基于eBPF的服务网格性能瓶颈定位与优化

主讲人：陈鹏飞
单位：中山大学

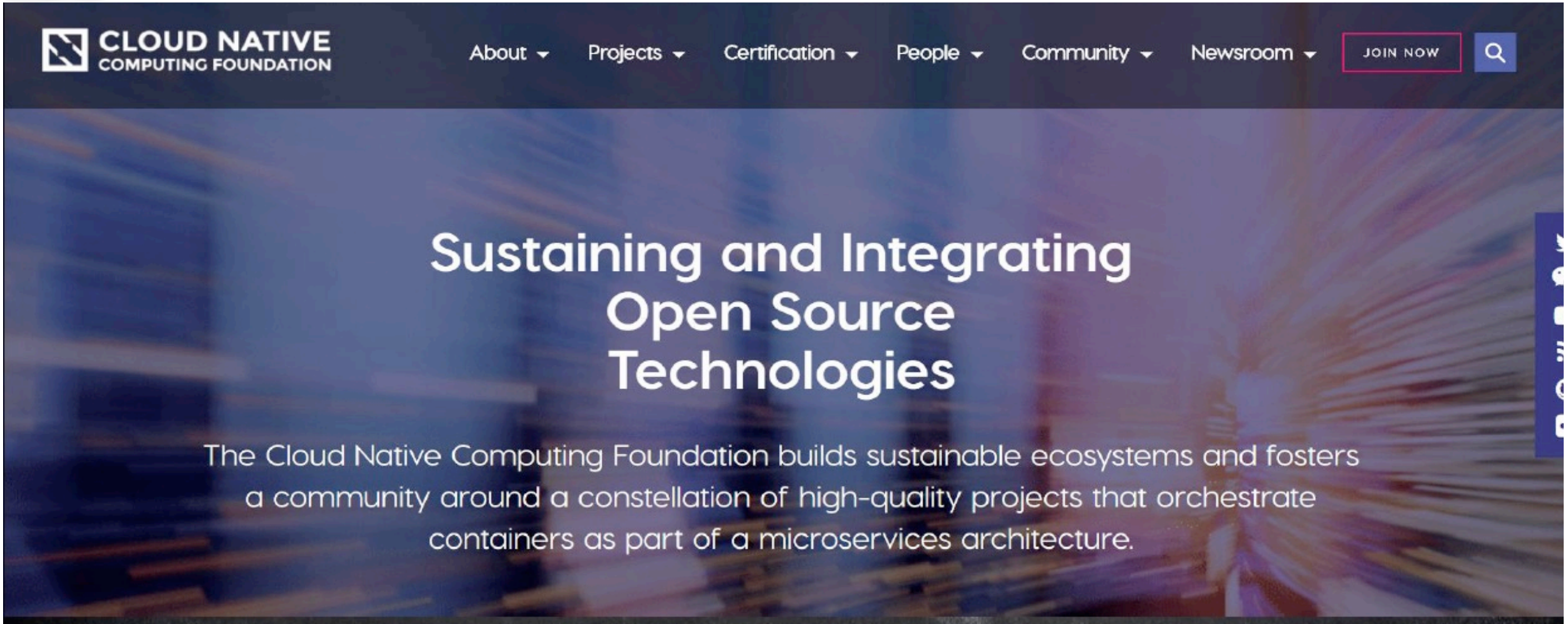
2022-11-12

目录

01

背景介绍

云原生系统

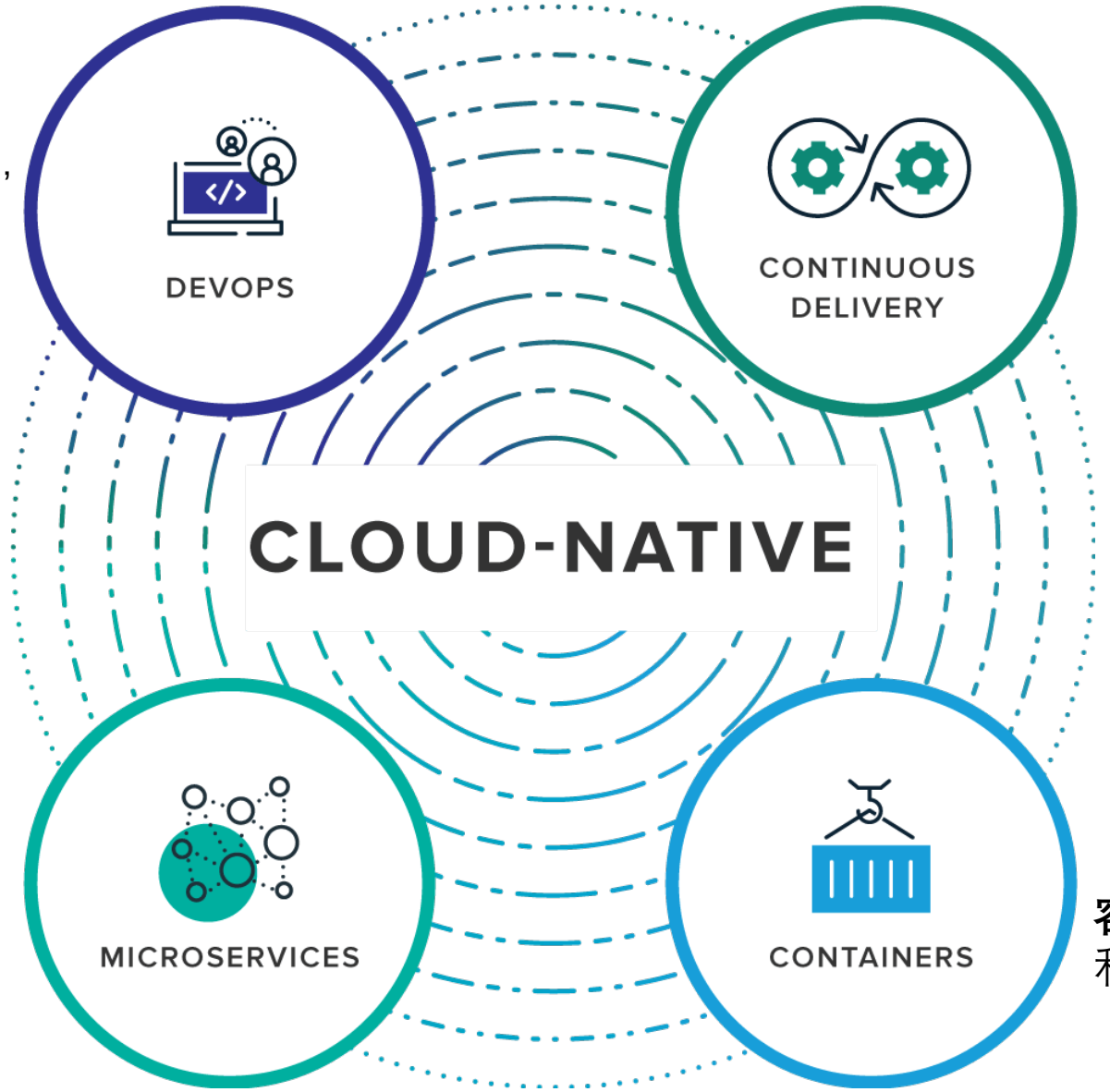


云原生技术帮助公司和机构在公有云、私有云和混合云等新型动态环境中，构建和运行弹性扩展的应用。云原生的代表技术包括容器、服务网格、微服务、不可变基础设施和声明API。

云原生系统

DevOps：新的软件开发模式，加速软件的开发速度；

连续交付：连续的开发和交付，减少业务Go-To-Market的时间



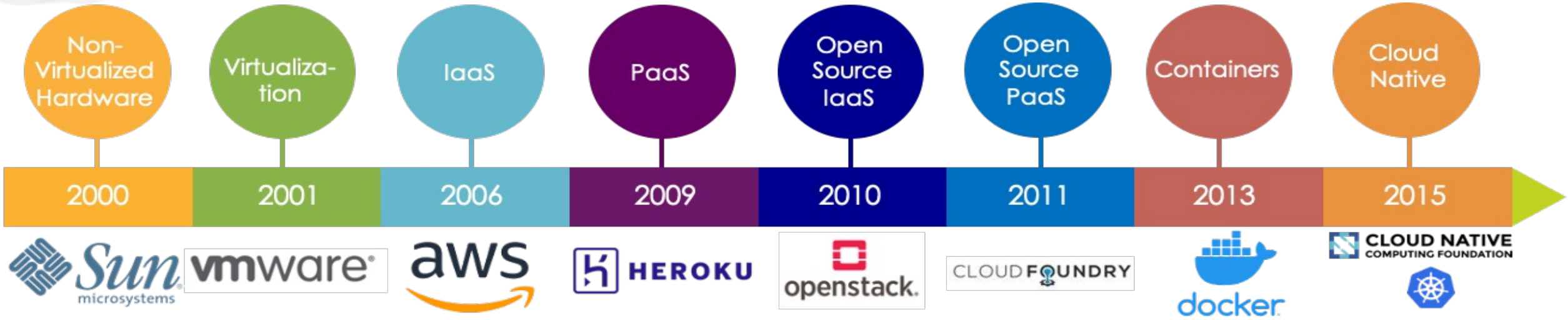
天下武功，唯快不破！

世间软件，唯快不赢！

微服务：小而精的软件产品，易于开发、交互和维护；

容器：基础使能技术，使开发和部署软件系统的速度加快

云原生系统



2008.01

cgroups 合并进入 Linux 内核主干

2014.06

Kubernetes 项目发布

2017.07

CNCF 成立2周年
170个成员
14个基金会项目

2004~2007

Google 大规模使用容器（cgroups）技术

2013.03

Docker 项目正式发布

2015.07

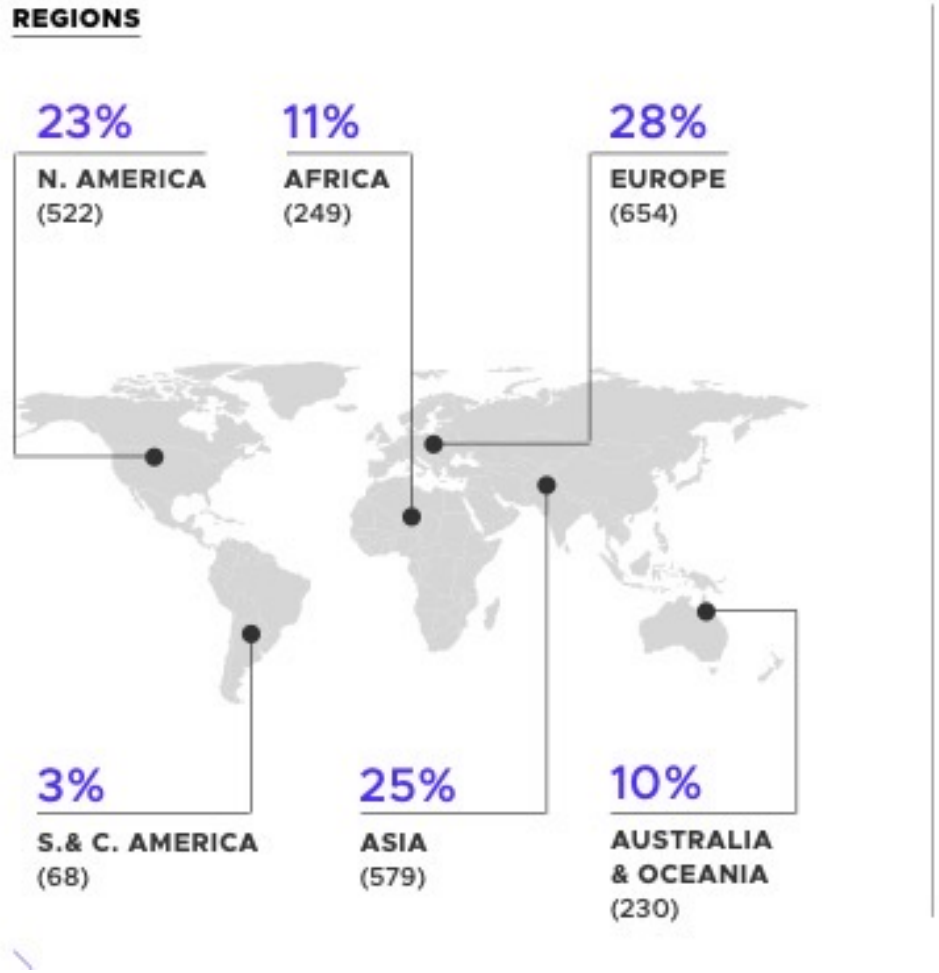
CNCF （云原生基金会）成立
22 个创始成员
Kubernetes 成为第一个 CNCF 项目

2018.07

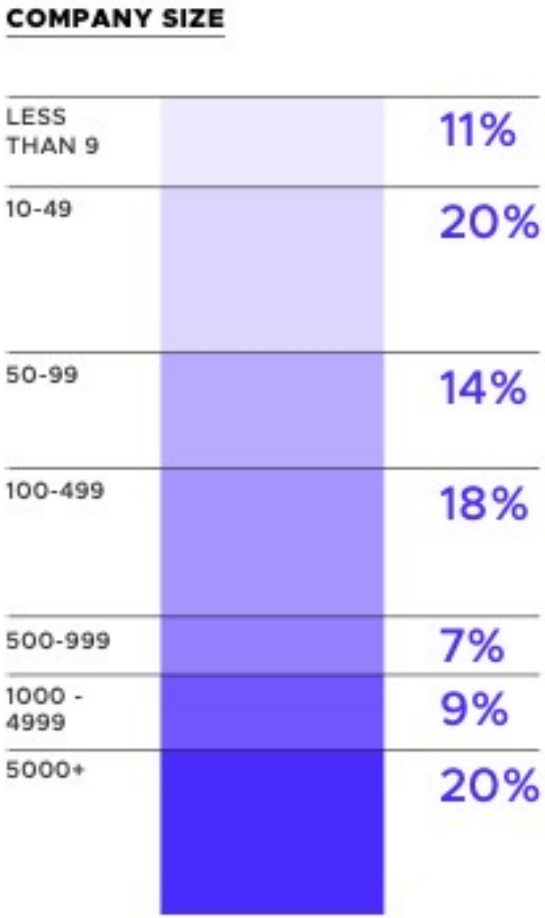
CNCF 成立3周年
195个成员
19个基金会项目
11个孵化项目

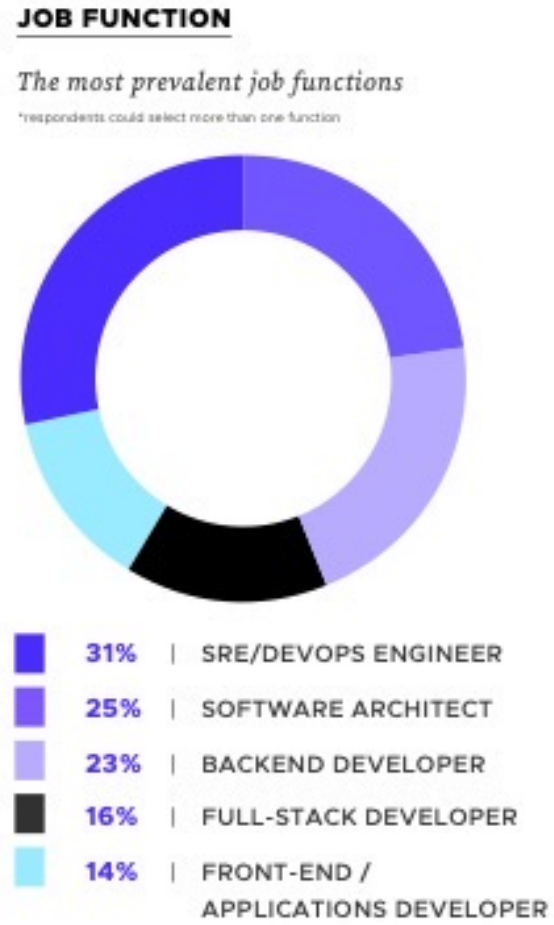
云原生系统

PART 1 / KUBERNETES AND CONTAINERS



Kubernetes在全球范围内的应用





引自CNCF《ANNUAL SURVEY 2021》

云原生系统

According to CNCF's respondents, 96% of organizations are either using or evaluating Kubernetes – a record high since our surveys began in 2016. Particularly interesting is the regional adoption of Kubernetes in production, with emerging technology hub Africa (73%) jumping ahead of other more established tech centers including Europe (69%) and North America (55%). Additionally, 93% of respondents are currently using, or planning to use, containers in production, echoing 92% in our 2020 survey.

96% OF ORGANIZATIONS ARE EITHER USING OR EVALUATING KUBERNETES

ARE YOU USING KUBERNETES?

Yes, in production | Yes, in test poc | Not yet, but we are evaluating | No | Not sure

AFRICA

AUSTRALIA & OCENIA

N. AMERICA

ASIA

EUROPE

S.& C. AMERICA

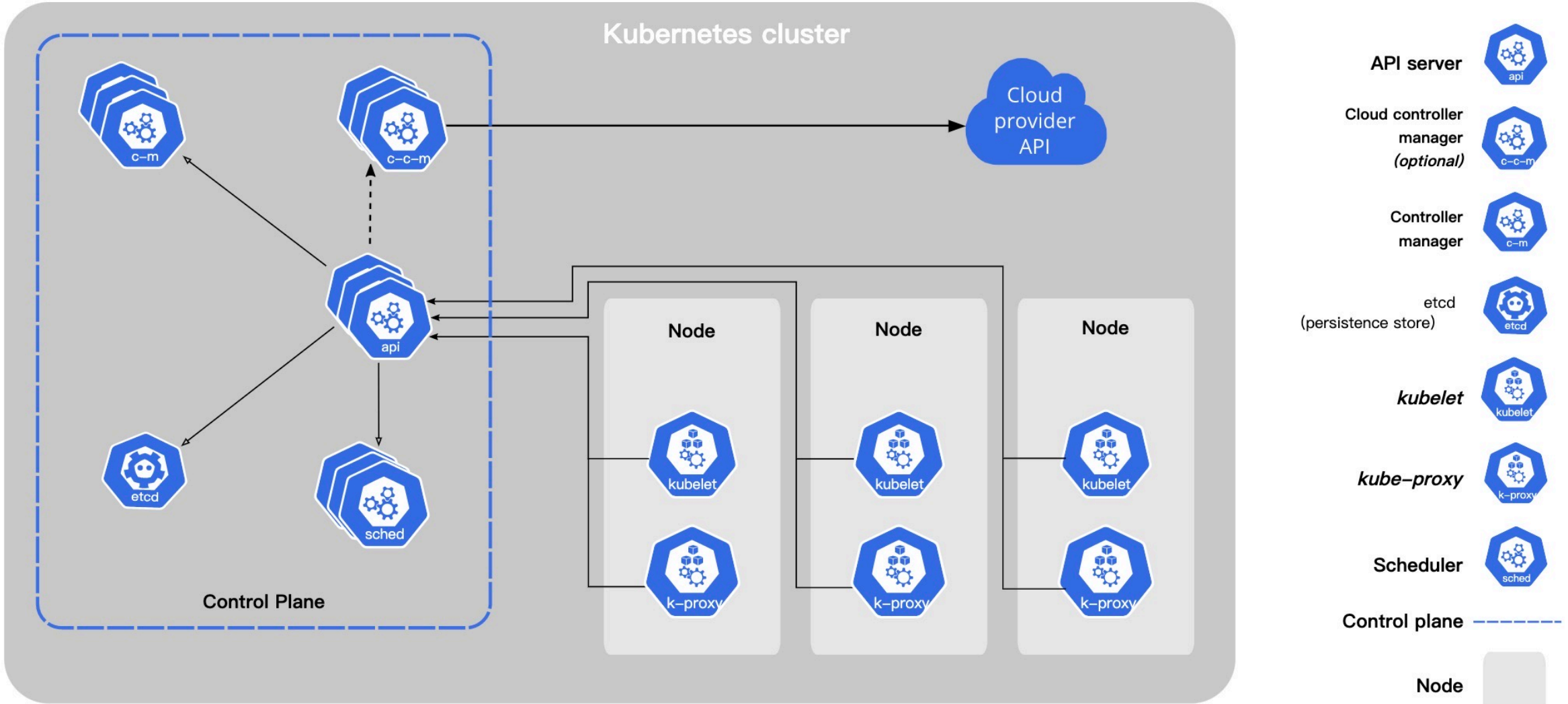
5 ANNUAL SURVEY 2021

CLOUD NATIVE COMPUTING FOUNDATION. 2021

Kubernetes的流行度

云原生系统

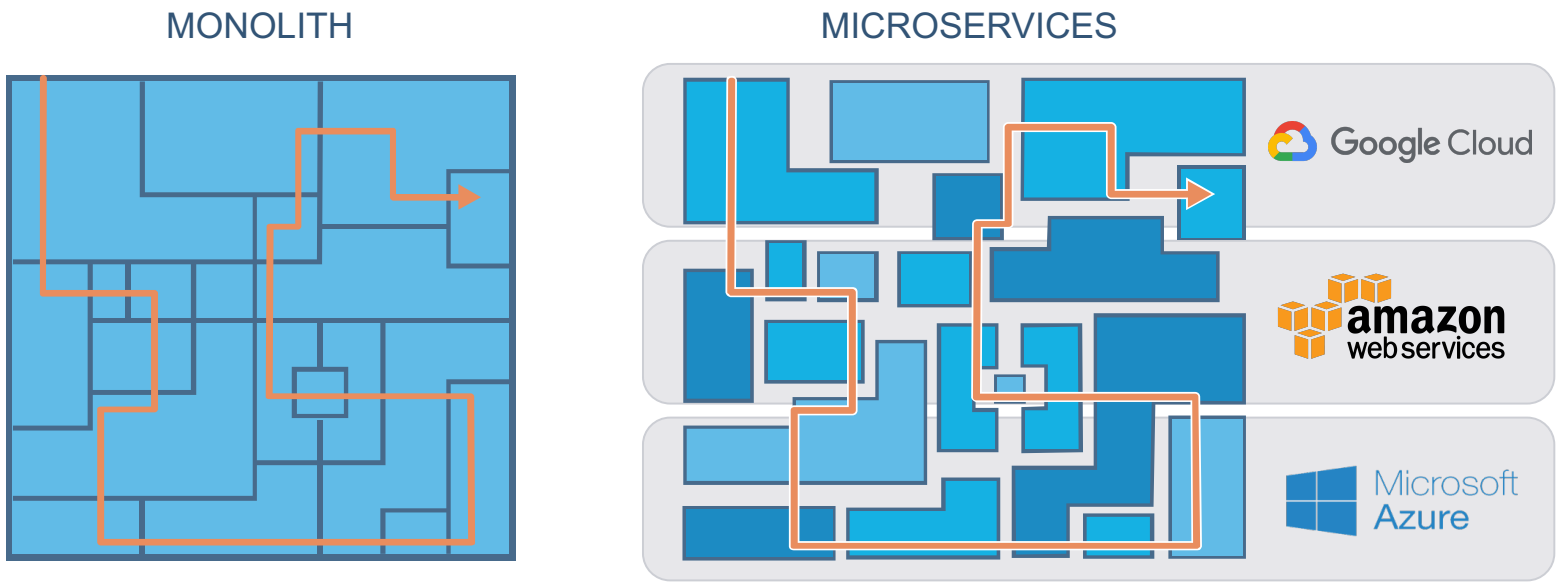
Kubernetes 是一个开源的容器编排引擎，用来对容器化应用进行自动化部署、 扩缩和管理。该项目托管在 CNCF。

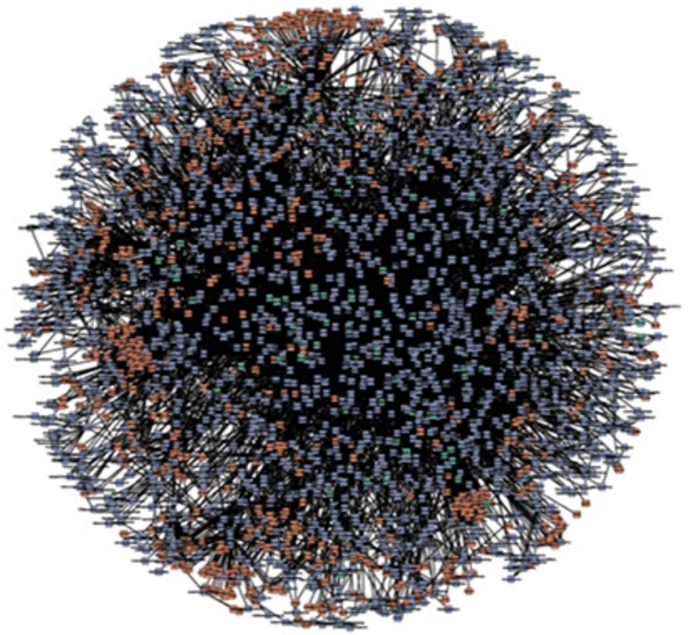


Kubernetes的架构

云原生系统

- 现代云原生软件系统基于微服务架构构建，的规模呈现指数级增加，动辄上千个微服务，例如：WeChat包含近3000种微服务，几十万个服务实例，Netflix超过700种微服务，Uber包含的微服务多达2200个，……；





amazon.com

HOW DO YOU MANAGE APIs?

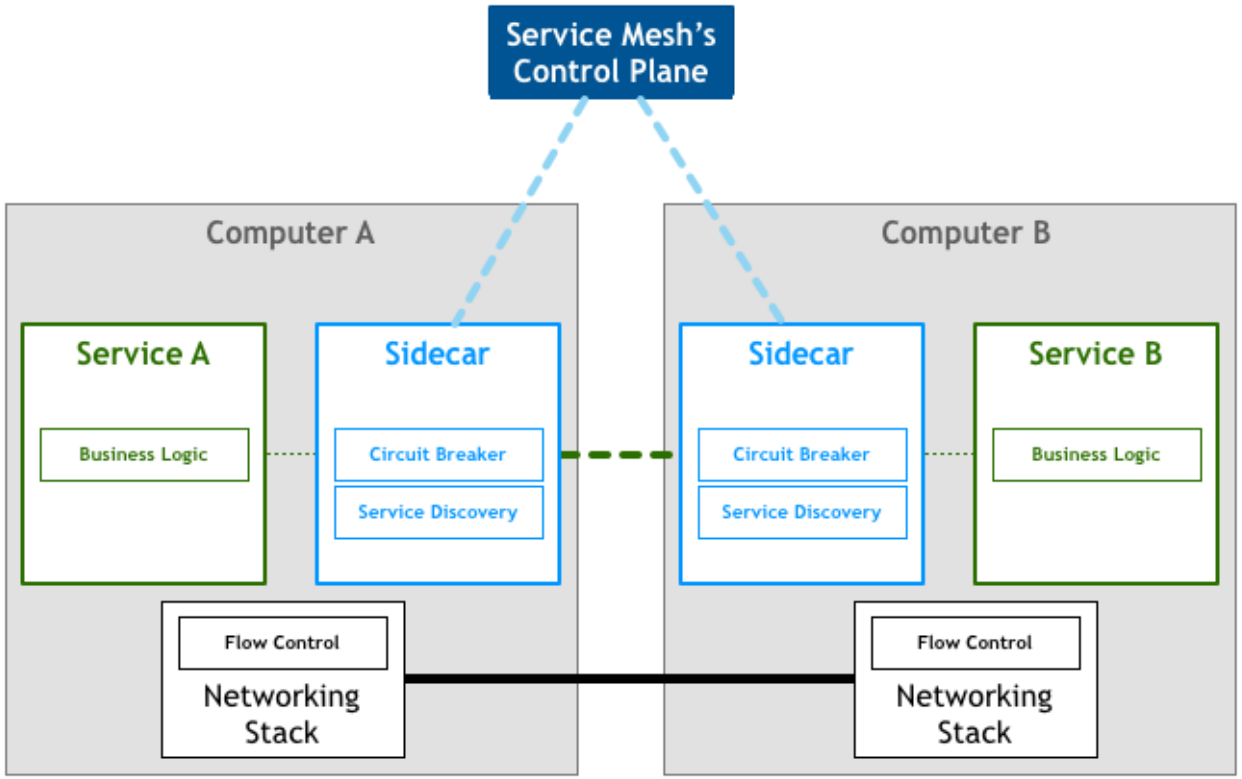
HOW CAN ENFORCE SECURITY?

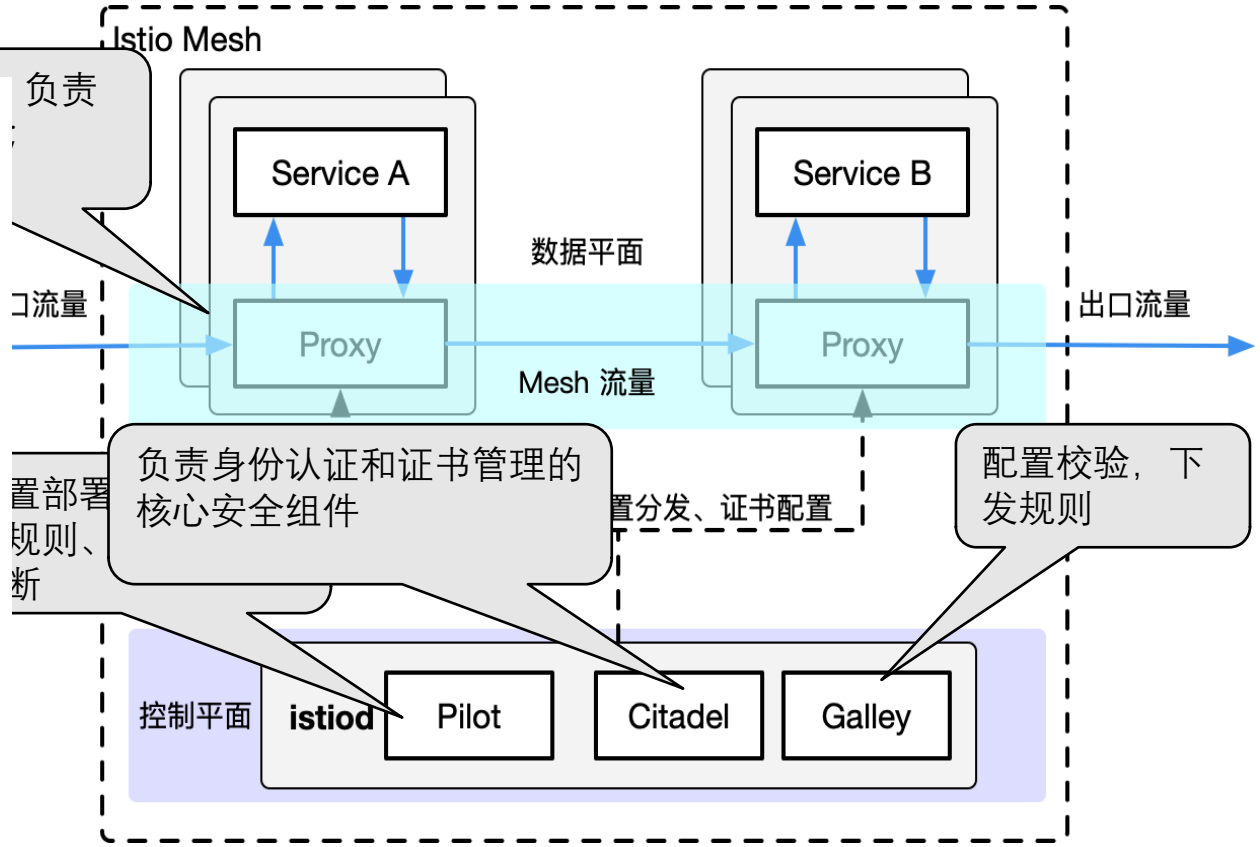
HOW DO YOU OBSERVE?

云原生系统

➢ 服务网格（Service Mesh）

- 加入了控制面，丰富了微服务的治理能力；
- 数据面进行了拓展，具备丰富的包处理能力；
- 微服务治理基本下沉到基础设施；

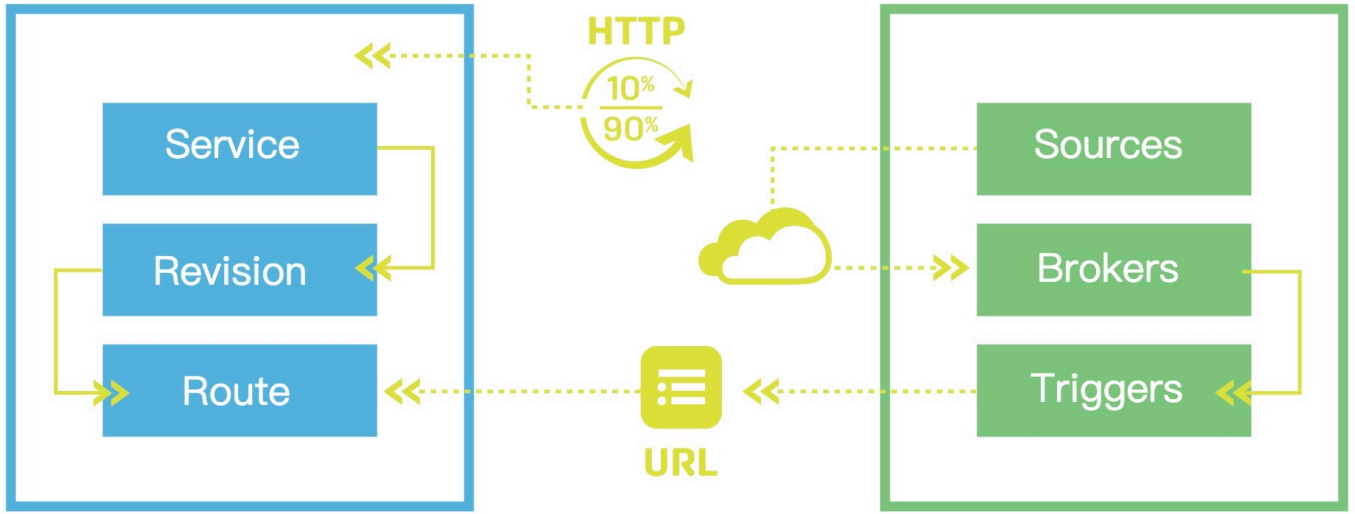




云原生系统

➢ Serverless （FaaS）

Knative 是一个典型的基于K8S和Istio的服务器计算平台，能够以事件触发的形式高并发运行Functions



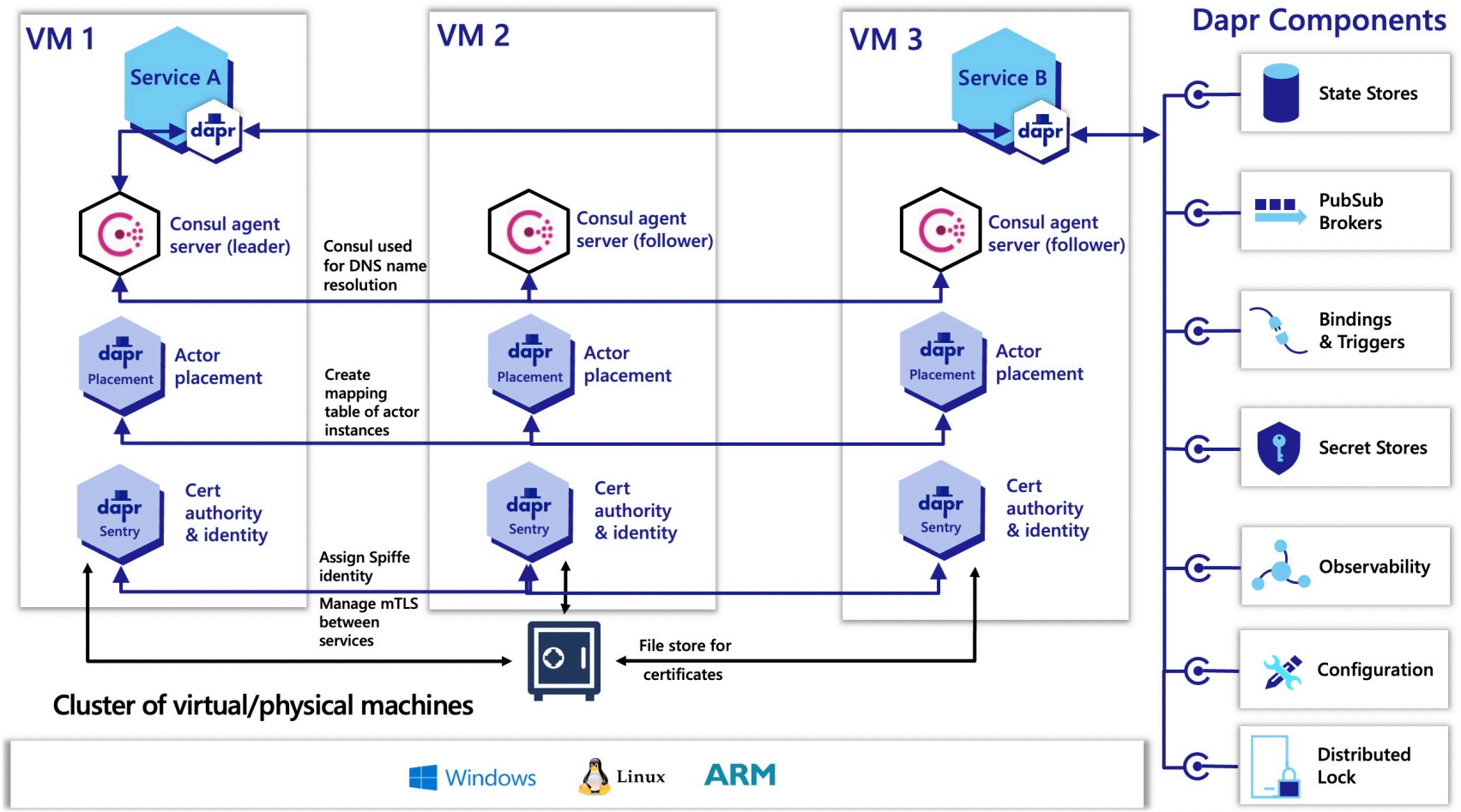
Knative 工作流

Knative 架构

云原生系统

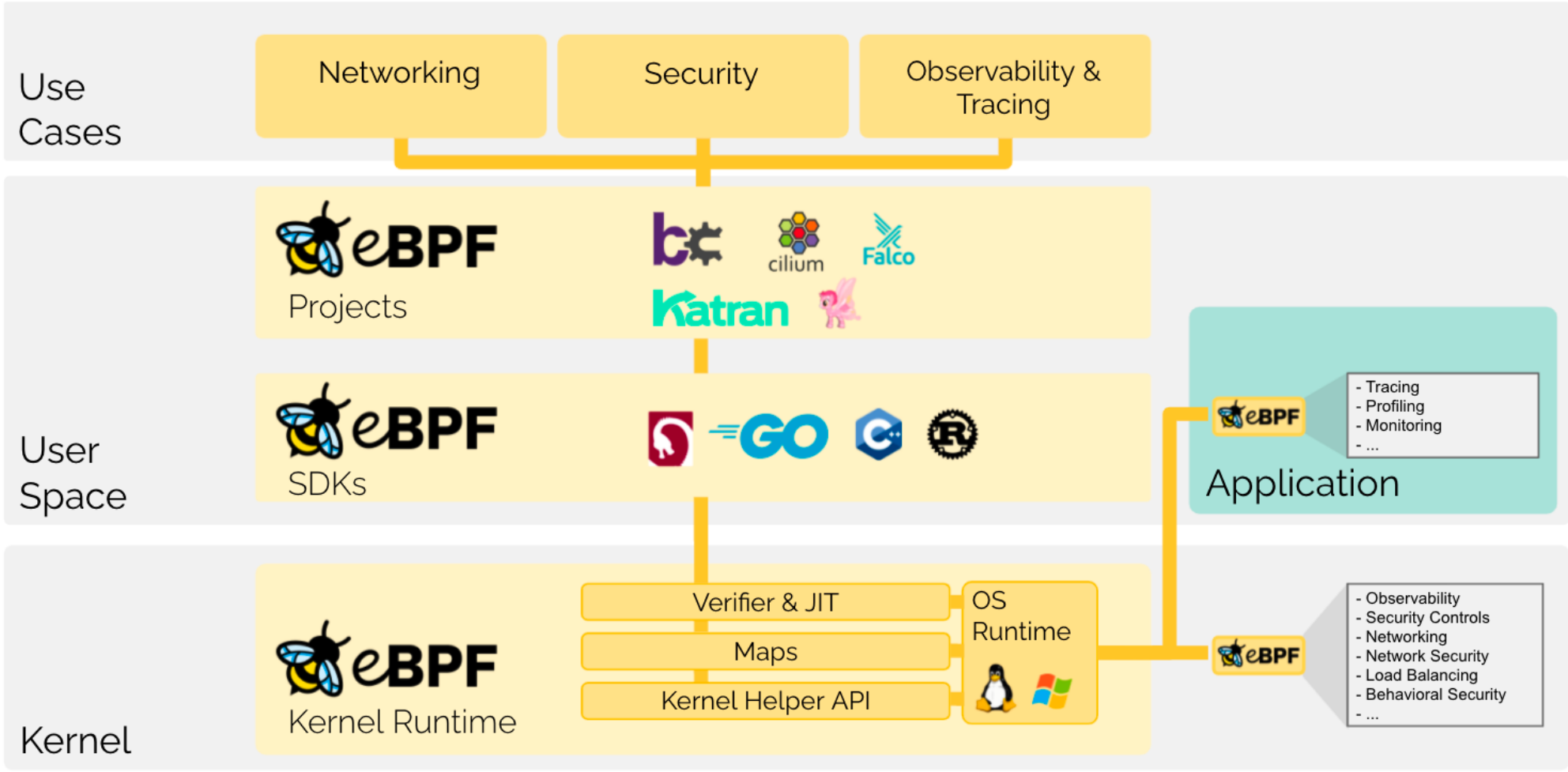
➢ Dapr (Distributed Application Runtime)

分布式应用运行时，一个事件驱动、可移植的运行时用于云上和边缘计算上构建微服务。



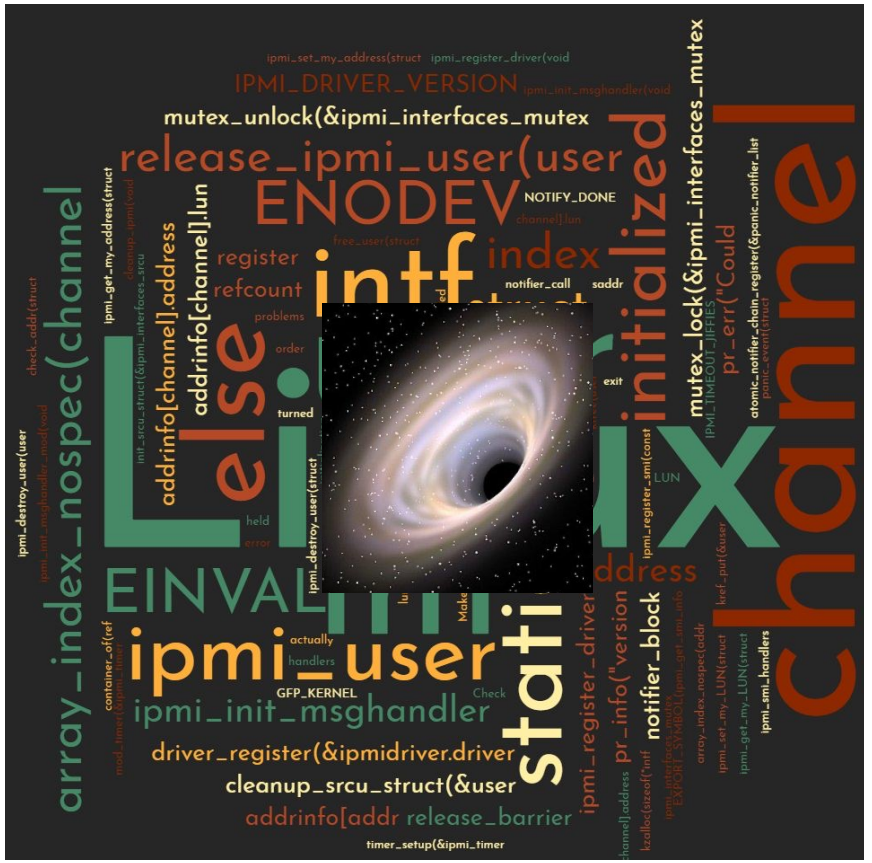
eBPF

➢ eBPF（extended Berkeley Packet Filter），是Linux内核中一种高度灵活、高效的类似虚拟机的结构，允许以安全的方式在各种挂钩点执行字节码。它用于许多Linux内核子系统，最突出的是网络、跟踪和安全（例如沙箱）。



就像宇宙中的“虫洞”提供一条一个时空到达另一个时空的捷径，eBPF是操作系统用户空间和内核空间的“虫洞”

宇宙“虫洞”



内核“虫洞”

➢ eBPF正在努力让操作系统内核可编程化，成为云原生时代软件系统的“瑞士军刀”；

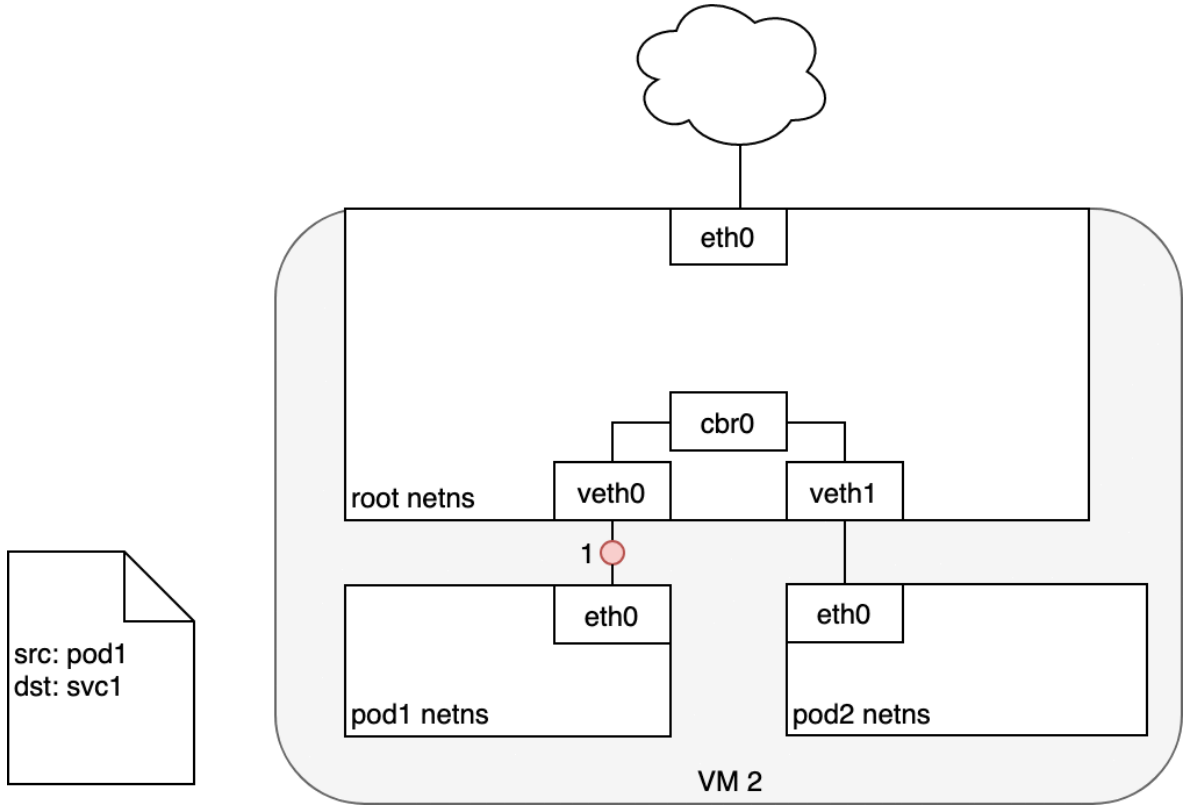
02

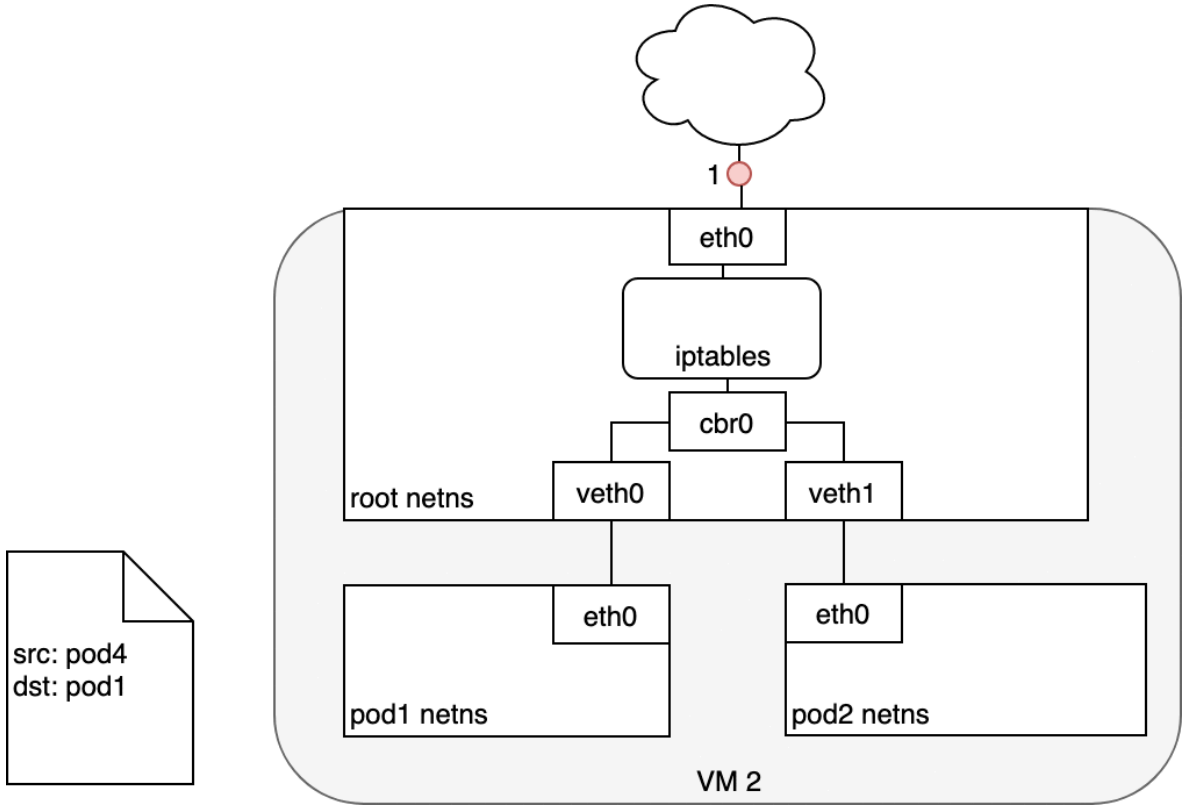
服务网格数据面优化

服务网格数据面优化

➢ 性能问题

❶ 基于K8S的容器管理平台产生了多层复杂的虚拟网络，网络栈复杂度增加，延长了端到端的请求处理时间





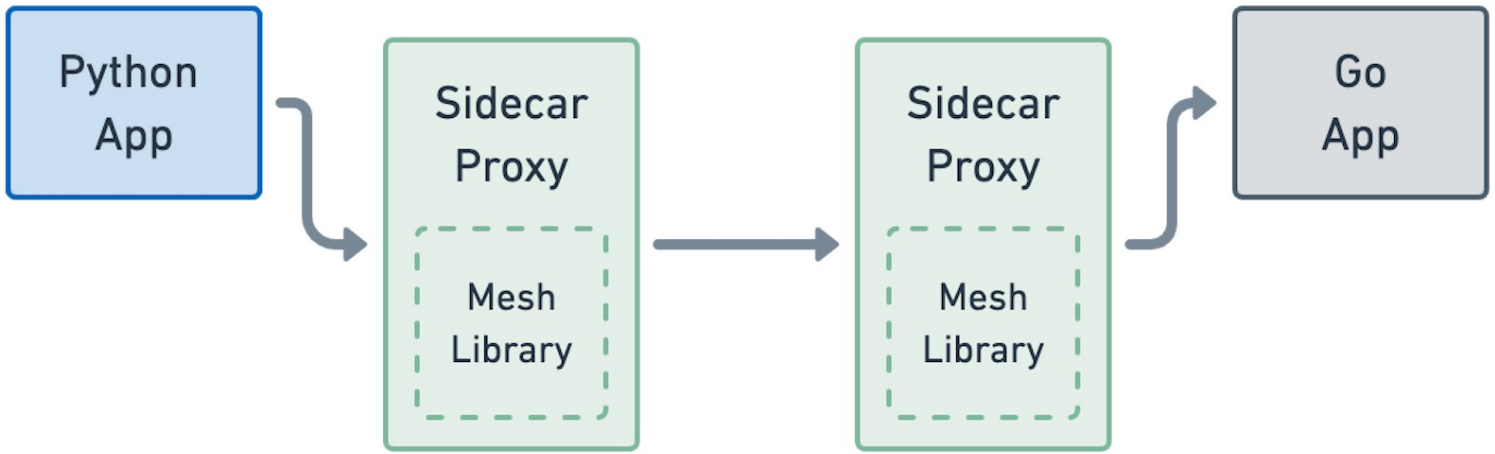
服务网格数据面优化

➢ 性能问题

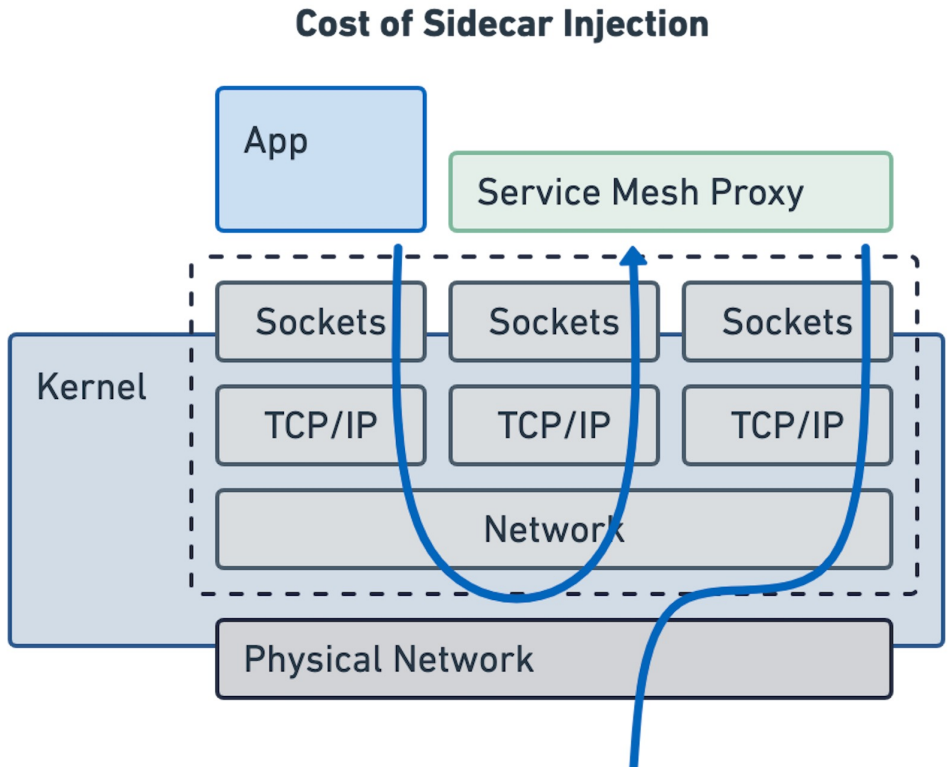
❷ Sidecar的引入增加了请求在网络协议栈传输路径，增加了延迟;

❸ Iptables的线性搜索增加了请求延迟；

❹ Sidecar中请求频繁的用户态与内核态之间的切换增加了请求延迟；

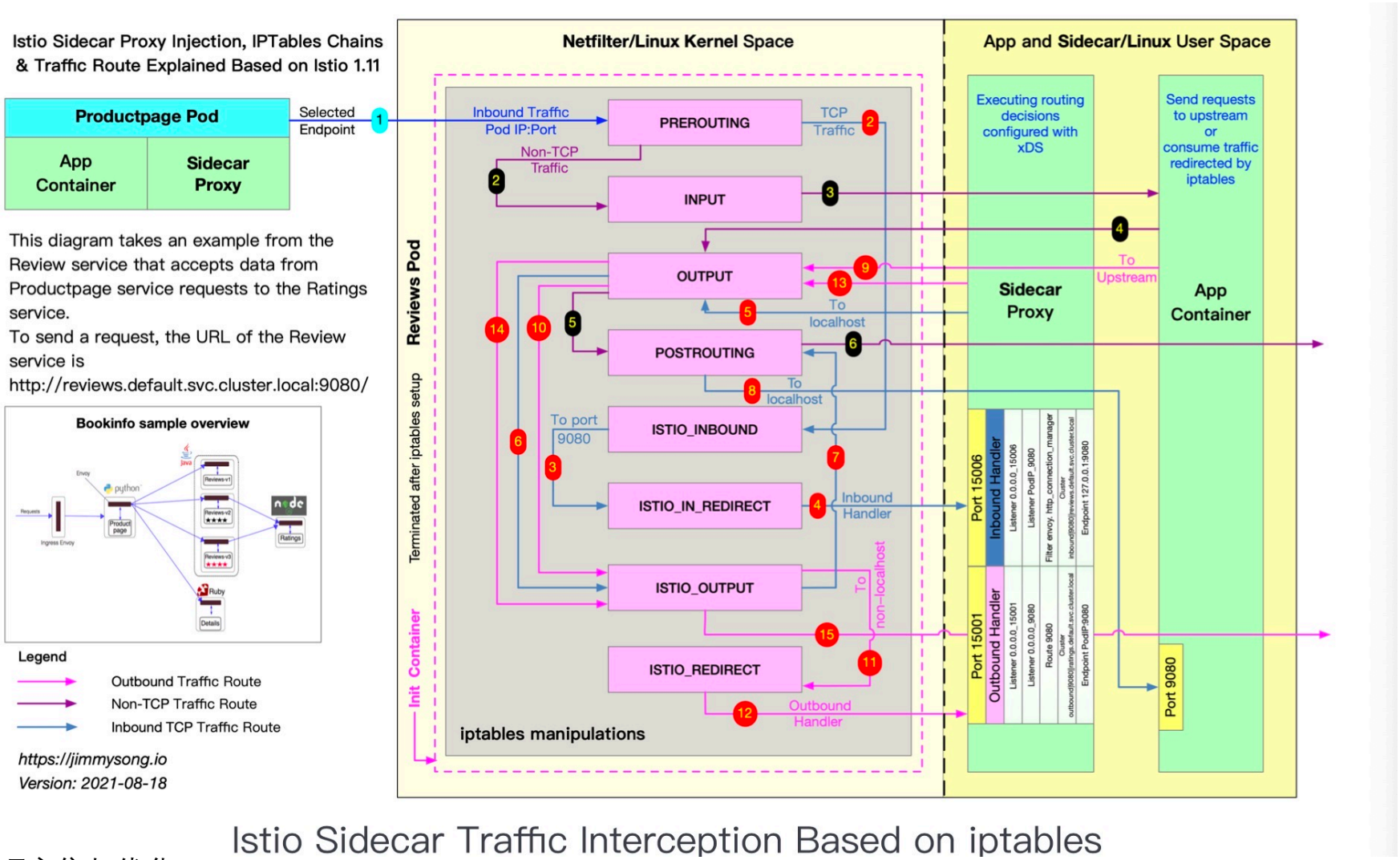


Service mesh is is embedded in a proxy running outside of the application.



服务网格数据面优化

➢ 性能问题

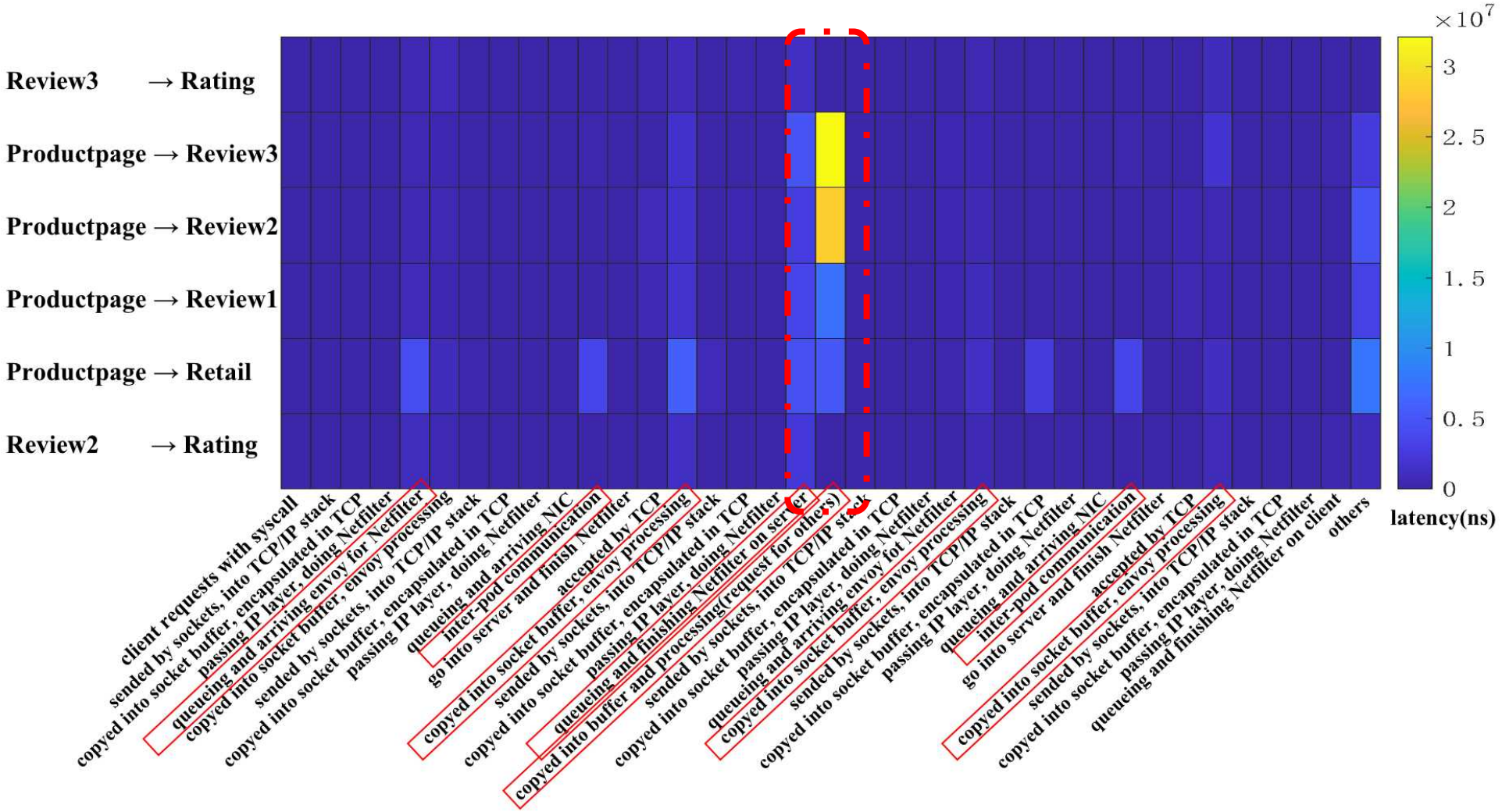


Istio Sidecar Traffic Interception Based on iptables

服务网格数据面优化

➢ 性能剖析

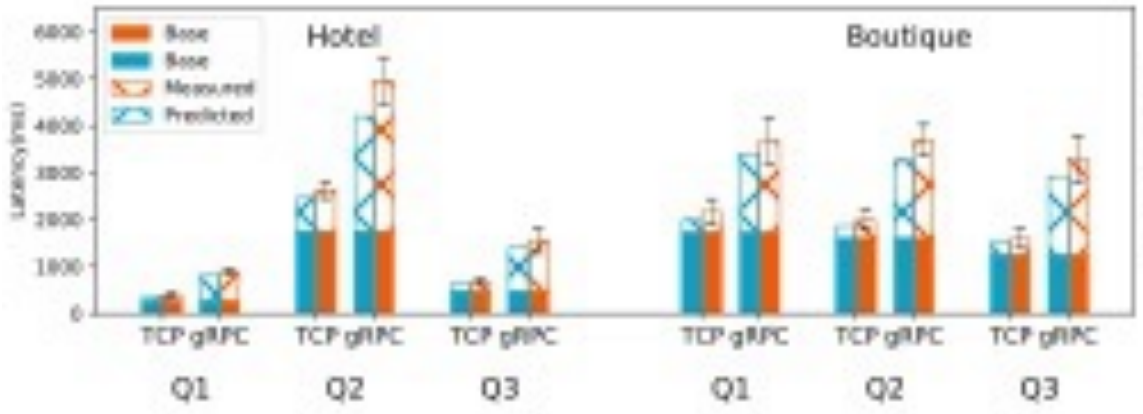
- 以 Isitio 提供的 BookInfo 作为 Benchmark；
- 跟踪请求的指向性过程以及涉及到的系统调用；
- 观察请求延迟在内核中的分布；
- 结论：请求在envoy网络协议栈中的时延较长；



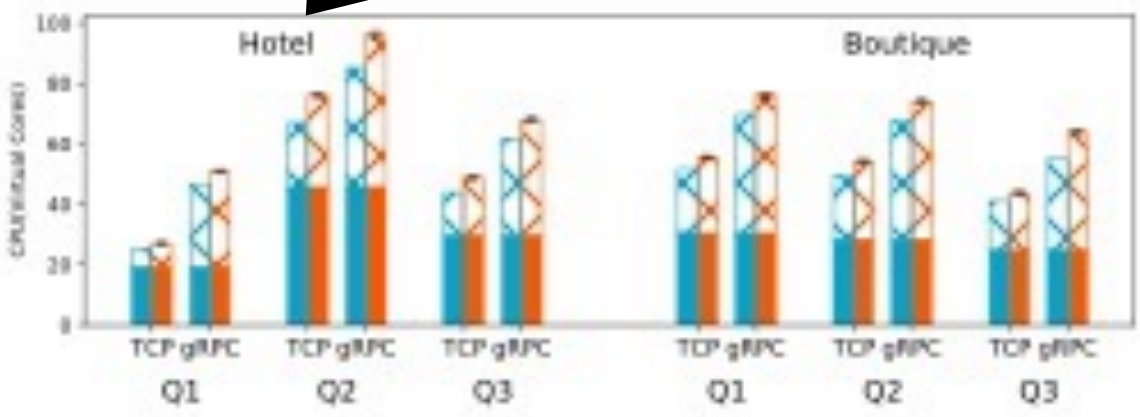
服务网格数据面优化

➢ 性能剖析

引自论文《Dissecting Service Mesh Overheads》



(a) Latency.



(b) CPU Usage.

Sidecar带来的延时和CPU开销

	Latency (us)			CPU Usage (Virtual Cores)		
	TCP	HTTP	gRPC	TCP	HTTP	gRPC
IPC	11.59 (30%)	12.75 (8%)	13.04 (7%)	0.49 (15%)	0.51 (5%)	0.55 (4%)
Read	8.14 (16%)	9.01 (5%)	9.37 (5%)	0.26 (8%)	0.29 (3%)	0.30 (2%)
Write	13.22 (34%)	13.80 (8%)	14.35 (7%)	0.45 (14%)	0.48 (5%)	0.57 (4%)
Notification	1.33 (3%)	1.27 (1%)	1.35 (1%)	0.26 (8%)	0.27 (3%)	0.26 (2%)
Protocol Parsing	-	117.35 (70%)	142.38 (73%)	-	6.00 (62%)	9.76 (71%)
Protocol Other	4.25 (11%)	13.07 (8%)	14.39 (7%)	1.79 (55%)	2.09 (22%)	2.34 (17%)
Total	38.63	167.25	194.79	3.25	9.65	13.79

Sidecar中不同的组件对不同协议的延迟贡献

服务网格数据面优化

➢ 解决方案

- 使用eBPF sock_ops中的redirect能力实现节点内socket数据直通；
- 使用TC/XDP redirect实现跨节点的网络接口直通；

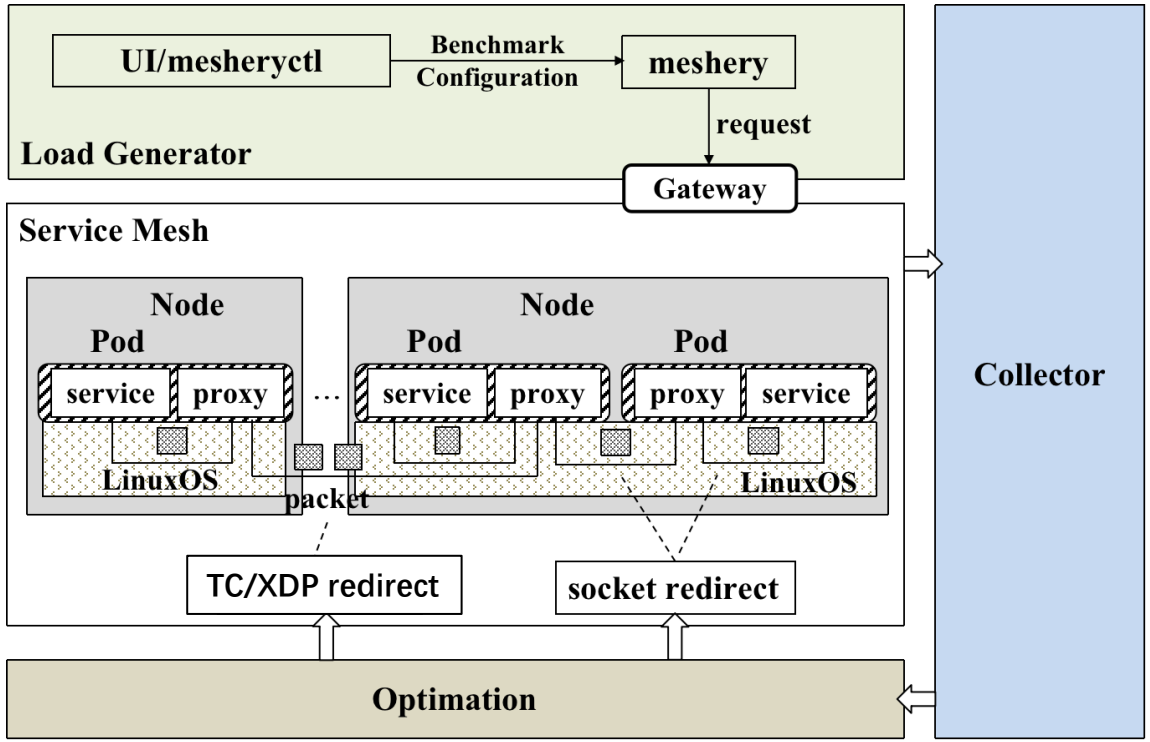
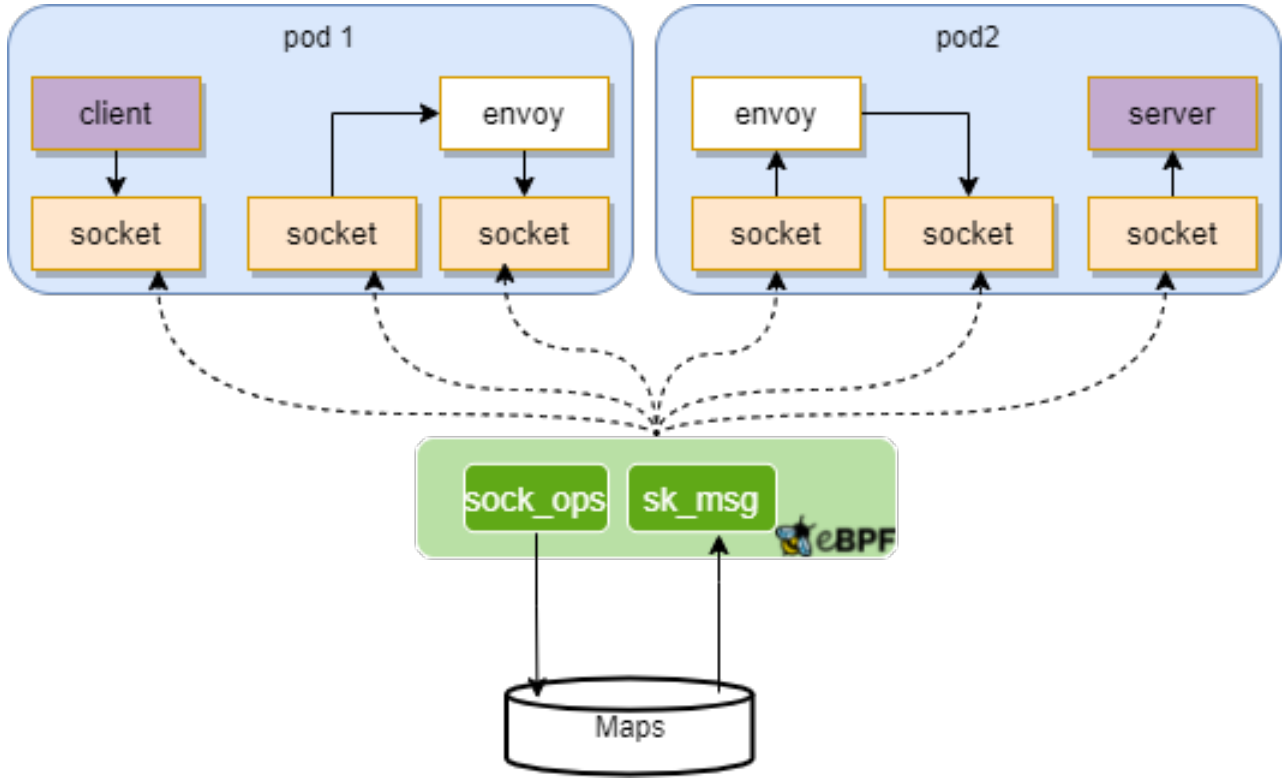


Figure 5: The overall architecture of our system.

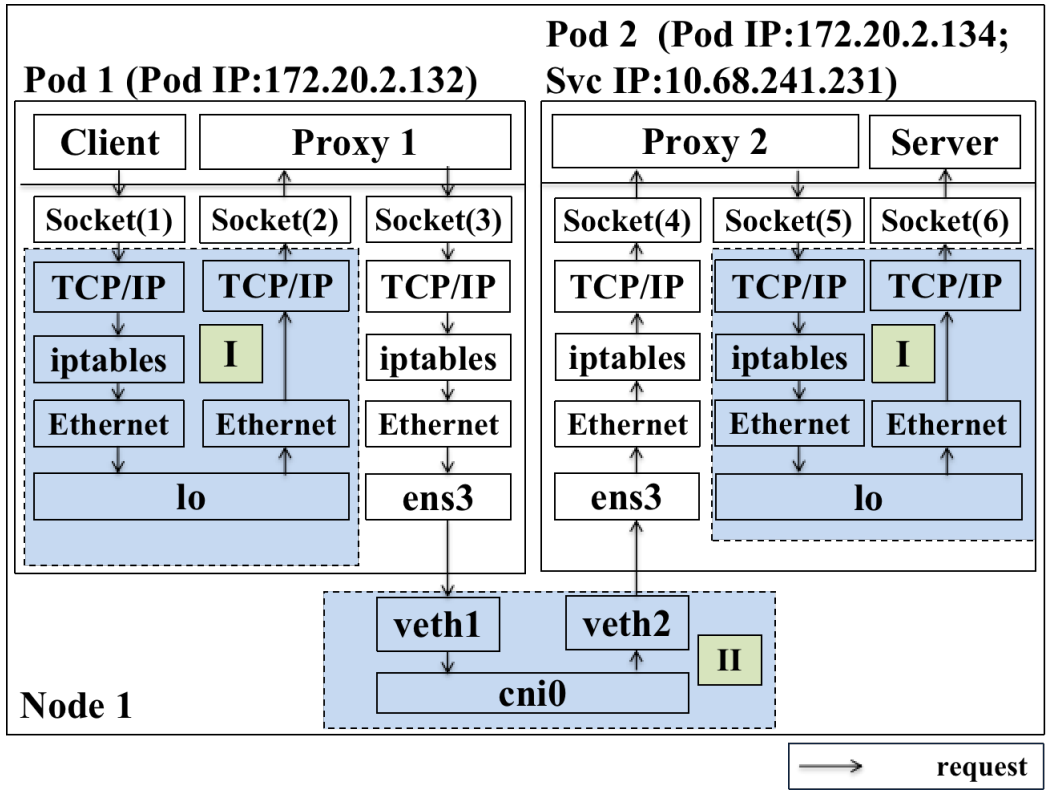


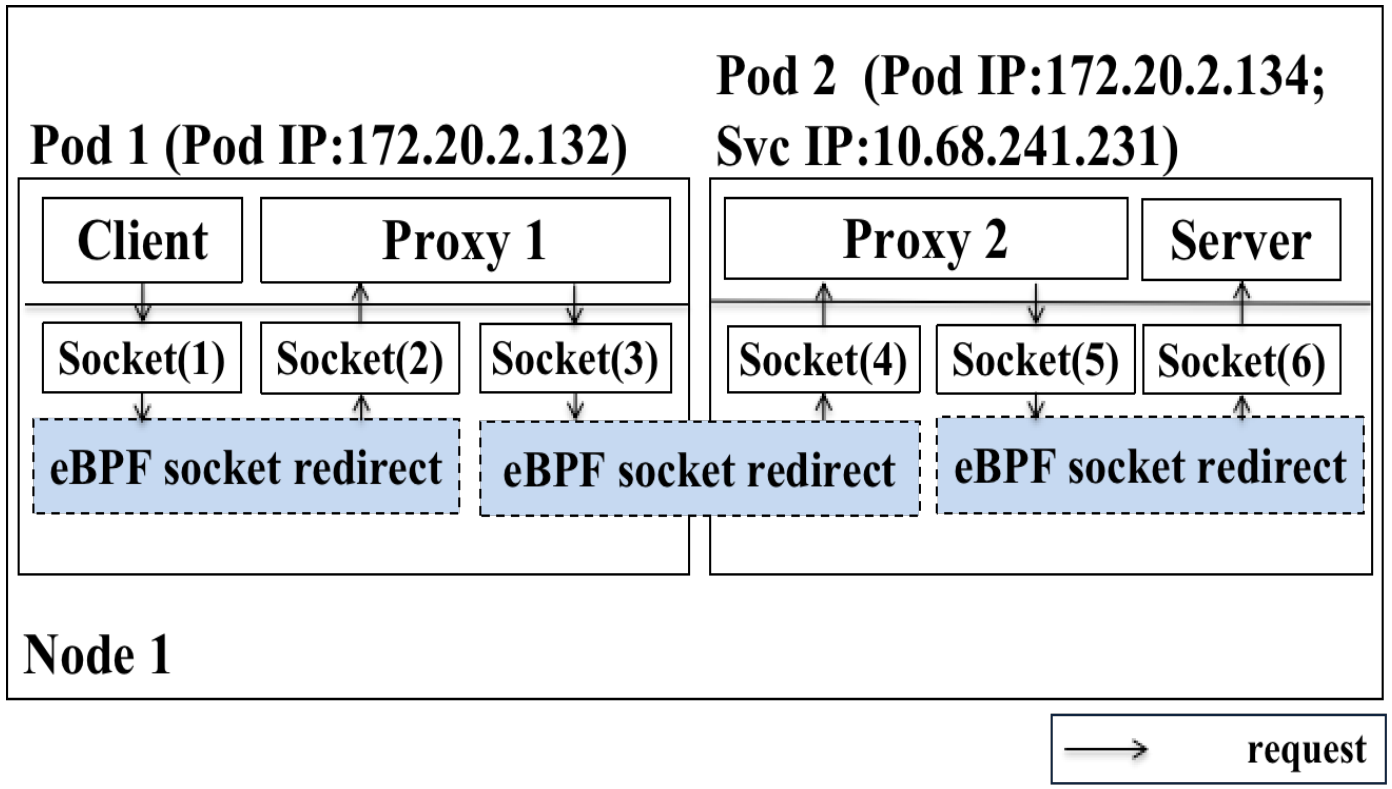
优化的基本思路

服务网格数据面优化

➢ 解决方案

◘ 单节点网络优化方法





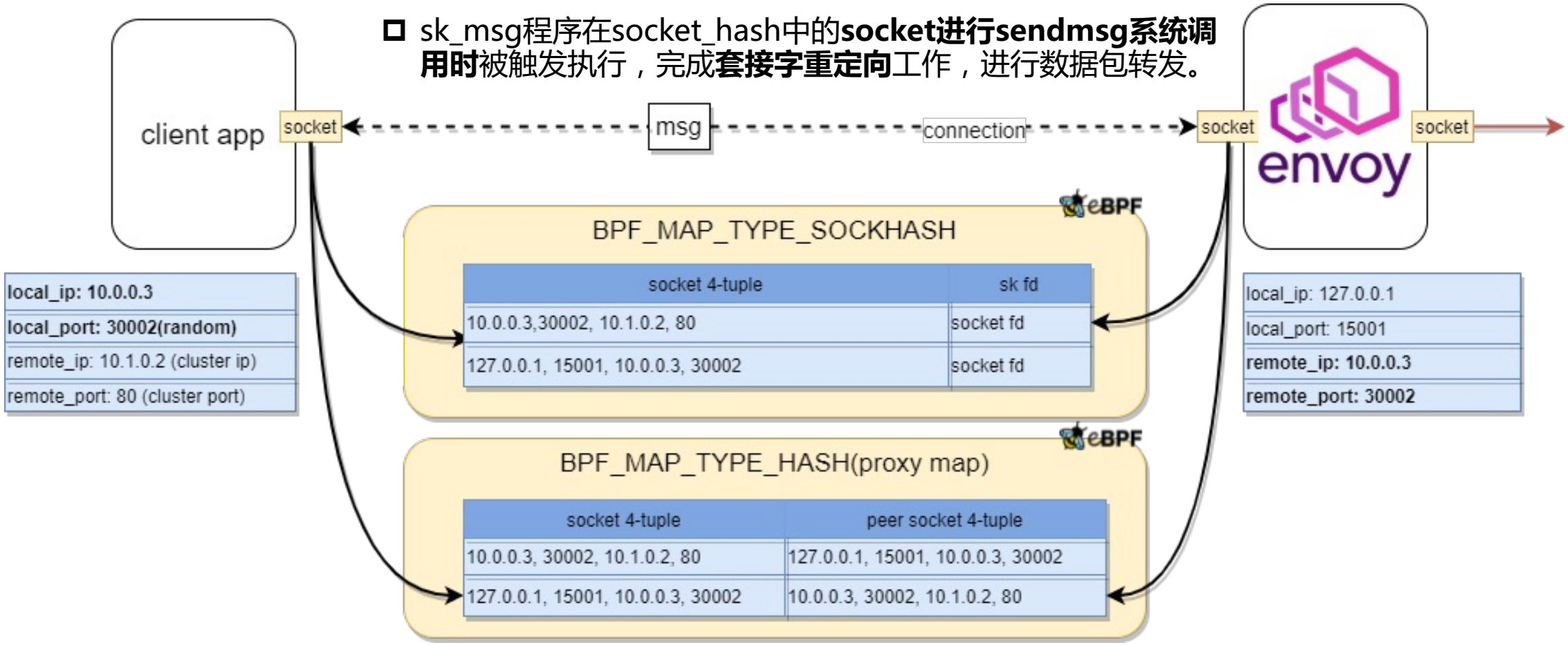
单节点上pod->proxy以及proxy->proxy的网络传输栈

单节点网络优化结果

服务网格数据面优化

➢ 解决方案

- 单节点网络优化方法
  - ➢ 套接字重定向
    - sock_ops 程序附加到套接字**连接建立**的钩子点，获取套接字选项信息，更新socket_hash
    - sk_msg程序在socket_hash中的**socket进行sendmsg系统调用时**被触发执行，完成**套接字重定向**工作，进行数据包转发。

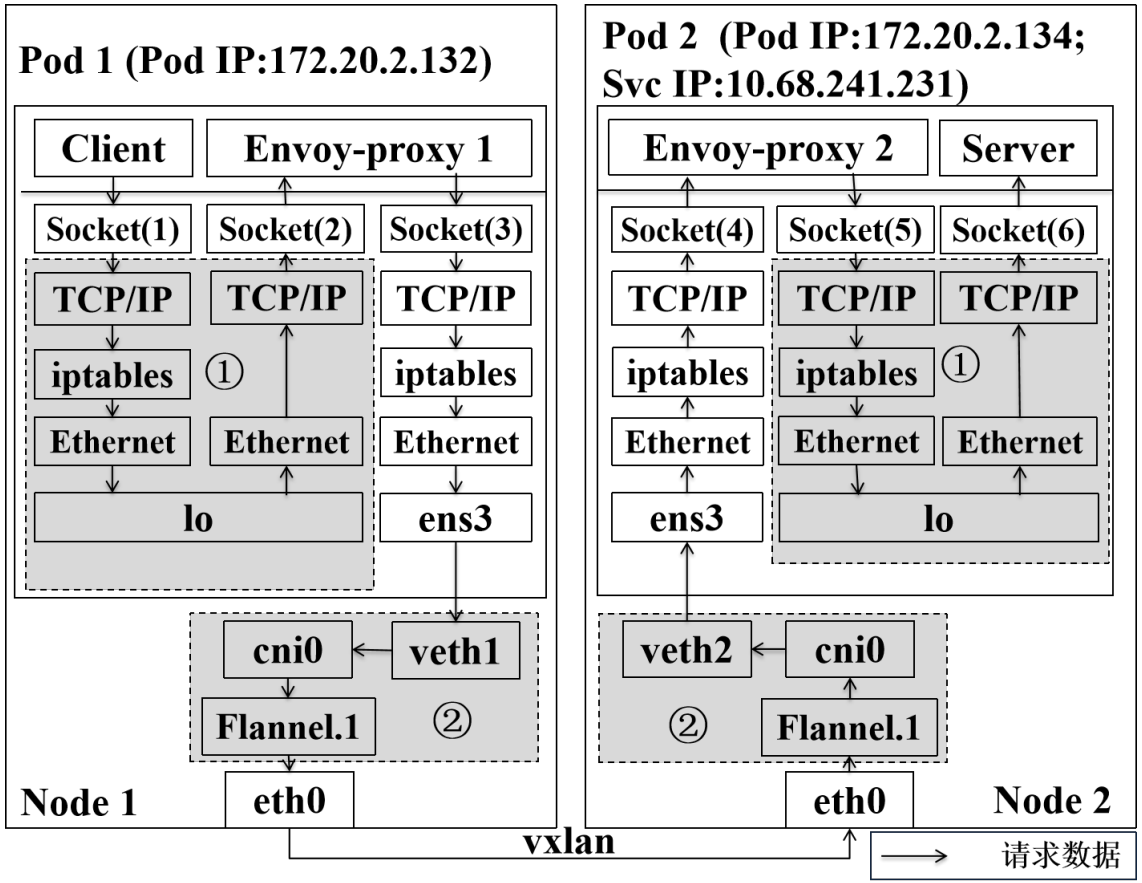


服务网格数据面优化

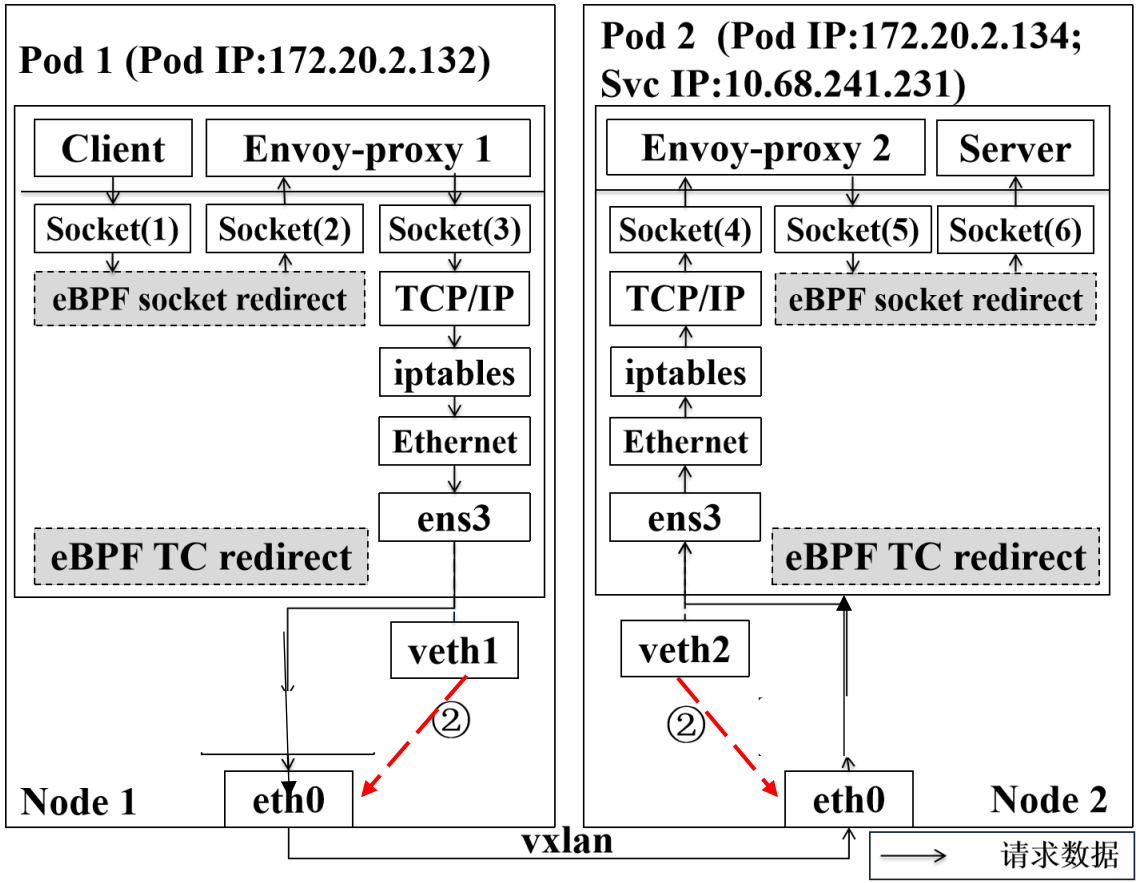
➢ 解决方案

□ 单节点网络优化方法

利用DPDK用户态协议栈实现CPU Bypass也可加速，比如网易方案



跨节点proxy->proxy的网络传输



跨节点proxy->proxy的优化结果

服务网格数据面优化

➢ 解决方案

- 套接字重定向
  - 对部署在相同节点上的pod间通信优化
- TC层重定向
  - 对部署在不同节点上的pod间通信优化
  - Traffic Control（简称 TC）是 Linux 负责**流量控制**的模块，通过在网卡设备上建立队列规则，建立数据包队列，定义队列中数据包的发送方式，从而实现流量控制
  - TC 中的队列规则类型 **clsact** 可作为钩子点挂载用户自定义的 eBPF 程序。**Ingress** 处理入口流量，**Egress** 处理出口流量。
  - 将数据包重定向到另一个网卡设备上，以此实现包转发

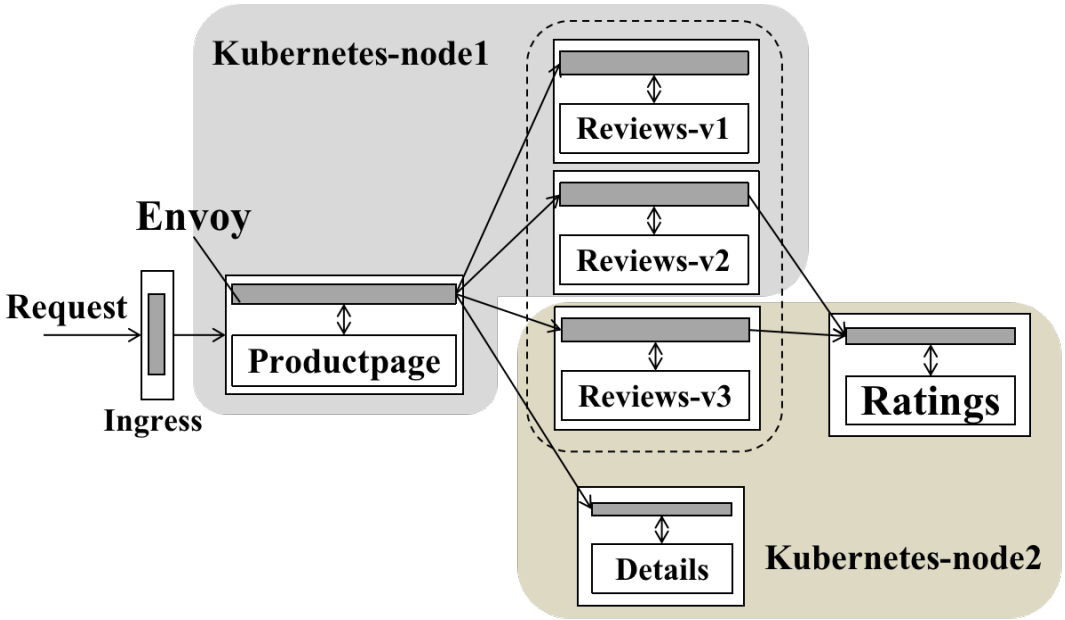
```
return bpf_redirect(ifindex, 0);
```

替换成XDP_redirect实现

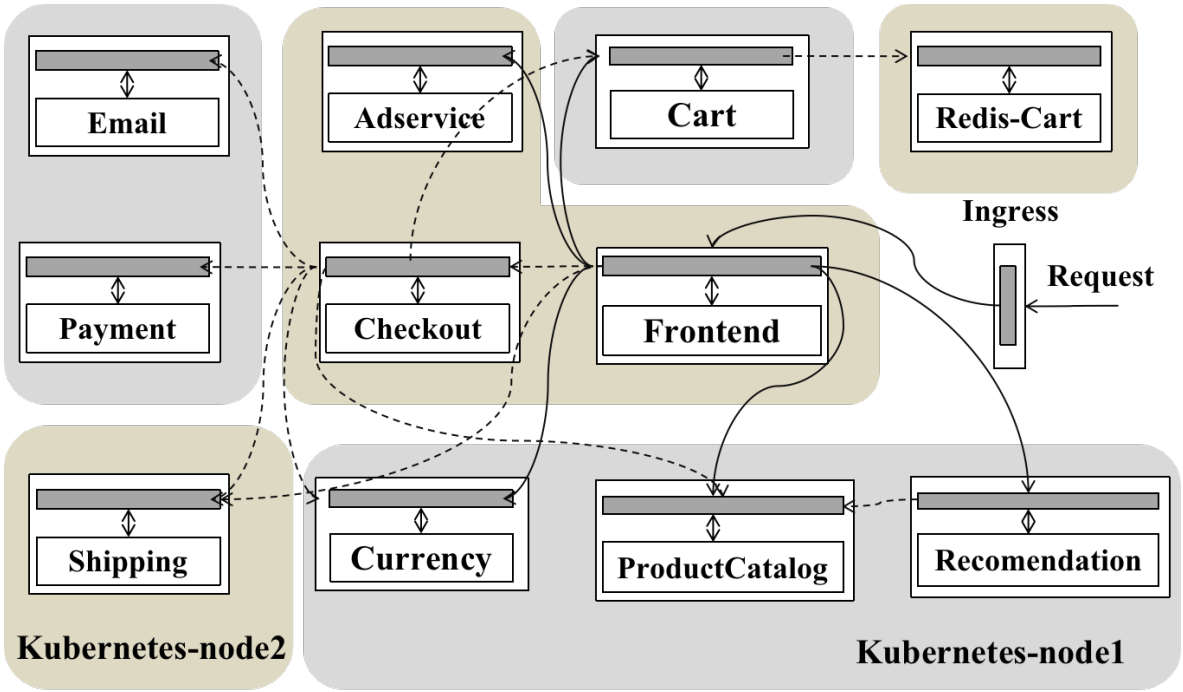
服务网格数据面优化

➢ 测试方案

- 选择BookInfo和Hipstershop作为Benchmark



(a) The architecture of Bookinfo.

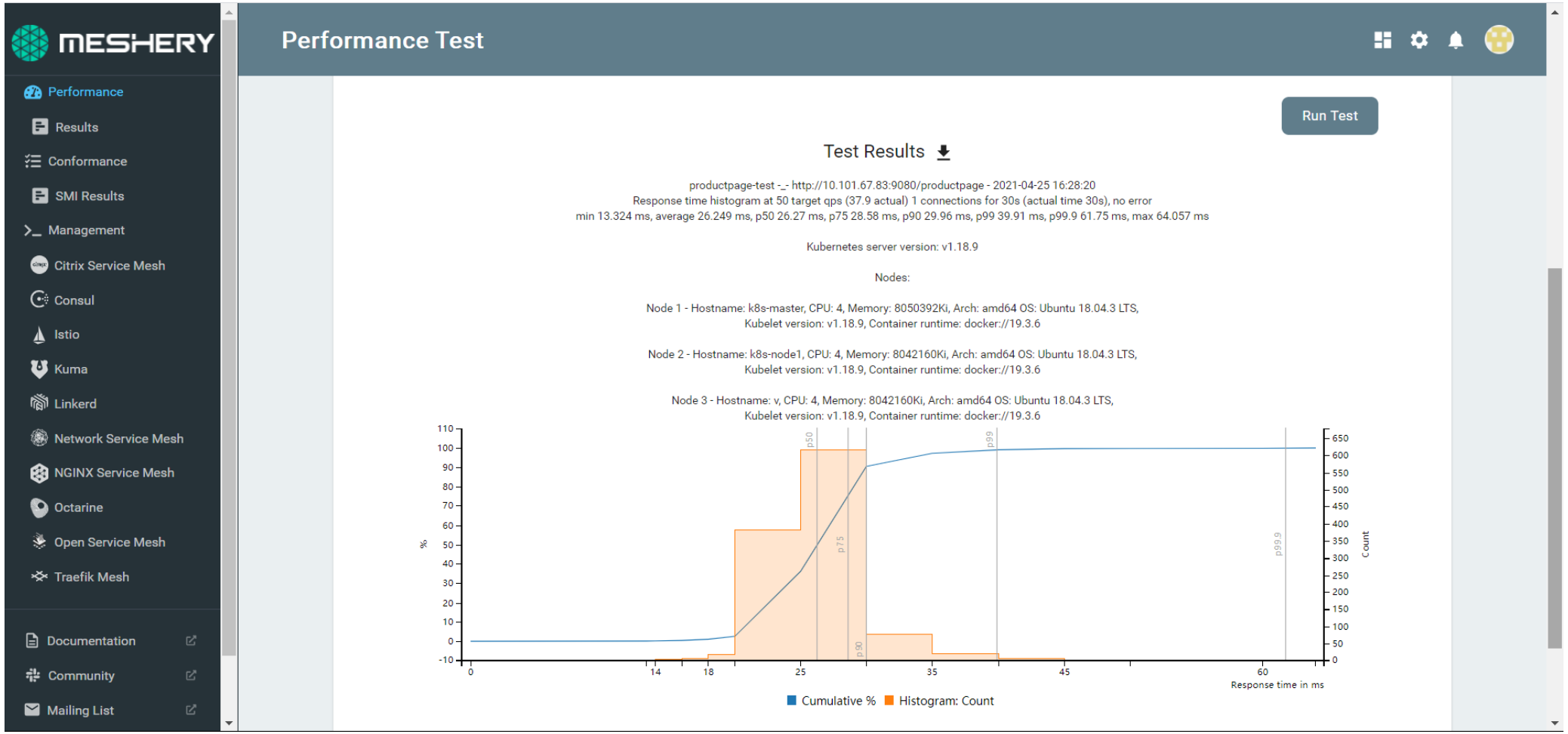


(b) The architecture of Histershop.

服务网格数据面优化

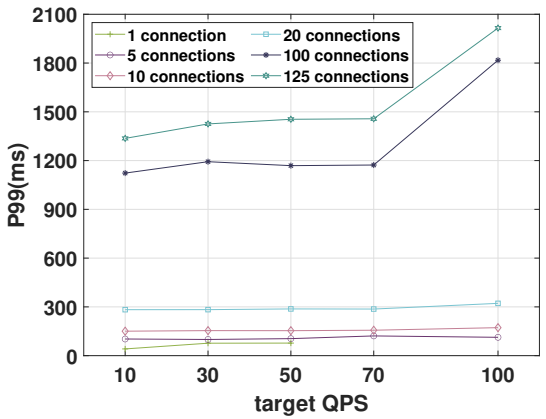
➢ 测试方案

- Meshery作为测试工具

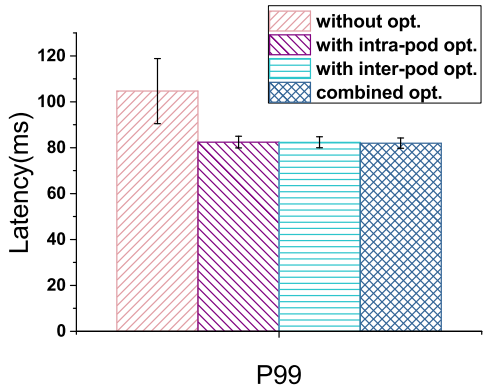


服务网格数据面优化

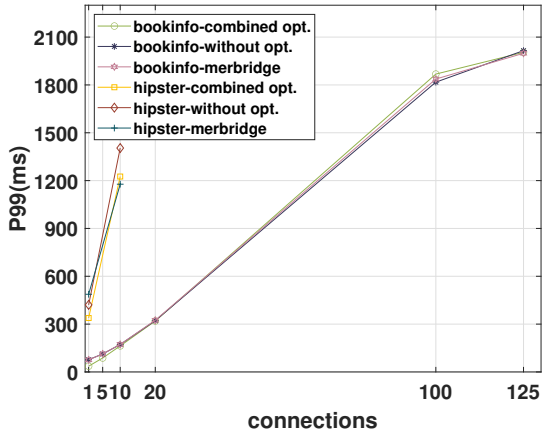
➢ 测试结果



(a) Service capacity of the Bookinfo application.

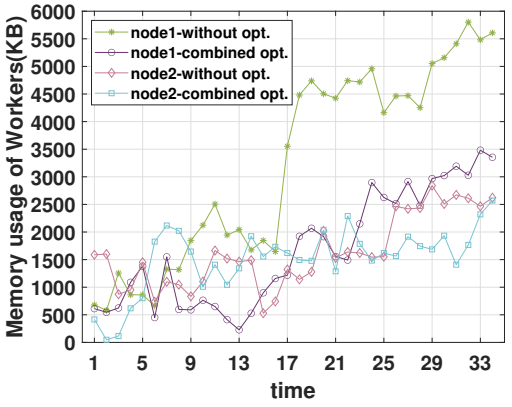


(b) Benchmark results under low load.

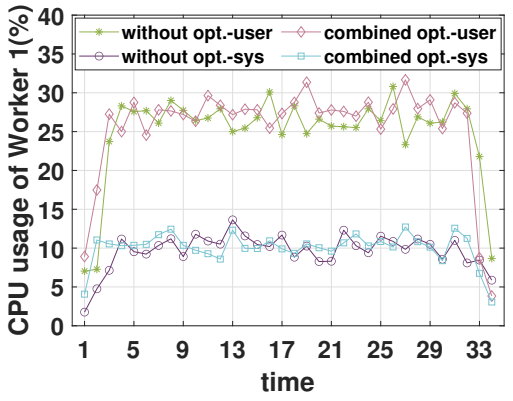


(c) Benchmark results under high load.

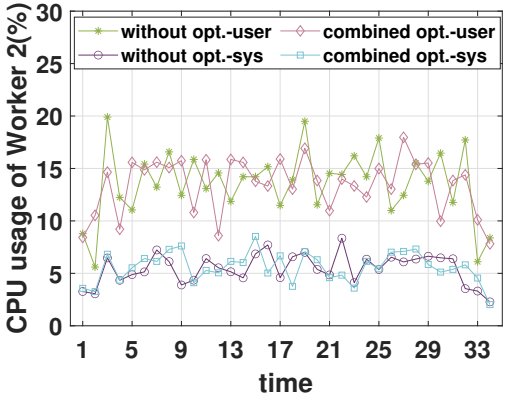
Figure 7: The service capacity of Bookinfo and tail latency under different loads.



(a) Memory usage on nodes.



(b) CPU utilization on Kubernetes-node1.



(c) CPU utilization on Kubernetes-node2.

服务网格数据面优化

➢ 测试结果

基础测试

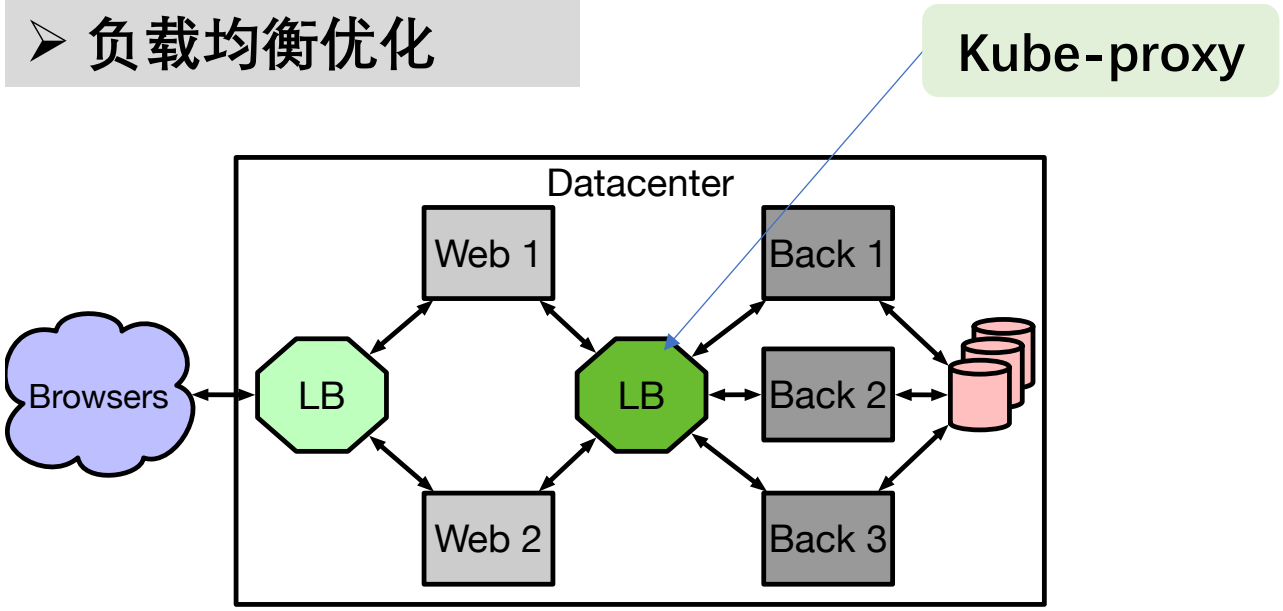
负载配置	min	P50	P99	max	aver
-c 1 -n 10	0.022408023	0.0253116	0.0551898	0.055515393	0.028807097
-c 10 -n 100	0.028957291	0.08271109	0.1307253	0.133949204	0.08539521
-c 100 -n 1000	0.281689738	0.8682089	1.30034	1.356995247	0.866798767
-c 1000 -n 10000	0.633421129	8.698837778	16.04225556	18.63808859	8.648890256

优化测试

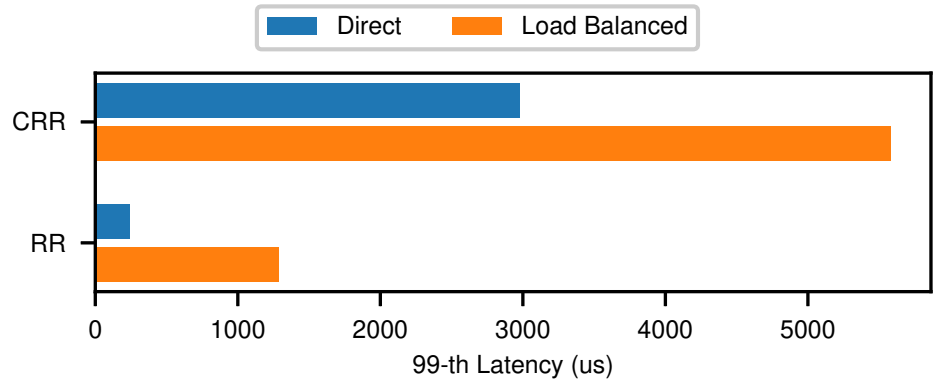
负载配置	min	P50	P99	max	aver
-c 1 -n 10	0.013663003	0.01489766	0.02504628	0.025114251	0.015779754
-c 10 -n 100	0.040168173	0.07502998	0.09836319	0.100887824	0.074587522
-c 100 -n 1000	0.181174838	0.7690795	1.172265	1.435065915	0.774125269
-c 1000 -n 10000	0.394261101	8.723534	15.0543	16.86270997	8.24495267

服务网格数据面优化

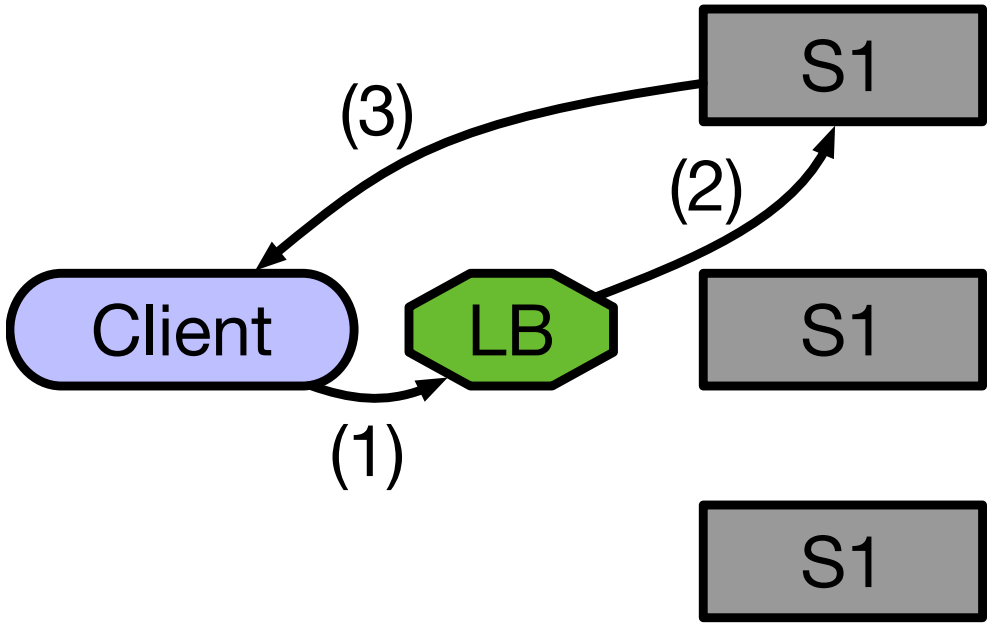
➢ 负载均衡优化



集群内部的负载均衡



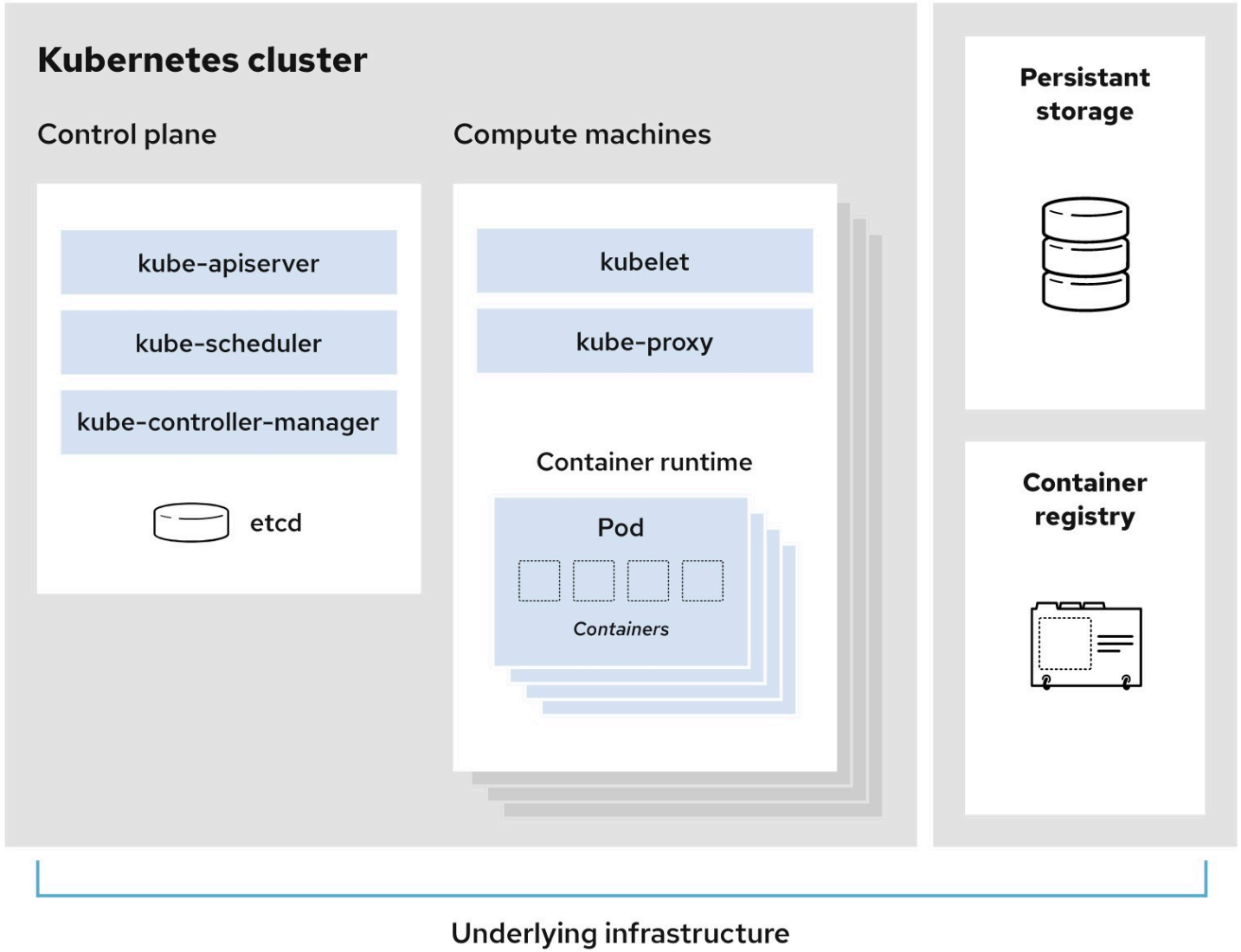
负载均衡开销



(b) L4 Load Balancing with DSR

服务网格数据面优化

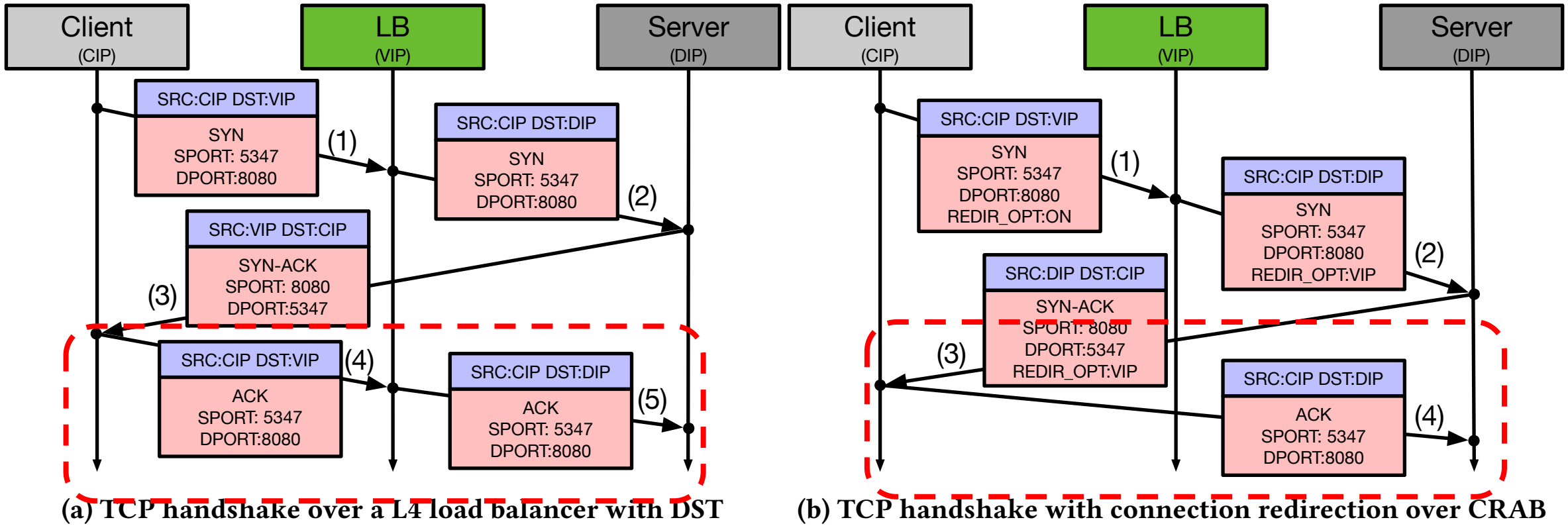
➢ K8S架构



服务网格数据面优化

➢ 负载均衡优化

- 优化后的负载均衡方案，ByPass LB



(a) TCP handshake over a L4 load balancer with DST

(b) TCP handshake with connection redirection over CRAB

服务网格数据面优化

➢ 负载均衡优化

- 优化后的性能

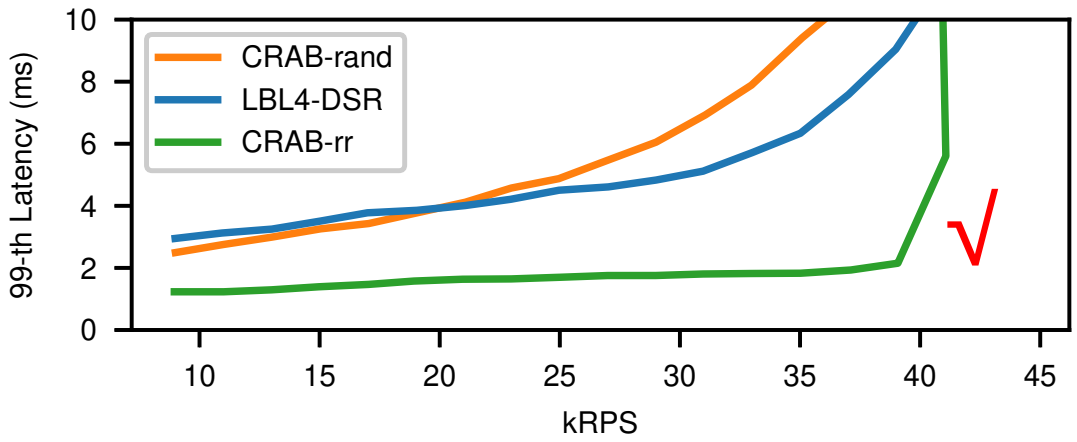


Figure 11: Load Balancing 48 single-core servers running a synthetic service time application with $\bar{S} = 1$ms

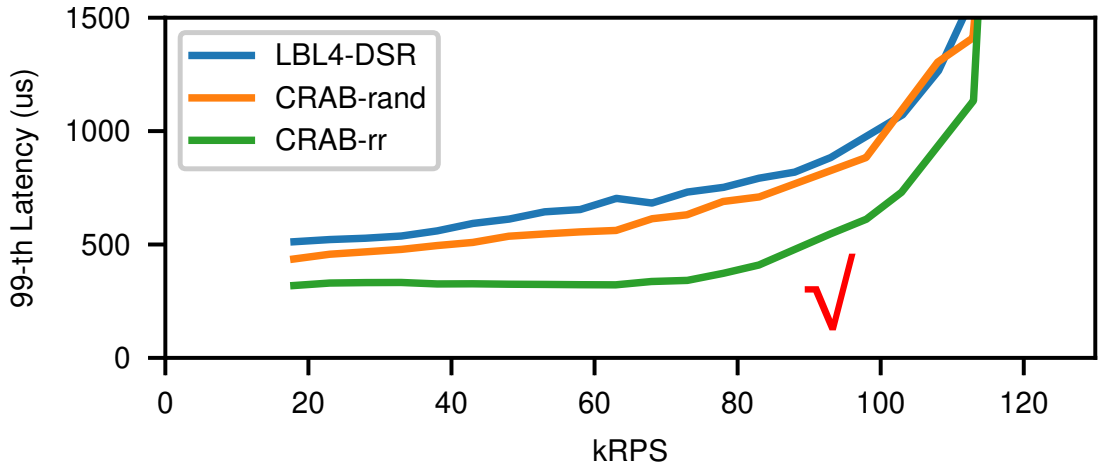
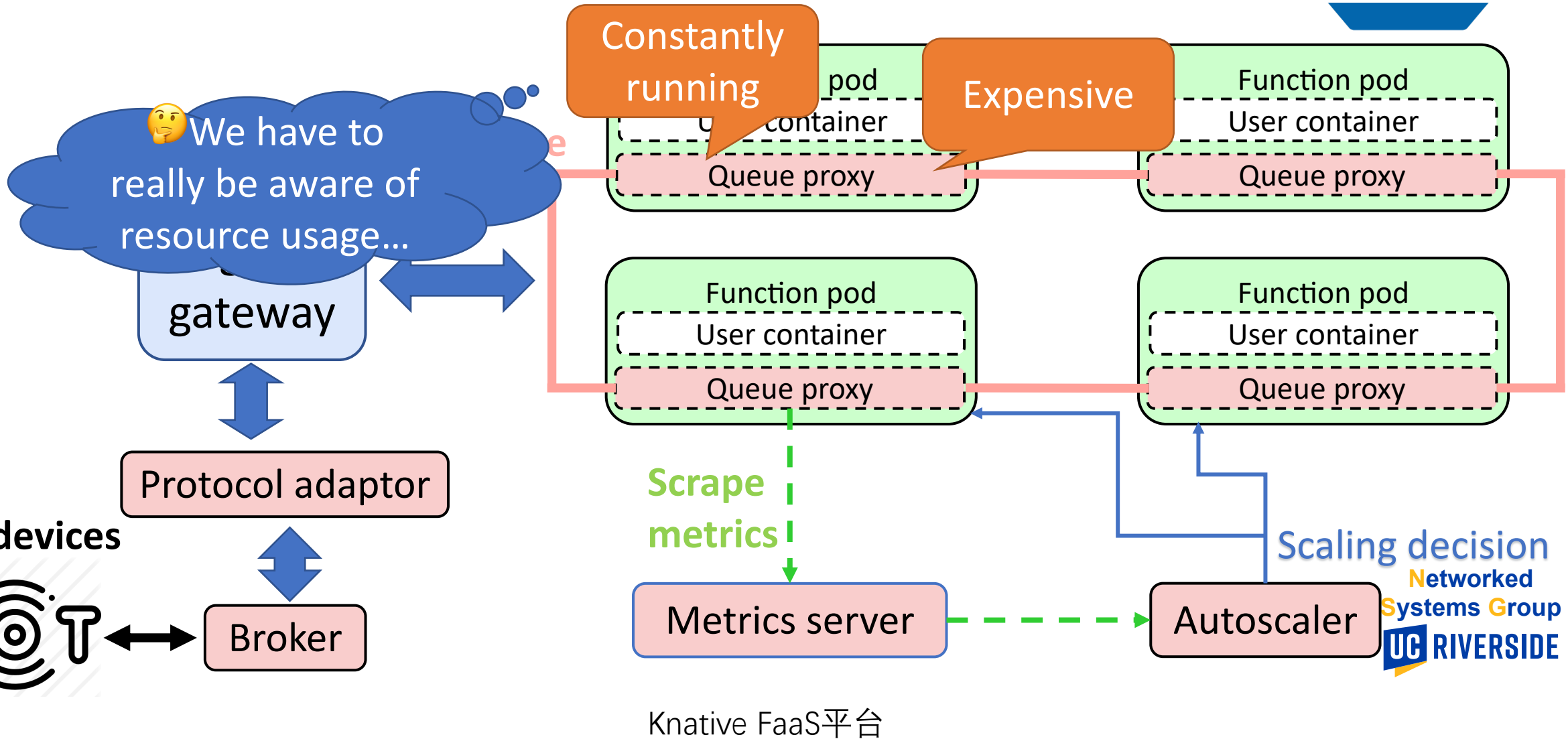


Figure 12: Load Balancing 48 NGINX servers serving an 8 kB static file.

03

FaaS数据面优化

FaaS数据面优化



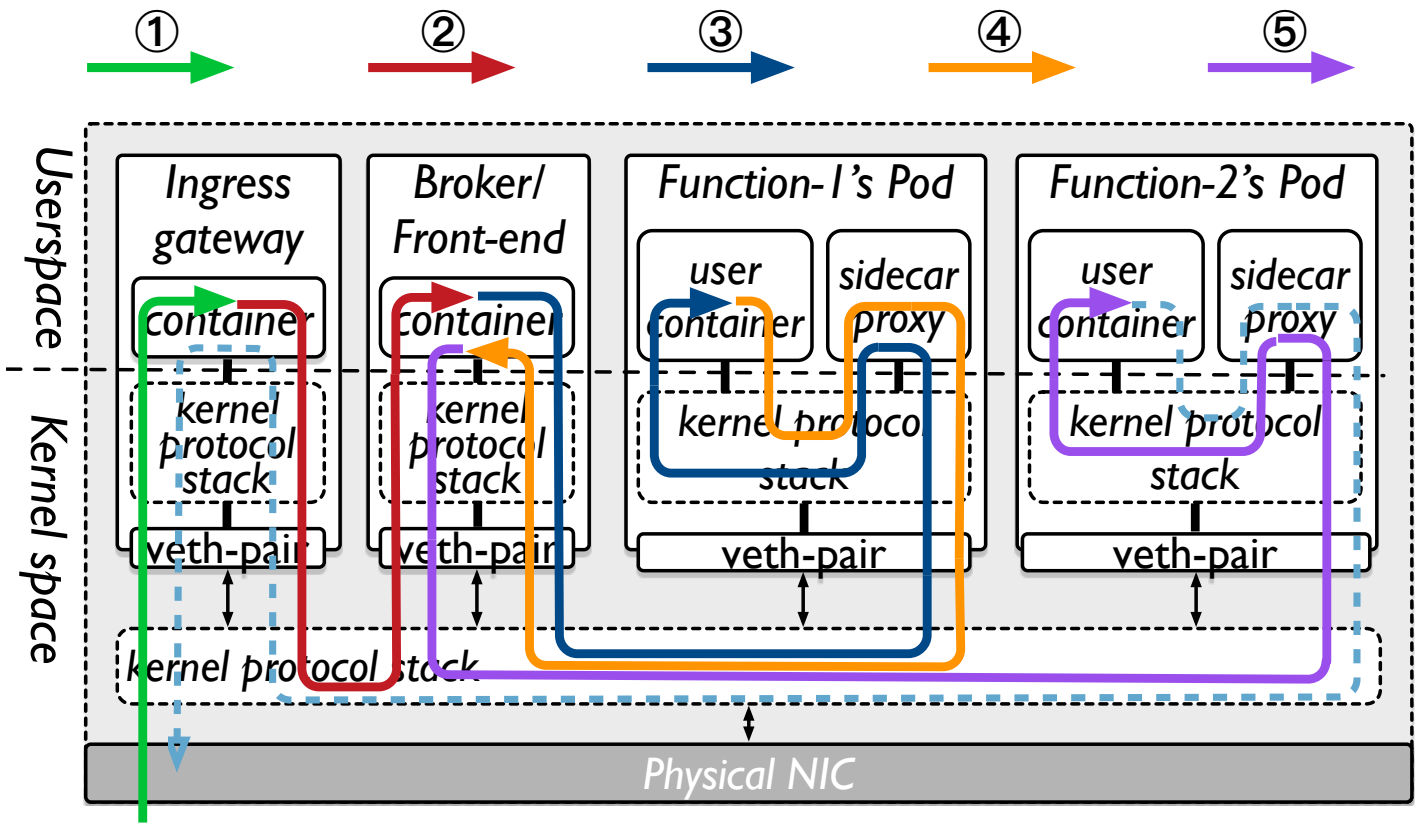
Knative FaaS平台

FaaS数据面优化

➢ 性能负荷

SPRIGHT: Extracing the Server from Serverless Compuing! High-Performance eBPF-based Event-driven, Shared-Memory Processing, Sigcomm 2022

- 典型的无服务器函数链处理中涉及的步骤：网络协议、复制、中断、上下文切换等;



Data Pipeline No.	External			Within chain				Total
	①	②	total	③	④	⑤	total	
# of copies	1	2	3	4	4	4	12	15
# of ctxt switches	1	2	3	4	4	4	12	15
# of irqs	3	4	7	6	6	6	18	25
# of proto. processing	1	2	3	3	3	3	9	12
# of serialization	1	1	2	2	2	2	6	8
# of deserialization	0	1	1	2	2	2	6	7

FaaS数据面优化

➢ 性能负荷

- 典型的无服务器功能链设置中涉及的处理：网络协议、复制、中断、上下文切换等;

Takeaway#1: Excessive data copies, context switches, and interrupts.

Takeaway#2: Excessive, duplicate protocol processing.

Takeaway#3: Unnecessary serialization/deserialization.

Takeaway#4: Individual, constantly-running heavyweight components.

Data Pipeline No.	External			Within chain				Total
	①	②	total	③	④	⑤	total	
# of copies	1	2	3	4	4	4	12	15
# of ctxt switches	1	2	3	4	4	4	12	15
# of irqs	3	4	7	6	6	6	18	25
# of proto. processing	1	2	3	3	3	3	9	12
# of serialization	1	1	2	2	2	2	6	8
# of deserialization	0	1	1	2	2	2	6	7

Networked Systems Grou

FaaS数据面优化

➢ 性能负荷

- Sidecar引起的额外负载

Takeaway#4: Individual, constantly-running heavyweight components.

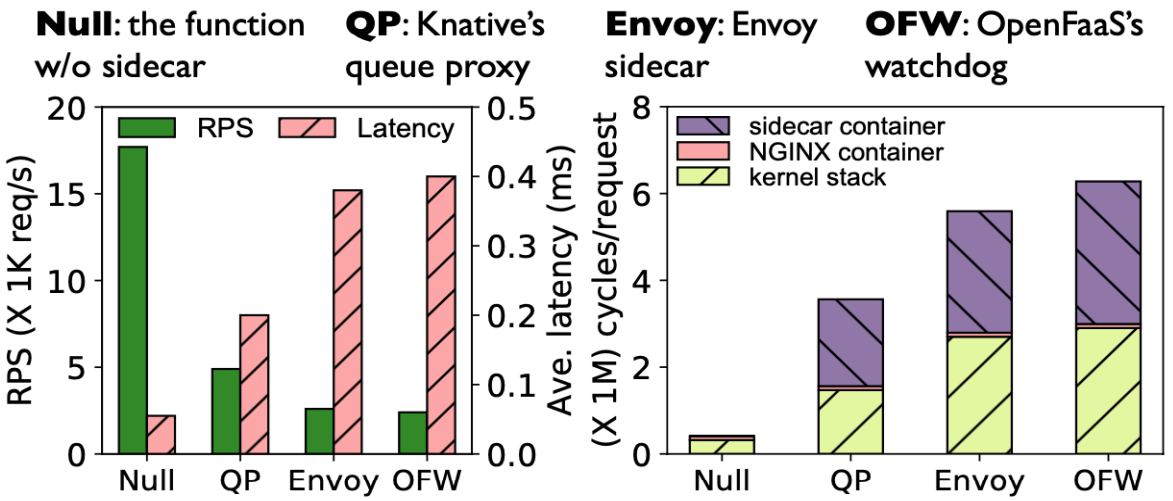


Figure 2: Performance and overhead breakdown of different sidecar proxy implementations.

😕 Having a sidecar proxy results in a 3×–7× reduction in throughput, 3×–7× higher latency, and a significant increase in CPU cycles per request.

😕 CPU overhead breakdown: 50% of CPU cycles are consumed by the kernel stack for the sidecar proxy.

FaaS数据面优化

➢ 系统实现

□ 系统的总体架构

- eBPF-based event-driven capability
- Shared memory processing

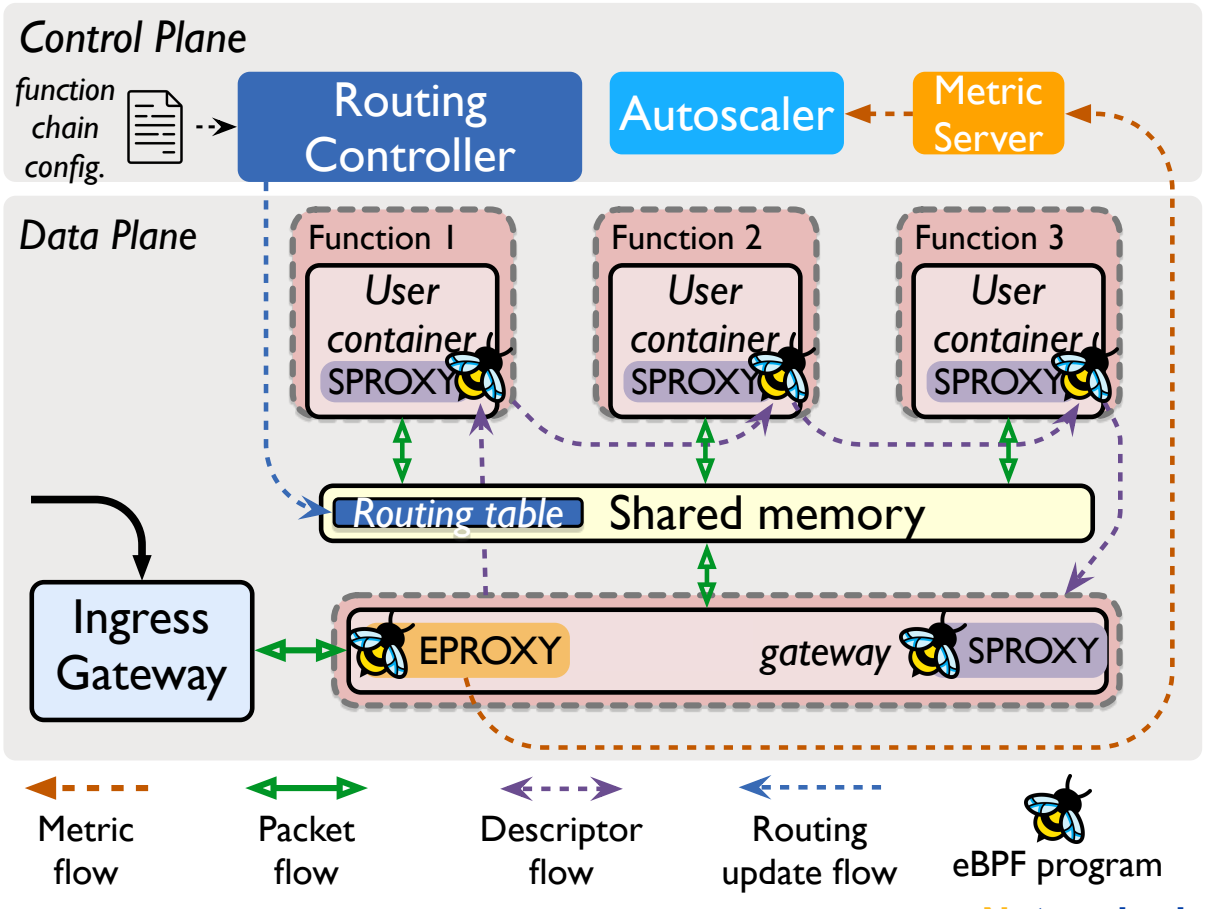
Optimization#1: Event-driven, shared memory function chain processing

Optimization#2: Direct Function Routing (DFR)

Optimization#3: Event-driven proxy

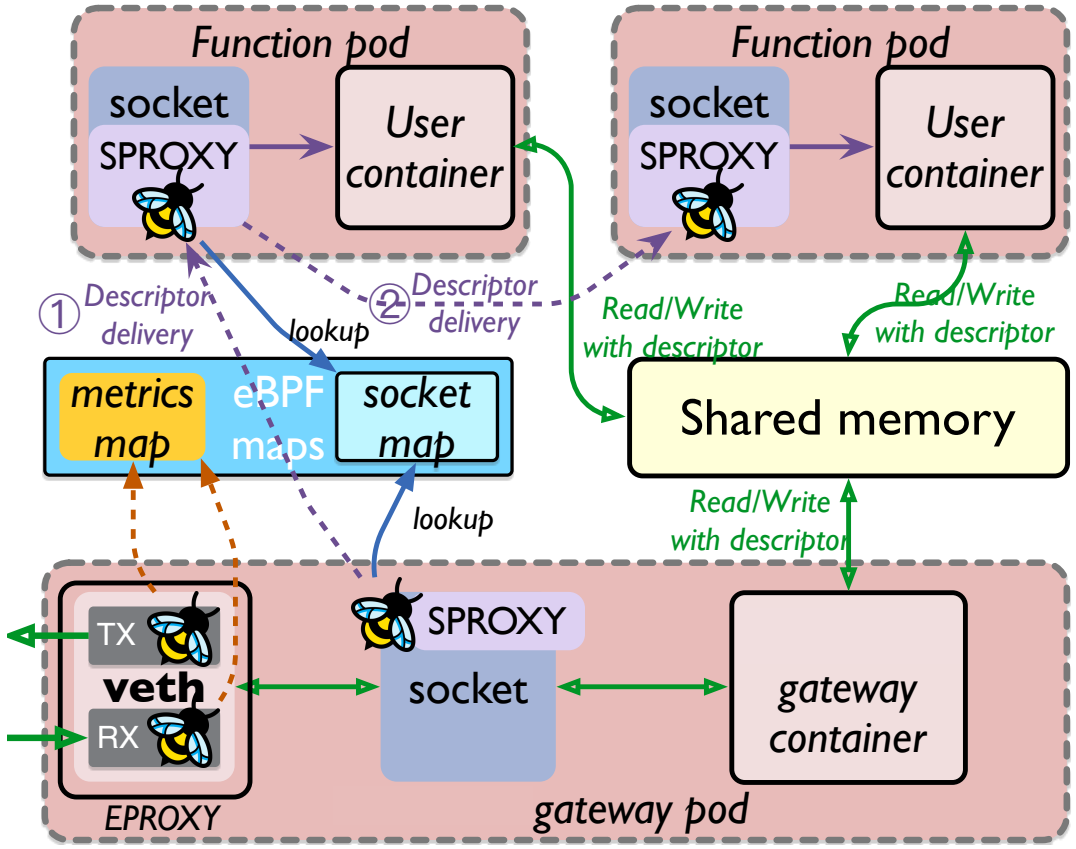
Optimization#4: eBPF-based dataplane acceleration for external communication

Optimization#5: Event-driven protocol adaptation (e.g., IoT)

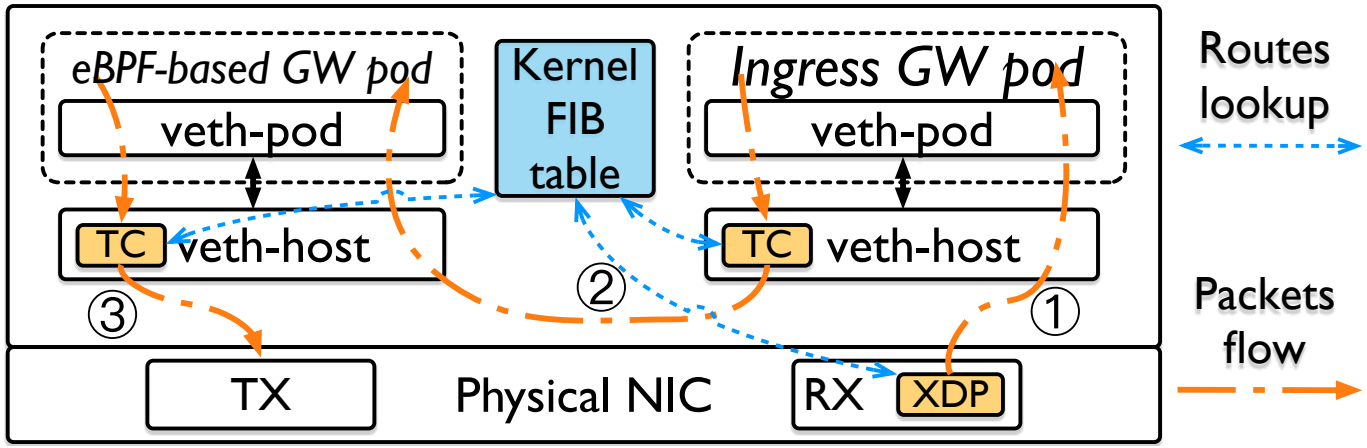


FaaS数据面优化

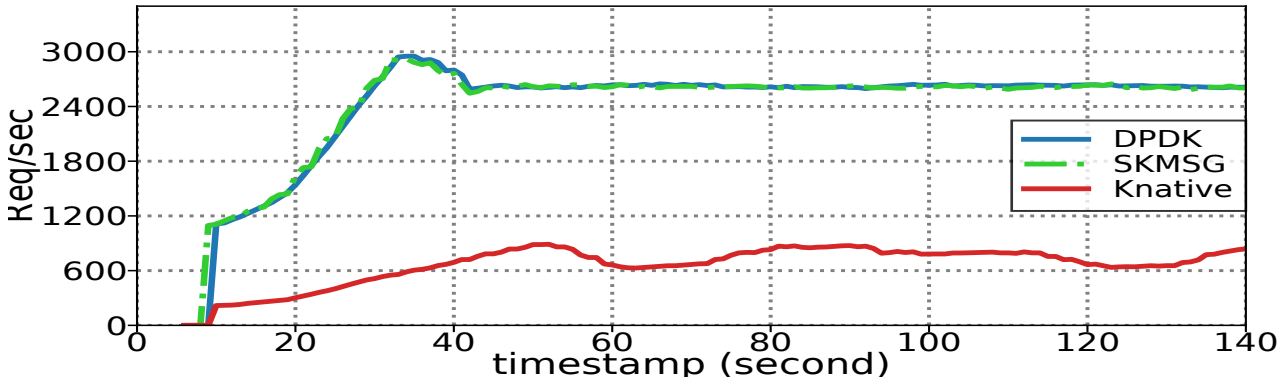
➢ 系统实现



基于eBPF的Proxy



基于eBPF的gateway



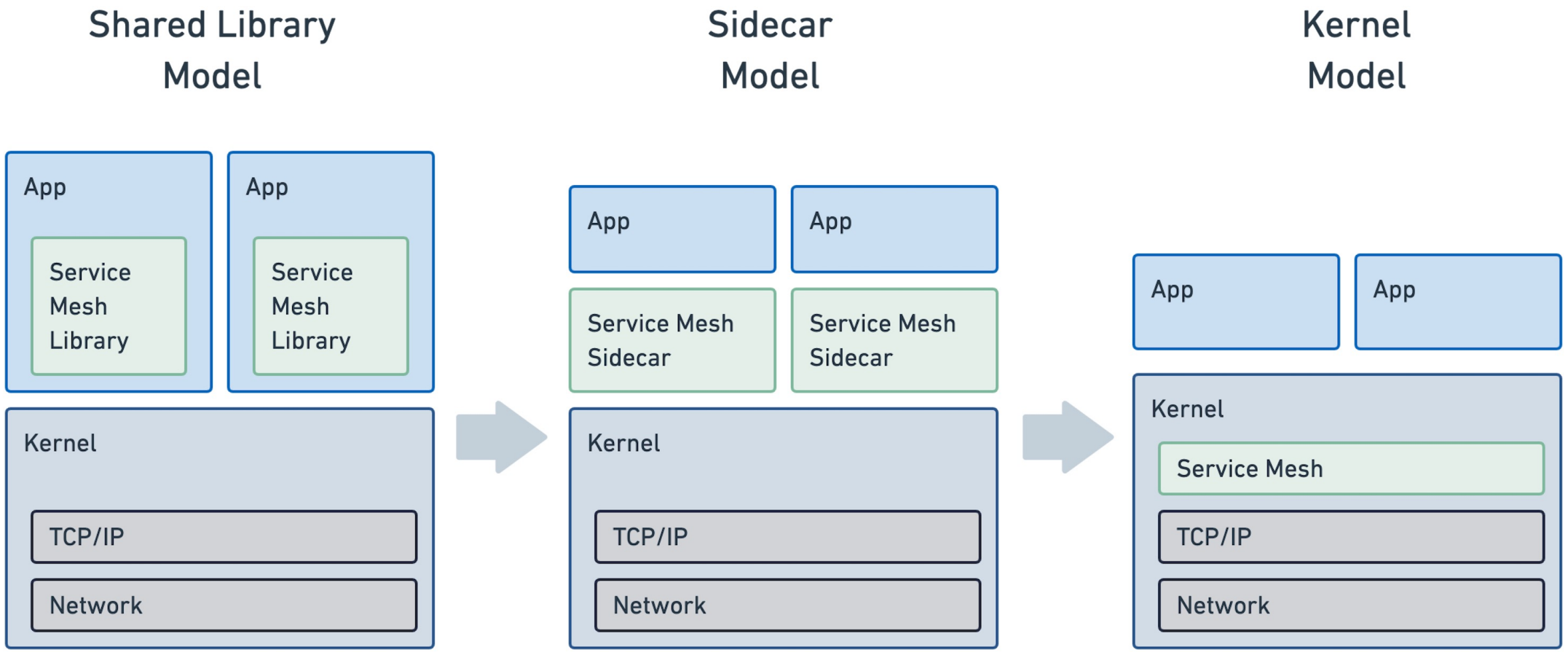
测试结果

01 展望

展望

➢ 服务网格数据面下沉到内核

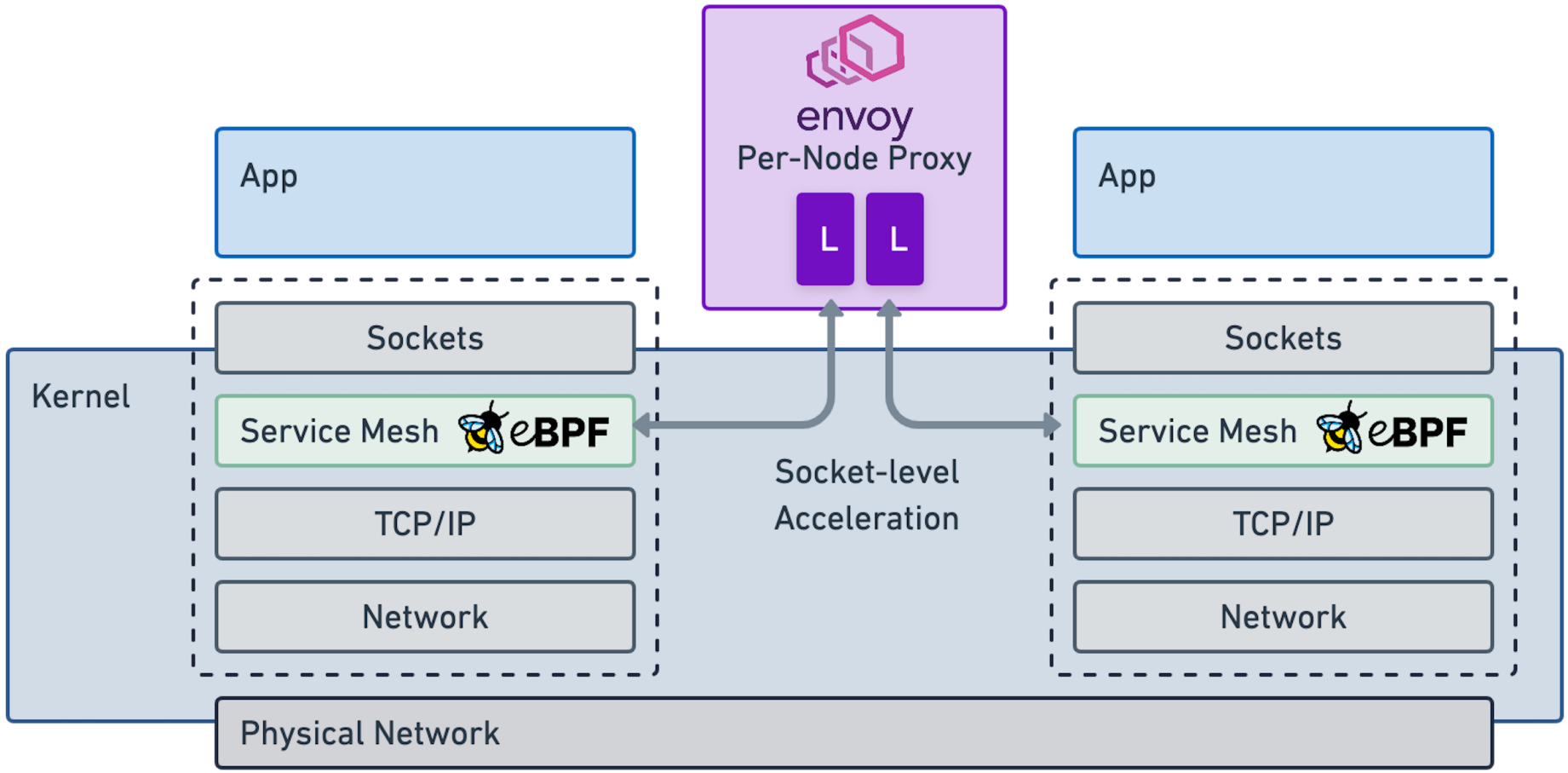
- 实现基于eBPF的Service Mesh数据面部分能力下沉，包括：请求转发、负载均衡、可观测性等



展望

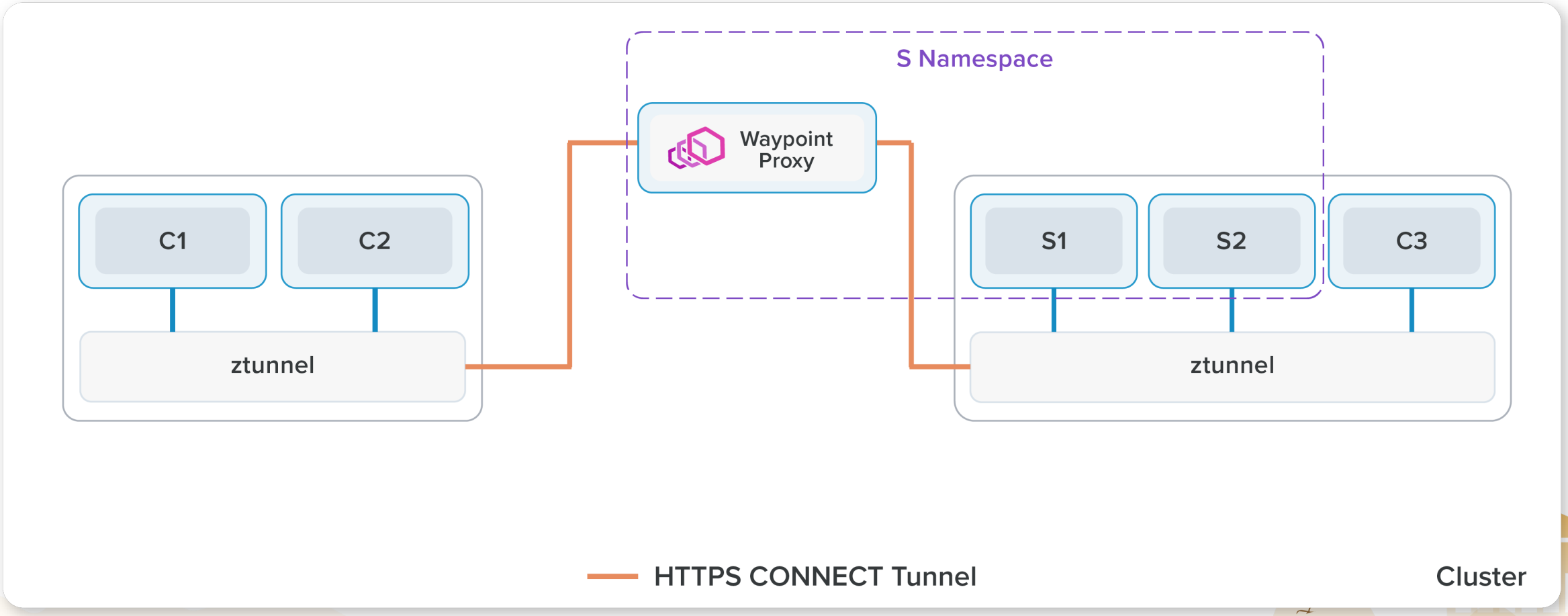
➢ 服务网格数据面下沉到内核

- eBPF + Proxy (Envoy) 实现丰富的服务治理如L7路由、灰度方法、故障注入等；



展望

- Istio ambient mesh 是 Istio 的一个无 sidecar 的数据平面，旨在降低基础设施成本和提高性能；



Thanks~!