacks.
2023-08-12T07:42:47.075191Z    29 debug     envoy connection [C1239251] onAboveWriteBufferHighWatermark
2023-08-12T07:42:47.075193Z    29 debug     envoy http     [C1241728][S8366719042739883616] Read-disabling downstream stream due to filter callbacks.
2023-08-12T07:42:47.075194Z    29 debug     envoy connection [C1241728] onBelowReadBufferLowWatermark
2023-08-12T07:42:47.075215Z    29 debug     envoy connection [C1239251] onBelowWriteBufferLowWatermark
```

```
root@ecs-guobaoqing-0054:~/Code/github/large-payload# curl -i -s -w "@curl-format.txt" -X POST -d "@10m-payload.txt" http://100.93.9.212:8000/post
HTTP/1.1 100 Continue

HTTP/1.1 413 Payload Too Large
content-length: 17
content-type: text/plain
date: Sat, 12 Aug 2023 10:54:02 GMT
server: istio-envoy
connection: close

Payload Too Large      time_total: 0.008459s
response_code: 413
payload_size: 2752512
```

结论：

1. 对maxRequestBytes参数配置1MB阈值后，测试传输超过1MB的图片时，报错基本稳定重现。
2. 分析代码发现，处理请求decode阶段，会对buffer limit做检查，如果请求数据量高于maxRequestBytes，则报错413。
3. 解决方法：通过envoyfilter为listener下发maxRequestBytes参数配置。如下：

```
metadata:
  name: limit-request-size
  namespace: istio-system
spec:
  workloadSelector:
    labels:
      istio: ingressgateway
  configPatches:
    - applyTo: HTTP_FILTER
      match:
        context: GATEWAY
        listener:
          portNumber: 1026
          filterChain:
            filter:
              name: envoy.http_connection_manager
      patch:
        operation: INSERT_BEFORE
        value:
          name: envoy.buffer
          typed_config:
            '@type': type.googleapis.com/udpa.type.v1.TypedStruct
            value:
              maxRequestBytes: 1048576 # 1 MB
```

非时延类问题

修改Listener配置，导致长连接断开：

现象：Istio 1.8修改EnvoyFilter内容，45s后websocket连接会断开，虽然客户有重试，但大量连接请求对后端造成压力。

```
---yaml
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: filter-ncms-gray
  namespace: istio-system
spec:
  configPatches:
    # The first patch adds the lua filter to the listener/http connection manager
    - applyTo: HTTP_FILTER
      match:
        context: GATEWAY
        listener:
          filterChain:
            filter:
              name: "envoy.filters.network.http_connection_manager"
              subFilter:
                name: "envoy.filters.http.router"
      patch:
        operation: INSERT_BEFORE
        value: # lua filter specification
          name: envoy.lua
          typed_config:
            "@type": "type.googleapis.com/envoy.extensions.filters.http.lua.v3.Lua"
```

分析方法：

1. 进行重现发现此问题可以稳定复现。
2. 对比使用EnvoyFilter前后的Envoy dump文件配置内容，并主要关注受影响的Listener的配置变化。

结论：

1. 分析Envoy代码得知，Envoy采用Listener->filterChain配置结构，同时接收的客户端连接将同时保存与此filterChain关系。
2. 当apply新EnvoyFilter时，LDS将调用addOrUpdateListener添加新的Listener，并根据当前Listener是否支持inPlace判断是否支持原地替换。
3. 如果不支持，则调用startDrainSequence启动老Listener下线流程，并在到达Drain时间后，调用removeFilterChains关闭此Listener->filterChain关联的所有活动连接。
4. 此断开连接的底层逻辑是：由于连接上配置的过滤器内容都是在接收连接时确定的，如果不断开，则Listener上已有连接的执行逻辑可能与新建立的连接不一致。

成果展示

1. 通过增强 Envoy 的运维功能，可以将线上时延类问题的**定位范围**缩小到原来的1/10。
2. 此运维增强日志扩展了Envoy AccessLog，可以**在线上业务**正常运行时实时记录和分析日志。

案例复盘与总结

1. 引入了服务网格Envoy作为透明代理，其透明拦截机制与未加入网格时对比在用户链路中产生一些**不确定**因素。
2. 时延类问题是比较**难以定位**的，如在问题发生后重新搭建环境进行重现，这样往往就失去了先机。
3. 日志增强功能不仅是对AccessLog本身的增强，关键状态以**最精简**的形式体现出来。

网格数据面演进：案例背景

- 1.在使用服务网格产品时，用户通常会遇到Sidecar资源消耗较大和网格**规模受限**的问题，这会影响客户购买的集群资源的利用率和单个网格应用的部署规模。
- 2.在进行NodeProxy节点级模式探索时，遇到如金融用户或游戏用户需求：，其中有些服务需要保证其**更高的服务优先级**。

问题与挑战

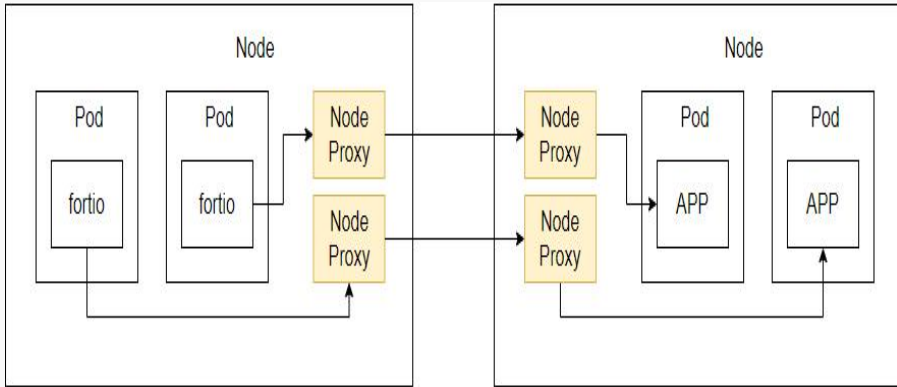
- Envoy代理逻辑复杂，随着网格规模的增加，网格内的配置占用的内存也会增加。并且随**配置同步通信量的增加**大规模网格下很难保持稳定运行。
- 容器自动调度机制，同一节点上可能有**不同优先级用户**。很难规划NodeProxy节点的内存和CPU参数。
- NodeProxy需要在每个节点上运行完整的代理，同样会**占用一定的节点内存**。希望尽可能地确保用户购买的**资源不受影响**，并且能够在流量增减时进行可控的L7弹性扩缩容。

破题思路

1. ASM产品：从Sidecar到节点模式NodeProxy，并向下一代**L4/L7分离**的kmesh卸载模式的演进路线。
2. 提供根据服务元数据进行L4优先级和**QoS能力**问题。
3. ASM采用了共享节点内控制面连接，**降低**每个节点对控制面的连接数，这些能力也适用于L4/L7分离场景。
- 4.在下一代L4/L7分离架构中，为了提高**L4处理性能**和实现更灵活的**L7弹性能力**，ASM采用了每节点L4内核卸载加L7弹性部署的方式。

核心实践：NodeProxy节点模式与Sidecar模式对比

NodeProxy模式



NodeProxy节点模式优化方案：

1. 解决**每Pod**内存占用。
2. 支持**元数据**进行灵活的L4负载均衡。
3. 节点内控制面连接共享。
4. NodeProxy之间**共享TCP**连接。
5. 支持节点内服务**优先访问策略**。
6. 数据面升级控制老版本**缓慢下线**

• 网格数据面随集群增加的内存占用情况比较：

网格规模	总Sidecar内存占用GB	总NodeProxy内存占用GB	内存下降比例
30pods / 200svcs	$30 * 0.04 = 1.2$	$4 * 0.08 = 0.32$	$1 - (0.32/1.2) = 73\%$
50 pods / 200svcs	$50 * 0.04 = 2$	$4 * 0.08 = 0.32$	84%
100pods / 200svcs	$100 * 0.04 = 4$	$4 * 0.08 = 0.32$	92%
100pods / 500svcs	$100 * 0.08 = 8$	$4 * 0.19 = 0.76$	91%
100pods / 1000svcs	$100 * 0.14 = 14$	$4 * 0.24 = 0.96$	93%
100pods / 1500svcs	$100 * 0.18 = 18$	$4 * 0.3 = 1.2$	93%

测试环境：

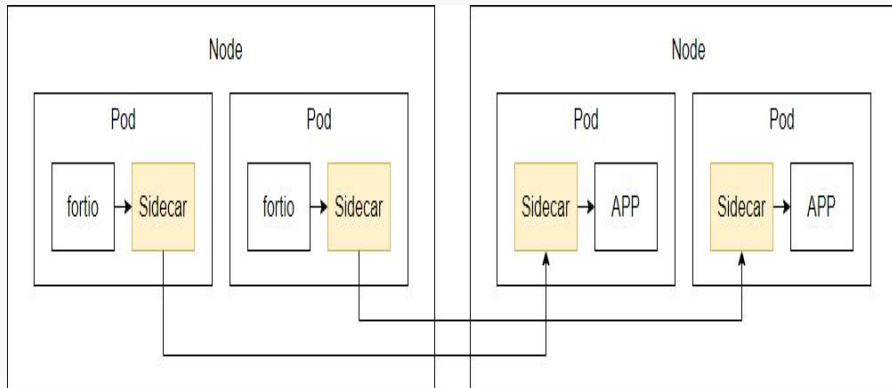
Istio 1.8.4-r3, k8s 1.19
16 核|32 GB|Sit3.4xlarge.2

分析：

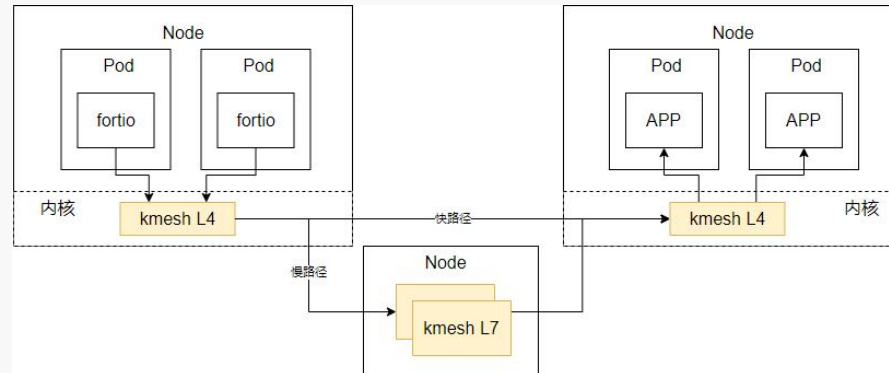
- Envoy内存占用的增长在Sidecar模式下会因为每Pod注入而被放大。但是在NodeProxy模式下则可以忽略。
- 随着网格内Pod和服务数量的上升，NodeProxy对内存占用下降可以达到**70%以上**。

核心实践：ASM网络数据面的演进路线

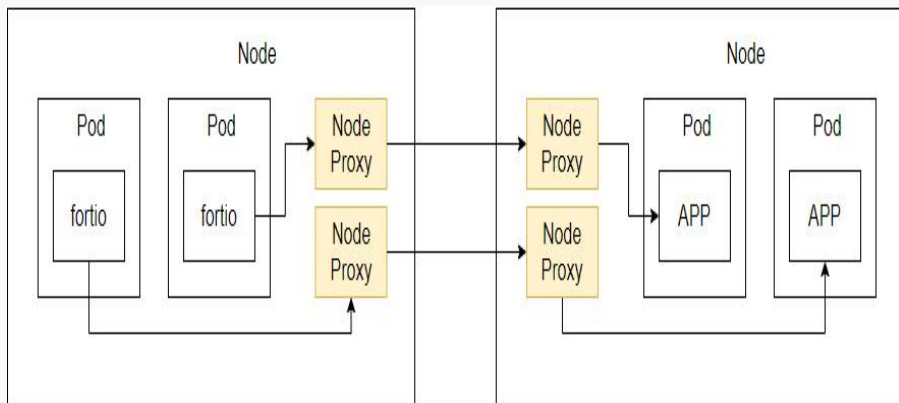
Sidecar模式



L4/L7分离模式



NodeProxy模式



NodeProxy问题:

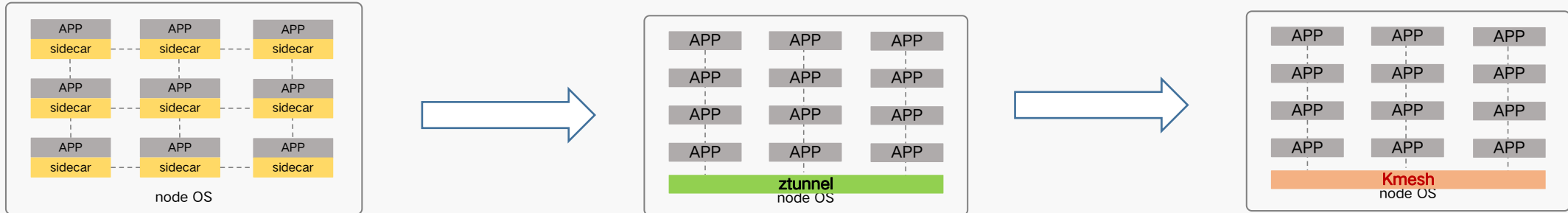
1. 固定L7代理，占用较大用户购买资源。
2. 每个请求需要拷贝用户完整数据。
3. 需要两侧都为NodeProxy部署，但需要控制面参与。

TOP1 👁️ 👁️



1. L4内核卸载**兼容**现有 iptables 配置规则。同时**减少**拷贝数据到用户态开销。
2. L4拦截区分**快慢**路径。
3. 支持除Istio外**第三方**安全认证中心。
4. 部分 L7 能力**可卸载**，从而提高性能。
5. 支持多种**异构基础设施**的互联互通场景，可集成多种容器CNI。
6. 支持L7本地及拉远两种模式，可提供**全托管**的数据面。
7. 提供了对已加入网格的服务进行**动态绕过**的能力。

核心实践：L4/L7分离与Sidecar模式性能及资源对比



- 有负载情况下，网格数据面L4代理性能及资源占用情况：

连接数	L4内存占用 MB	L4 CPU 占用	平均时延ms	P90时延ms	P99时延ms	QPS
64连接（未加入网格）	/	/	1.102	2.965	5.844	58038
64连接（ztunnel）	16.5	109.7%	1.741	3.196	5.422	36747
64连接（kmesh L4）	12	10%	1.047	2.949	5.71	61041

- 无负载情况下，网格数据面随集群增加的内存占用情况：

网络规模	单个Sidecar	Ztunnel	Waypoint	Kmesh L4
空载	56M	13M	55M	11M
200 pods / 100 svcs	74M	14M	78M	12M
1000 pods / 500 svcs	147M	18M	157M	16M
5000 pods / 2500 svcs	514M	38M	496M	33M
2.5w pods / 1.25w svcs	2.1G	112M	1.3~1.8G（不稳定）	96M
10w pods / 5w svcs	7.6G	150M	不稳定	130M

测试环境：

Istio 1.19, k8s 1.27

分析：kmesh相比未加入网格的直连情况性能持平（甚至从测试结果看略优），相比ambient的四层处理，CPU占用降低**90%**，平均时延仅后者的**60%**，QPS超过后者**60%**

分析：

1：虽然L4/L7分离后，每个Waypoint代理占用内存与单个Sidecar相当，但由于可以被共享，并且弹性扩容，整体集群内网格**固定内存消耗减少**。

2：节点内固定占用的网格L4部分内存随网络规模**增加较少**，转发稳定并与用户态L4相比降低**10%~15**内存占用。

核心实践：部分L7能力内核卸载



节点内L7治理加速：

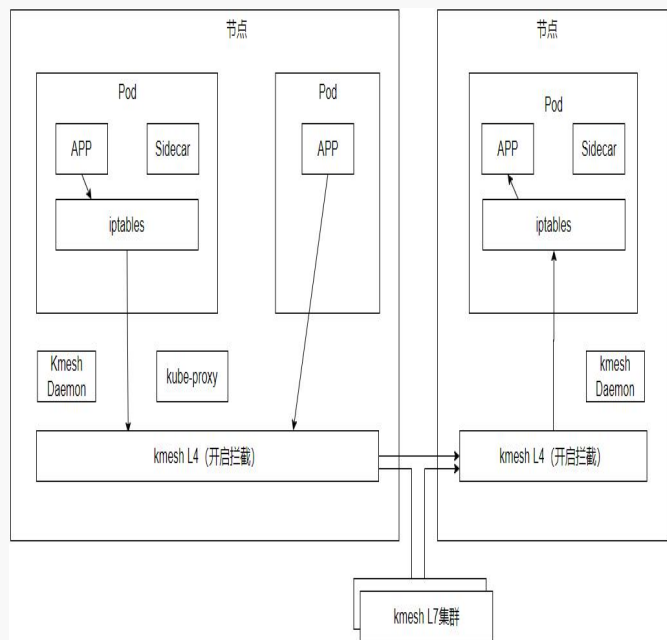
- 基于伪建链、延迟建链等技术，内核L4中实现部分L7的治理能力；
- 基于ebpf，在内核协议栈中构筑可编程的L7扩展能力；

核心实践：L4/L7分离数据面动态绕过能力

用户不敢切换
或升级网格！！

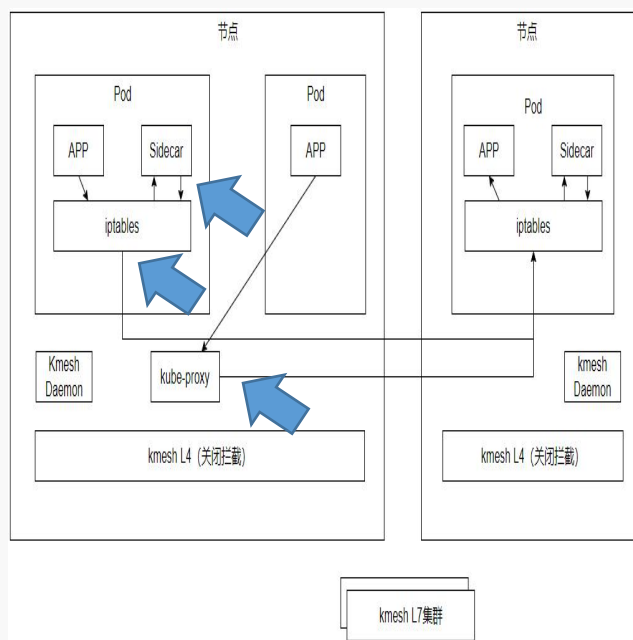
1. 服务可达、时延增加、内存占用增加等问题是否由网格引入定界难。
2. 一旦启用网格，修改代理模式重启，影响业务。
3. 动态修改iptables规则，有时间窗口，性能低，网络不稳定。

默认所有流量被kmesh L4拦截



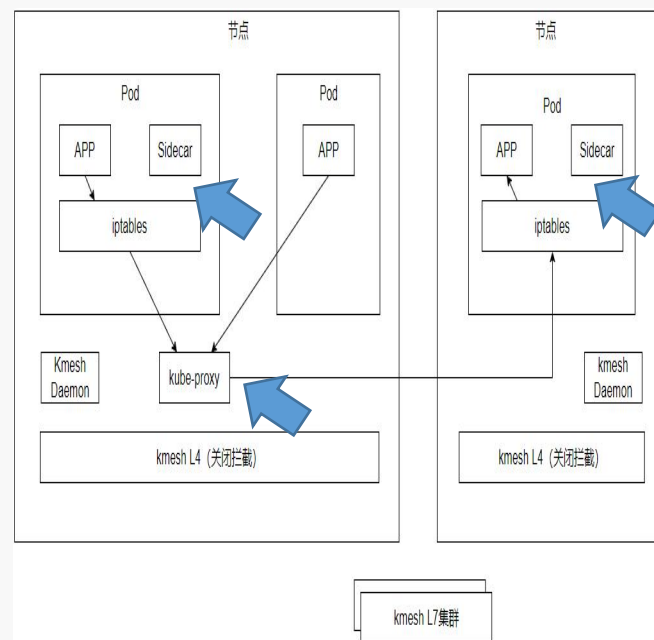
1. kmesh L4默认接管所有应用流量
2. 在Pod创建时注入少量iptables规则，运行态不增删。

模式一：跳过kmesh拦截



1. 只绕过kmesh 拦截，恢复已有流量路径。
2. 帮助判断是否由kmesh引入流量治理问题。

模式二：跳过所有网格拦截



1. 绕过所有网格 拦截，恢复Kubernetes流量。
2. 帮助判断是否由网格引入流量治理问题。
3. 逃生通道!!

案例启示（总）

运维增强方面：

1. 在问题定位过程中归纳总结经验、提炼可下沉的技术积累、构建产品的基础能力、持续迭代和改进。

问题总结->方法积累->能力下沉->持续迭代

数据面模型发展：

1. 需要详细了解数据面的运作原理，包括线程，连接，处理流程等。
2. 归纳数据面各个部分的职责， 以及相关性。
3. 了解操作系统的能力和最新的功能提升，在设计上做到弹性化，共享化，并缩短路径。
4. 拥抱开源，通过社区不断完善产品：

梳理流程->明晰职责->优化设计->拥抱开源

* 前面的优化项已经在kmesh开源之中。

下一步

1. 在运维方面，kmesh 将继续扩展现有的线上运维增强能力。通过结合 L4/L7 带来的内核观测能力和现有的 L7 观测能力，以便能够实现更快速、更精确的定位线上时延、访问可达性等问题。
2. 在 L4/L7 数据面分离方面，kmesh 将致力于实现统一注册中心的功能，更平滑数据面升级能力，同时完善产品使用体验，这将为传统微服务调用和 Serverless 提供更一致的治理体验。

请关注kmesh项目，在提升网格代理性能、运维能力提升等方面做出贡献，
与大家一起共同创造更好用的网格数据面



kmesh项目地址：

<https://github.com/kmesh-net/kmesh.git>

微信群：



谢谢！



微信官方公众号：壹佰案例
关注查看更多年度实践案例