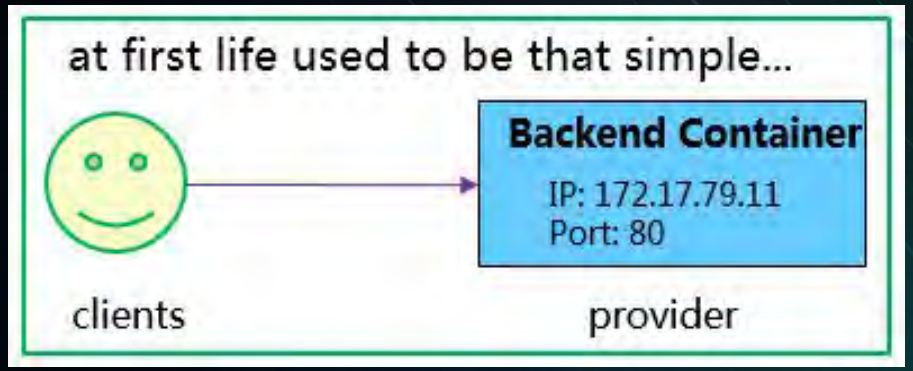
Kubernetes Service在大规模场景下的优化实战

杜军 (Github: @m1093782566)
华为云 高级工程师 - PaaS开源组
Kubernetes upstream, feature maintainer

目录

CONTENTS

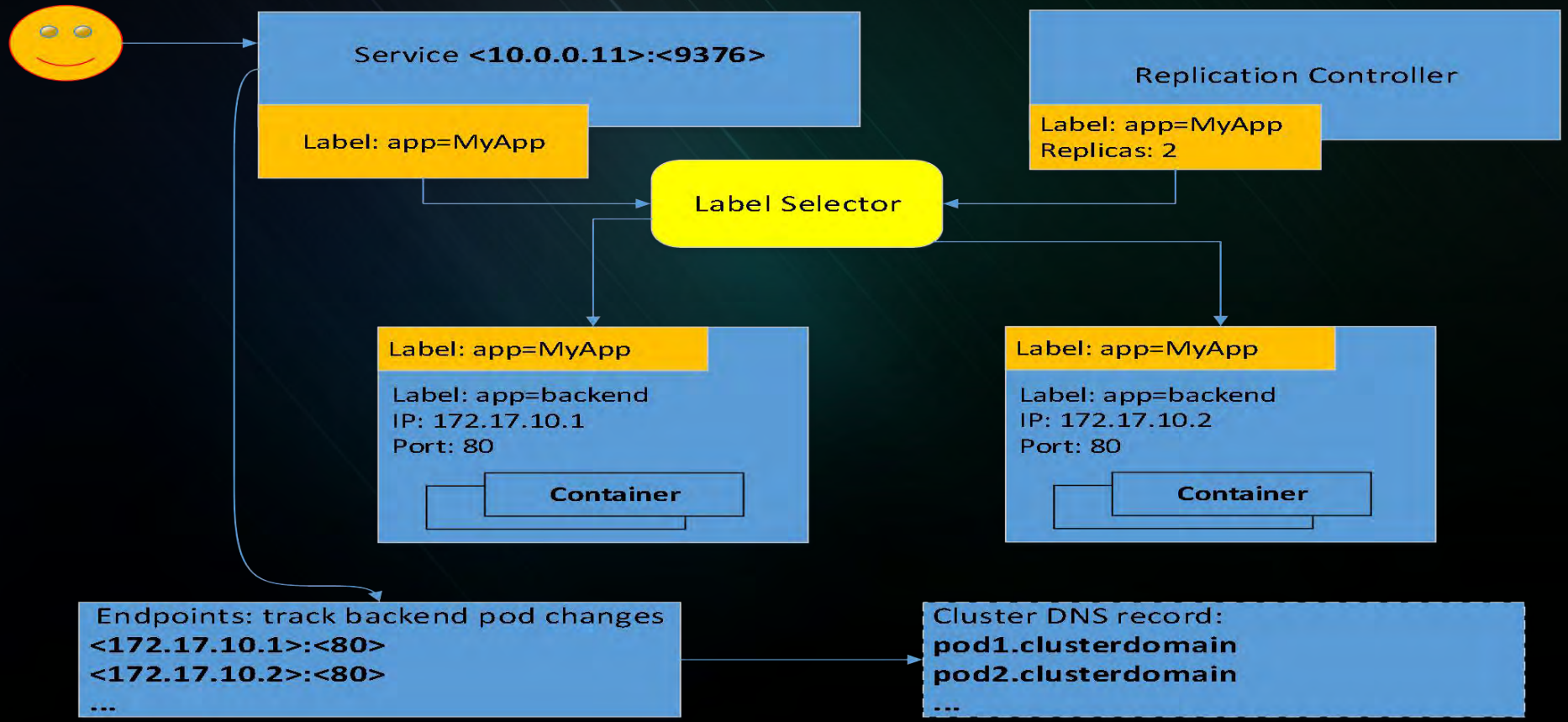
Kubernetes的Service



但，简单的生活总是暂时的：

- 多个后端实例，如何做到负载均衡？
- 如何保持会话亲和性？
- 容器迁移，IP发生变化如何访问？
- 健康检查怎么做？
- 怎么通过域名访问？

Kubernetes Service与Endpoints

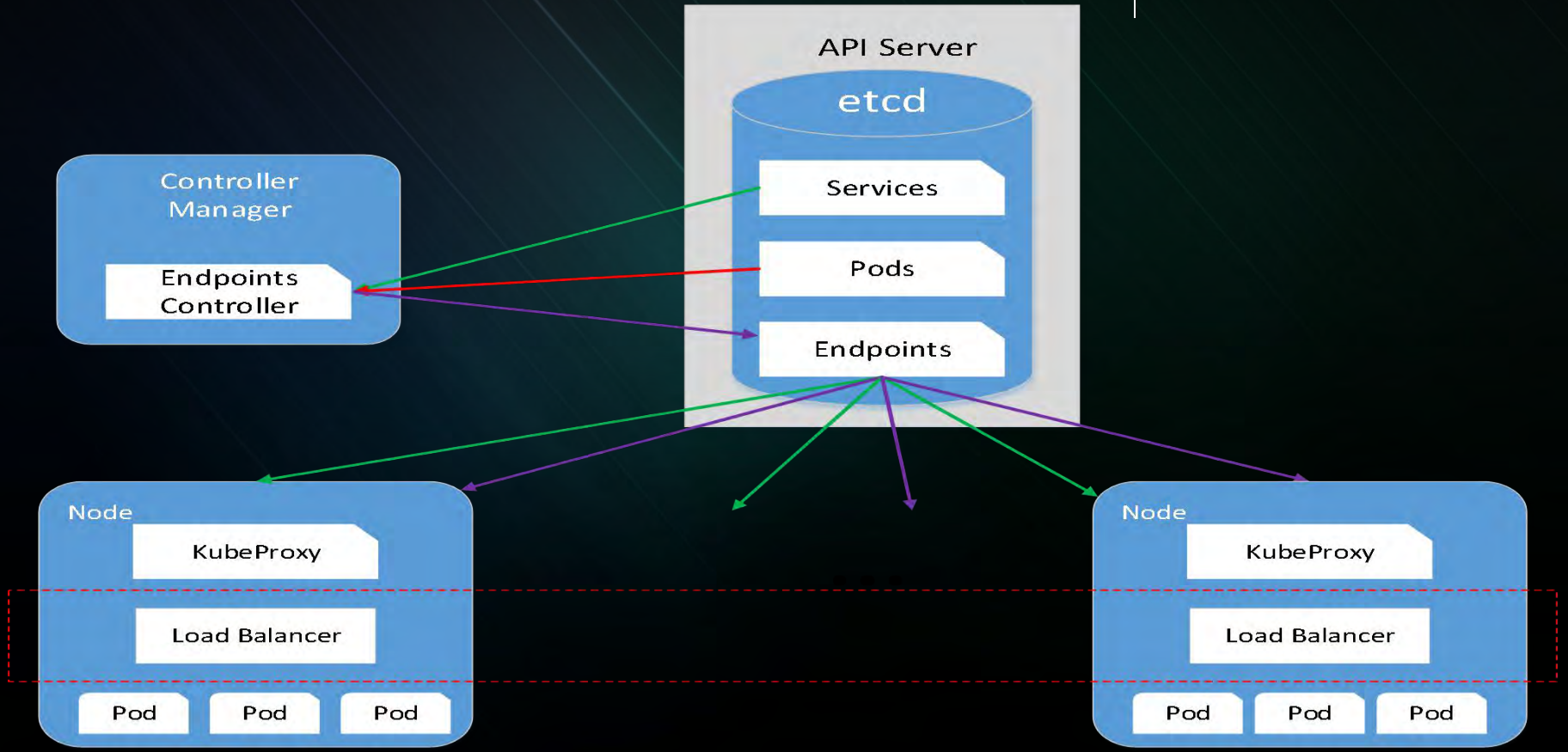


Service与Endpoints定义

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
  namespace: default
spec:
  clusterIP: 10.101.28.148
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 8080
  selector:
    app: nginx
```

```yaml
apiVersion: v1
kind: Endpoints
metadata:
  name: nginx-service
  namespace: default
subsets:
- addresses:
  - ip: 172.17.0.2
    nodeName: 100-106-179-237.node
    targetRef:
      kind: Pod
      name: nginx-rc-c8tw2
      namespace: default
  - ip: 172.17.0.3
    nodeName: 100-106-179-238.node
    targetRef:
      kind: Pod
      name: nginx-rc-x14tv
      namespace: default
  ports:
  - name: http
    port: 8080
    protocol: TCP
```

Service 内部逻辑

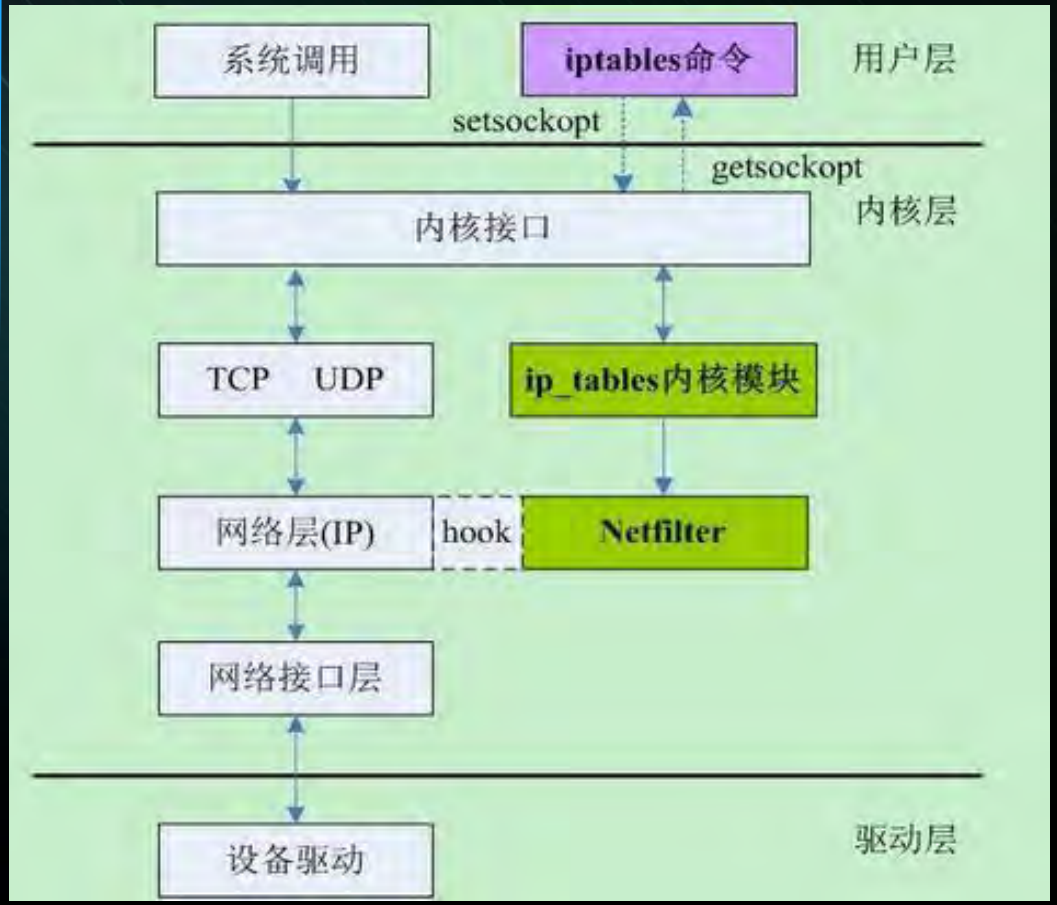


目录

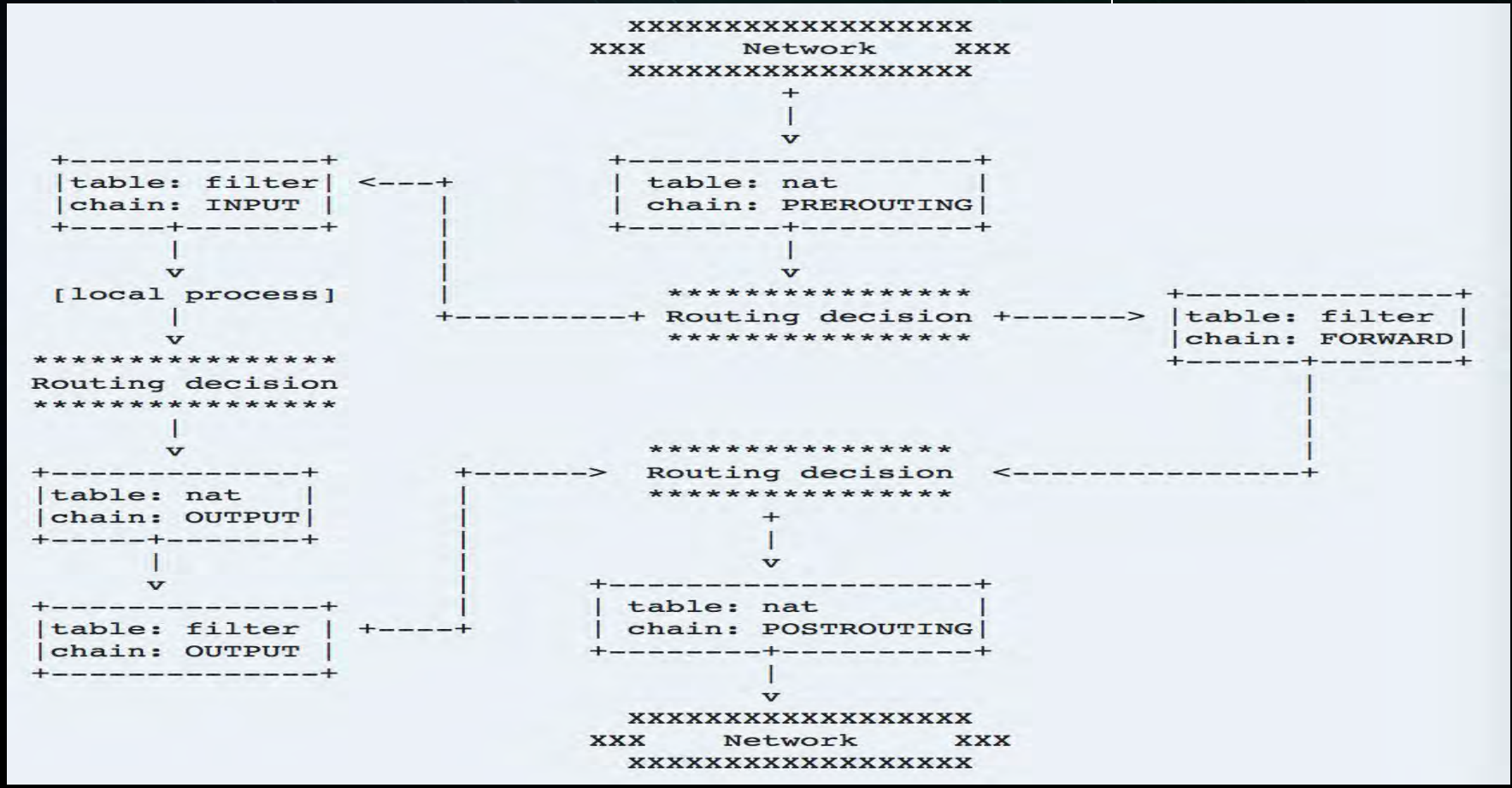
CONTENTS

Iptables是什么？

用户空间应用程序，通过配置Netfilter规则表（ Xtables ）来构建linux内核防火墙。



网络包通过Netfilter全过程



Iptables实现流量转发与负载均衡

Iptables如何做流量转发？

- **DNAT实现IP地址和端口映射**

iptables -t nat -A PREROUTING -d 1.2.3.4/32 --dport 80

-j DNAT --to-destination

10.20.30.40:8080

Iptables如何做负载均衡？

- **statistic模块为每个后端设置权重**

iptables -t nat -A

PREROUTING -d 1.2.3.4

--dport 80 **-m statistic**

--mode random

--probability .25 -j DNAT

--to-destination

10.20.30.40:8080

Iptables如何做会话保持？

- **recent模块设置会话保持时间**

iptables -t nat –A FOO

-m recent --rcheck --

seconds 3600 --reap --

name BAR -j BAR

Iptables在Kubernetes的应用举例

VIP:Port -> PREROUTING(OUTPUT) -> KUBE-SERVICES -> KUBE-SVC-XXX -> KUBE-SEP-XXX -> RIP:Port

```
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
KUBE-SERVICES  all  --  0.0.0.0/0            0.0.0.0/0

Chain KUBE-SERVICES (2 references)
target     prot opt source               destination
KUBE-SVC-6IM33IEVEEV7U3GP  tcp  --  0.0.0.0/0     10.20.30.40  tcp dpt:80

Chain KUBE-SVC-6IM33IEVEEV7U3GP (1 references)
target     prot opt source               destination
KUBE-SEP-Q3UCPZ54E6Q2R4UT  all  --  0.0.0.0/0            0.0.0.0/0

Chain KUBE-SEP-Q3UCPZ54E6Q2R4UT (1 references)
target     prot opt source               destination
DNAT       tcp  --  0.0.0.0/0            0.0.0.0/0       tcp to:172.17.0.2:8080
```

1

2

3

4

目录

CONTENTS

Iptables做负载均衡的问题

规则线性匹配时延

KUBE-SERVICES链挂了一长串KUBE-SVC-*链；访问每个service，要遍历每条链直到匹配，时间复杂度**O(N)**

规则更新时延

非增量式

可扩展性

当系统存在大量iptables规则链时，增加/删除规则会出现

kernel lock

Another app is currently holding the xtables lock. Perhaps you want to use the -w option?

可用性

后端实例扩容，服务会话保持时间更新等都会导致连接断开

Iptables规则匹配时延

	1	1000	10000	50000
First	575	614	1023	1821
Middle	575	602	1048	4174
Last	575	631	1050	7077

注：上面测试中，每个service在kube-services对应1条chain

更新Iptables规则的时延

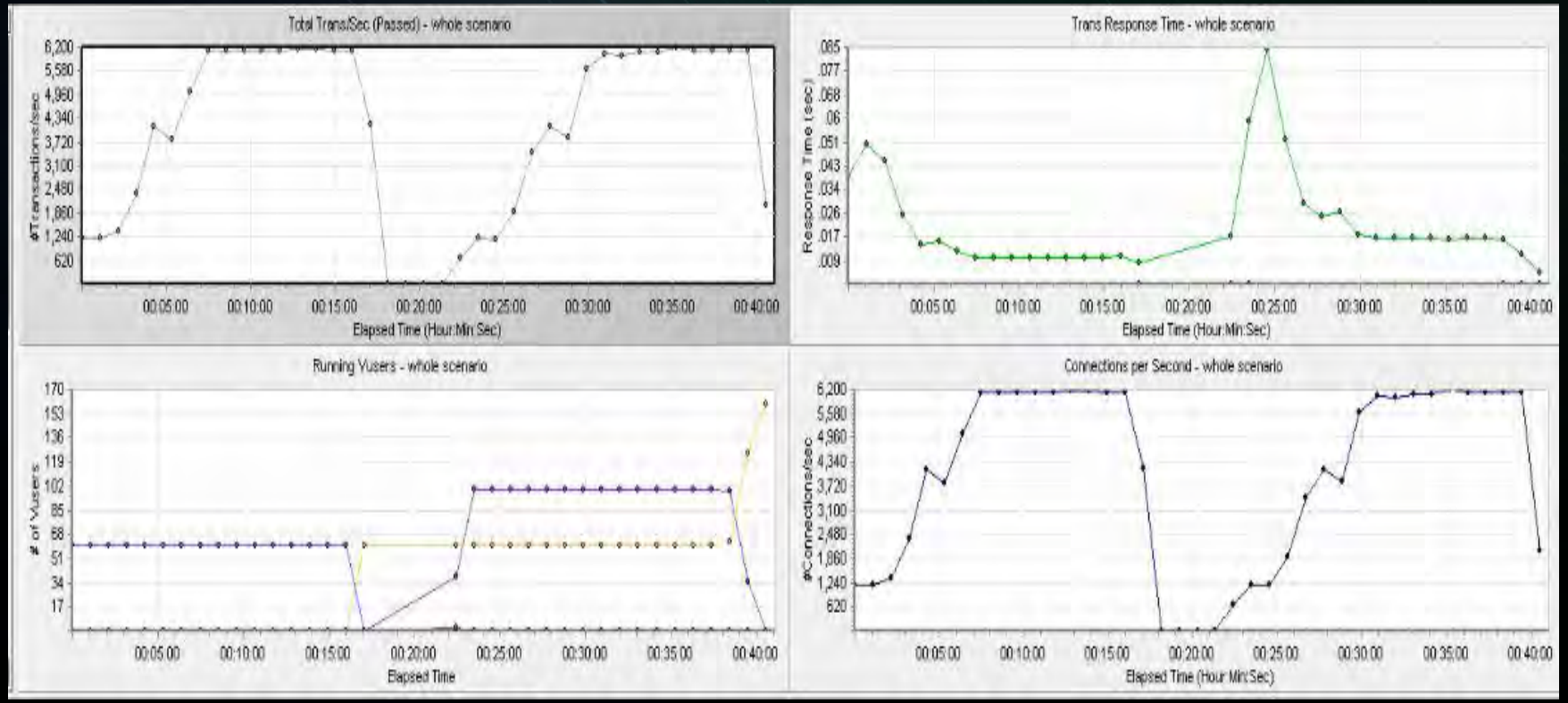
时延出现在哪？

- **非增量式，即使加上—no-flush(iptables-restore)选项**
- **Kube-proxy定期同步iptables状态：**
  - ✓ 拷贝所有规则 iptables-save
  - ✓ 在内存中更新规则
  - ✓ 在内核中修改规则 iptables-restore
  - ✓ 规则更新期间存在kernel lock

K service（40K 规则），增加一条iptables规则，耗时11min

20K service（160K 规则），增加一条iptables规则，耗时5h

Iptables周期性刷新导致TPS抖动



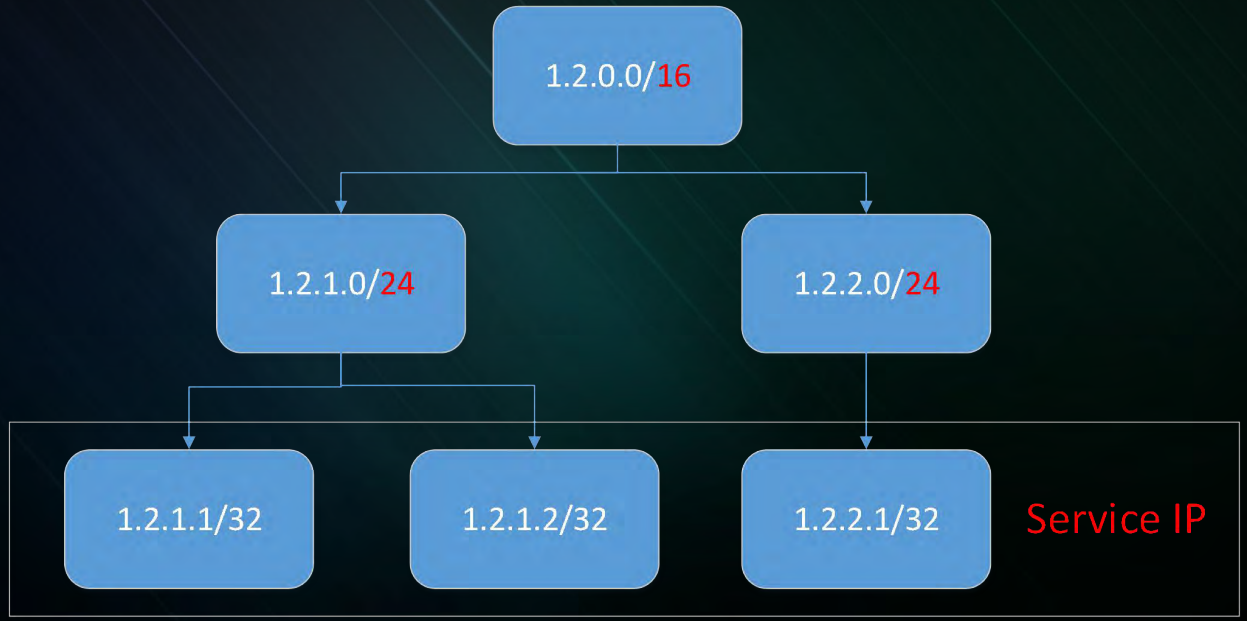
K8S Scalability

- 5000 Nodes
- 1000 Services ？？

优化方案

- 使用树形结构组织iptables规则
- IPVS

树形结构的iptables规则



路由时间复杂度取决于搜索树的高度(m)，时间复杂度O($\sqrt[m]{N}$)

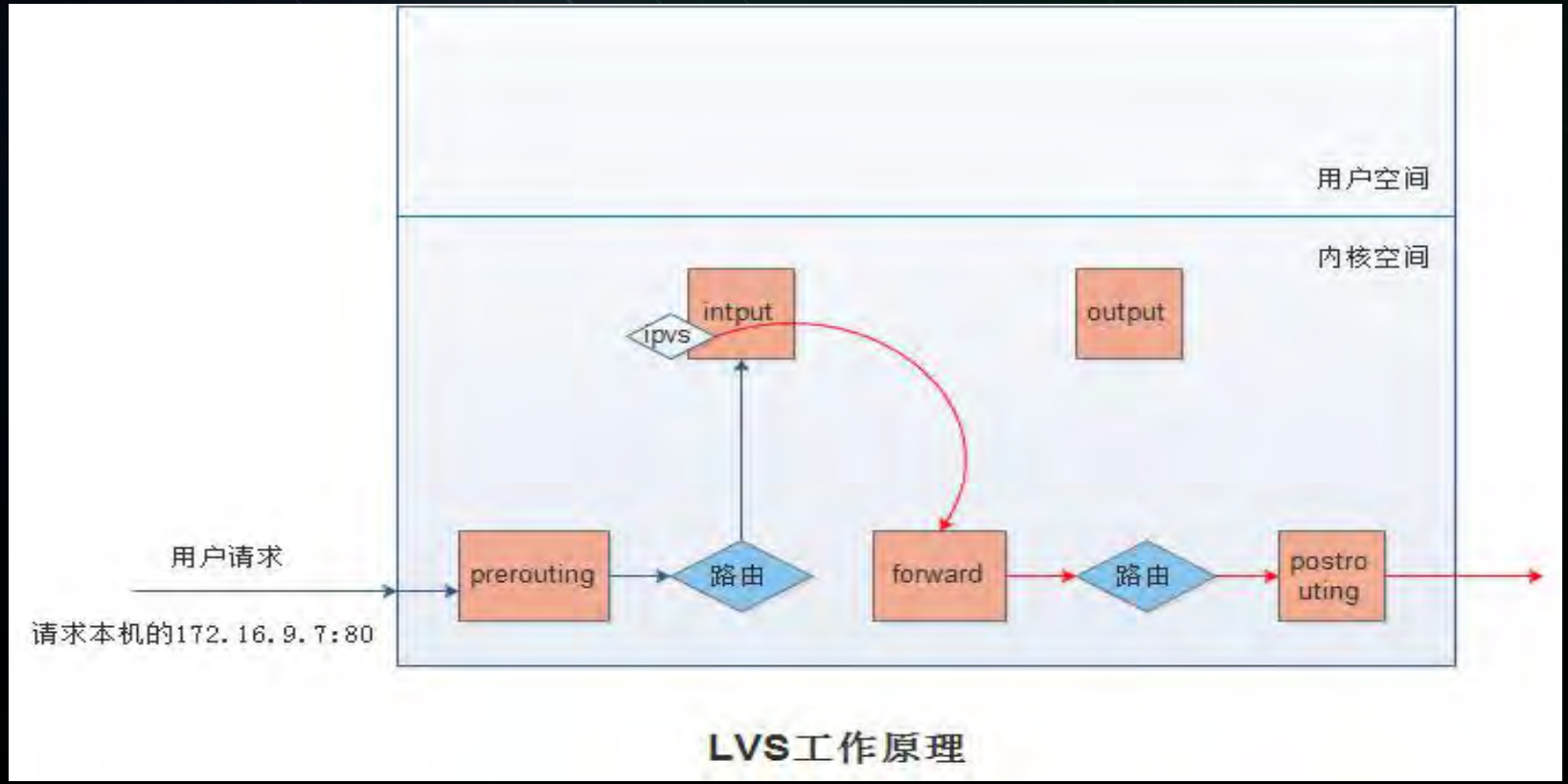
目录

CONTENTS

什么是IPVS（IP Virtual Server）

- Linux内核实现的L4 LB，LVS负载均衡的实现
- 基于netfilter, hash table
- 支持TCP, UDP，SCTP协议，IPV4，IPV6
- 支持多种负载均衡策略
  - rr, wrr, lc, wlc, sh, dh, lblc…
- 支持会话保持
  - persistent connection调度算法

IPVS工作流



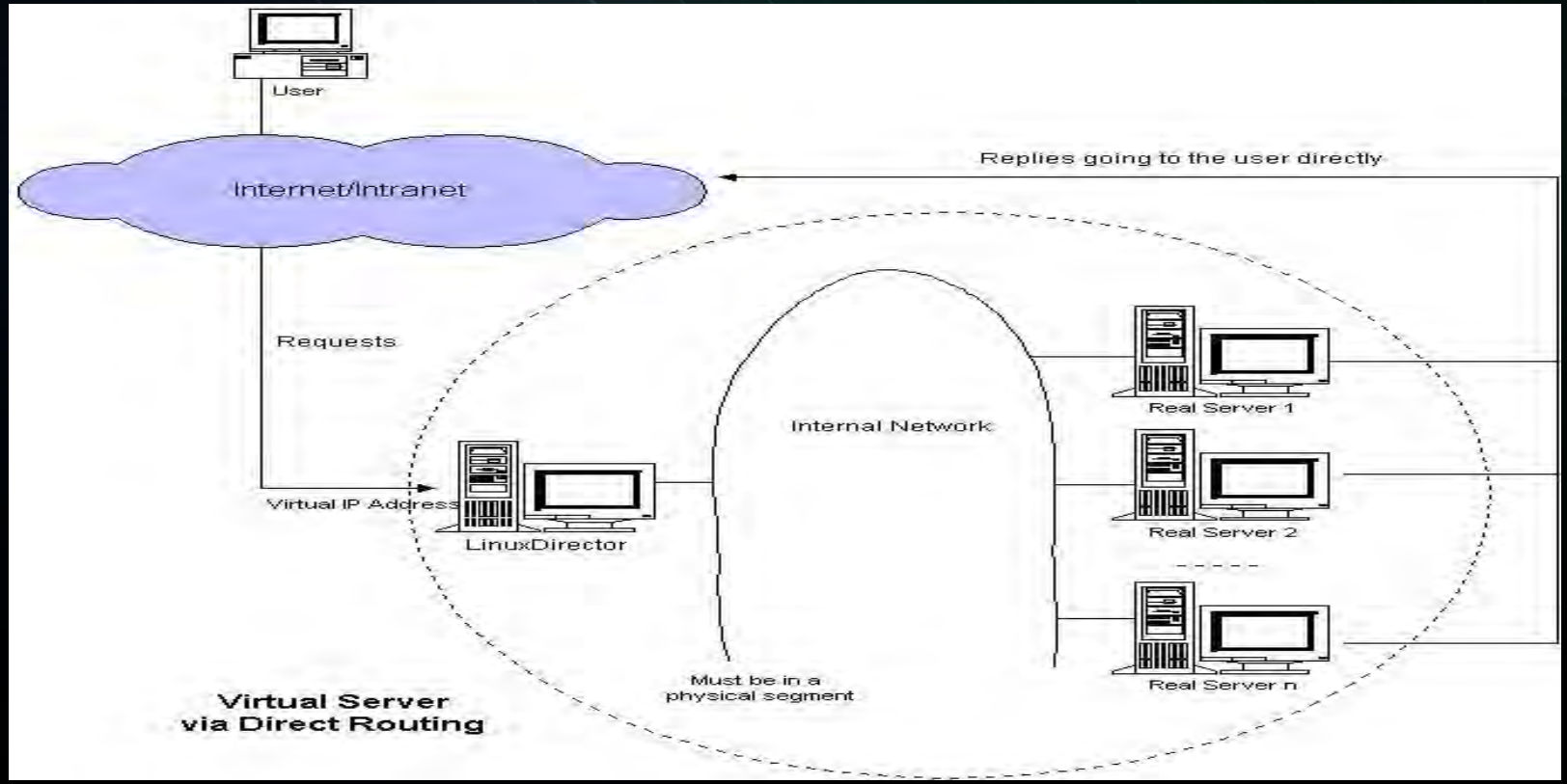
LVS工作原理

IPVS三种转发模式

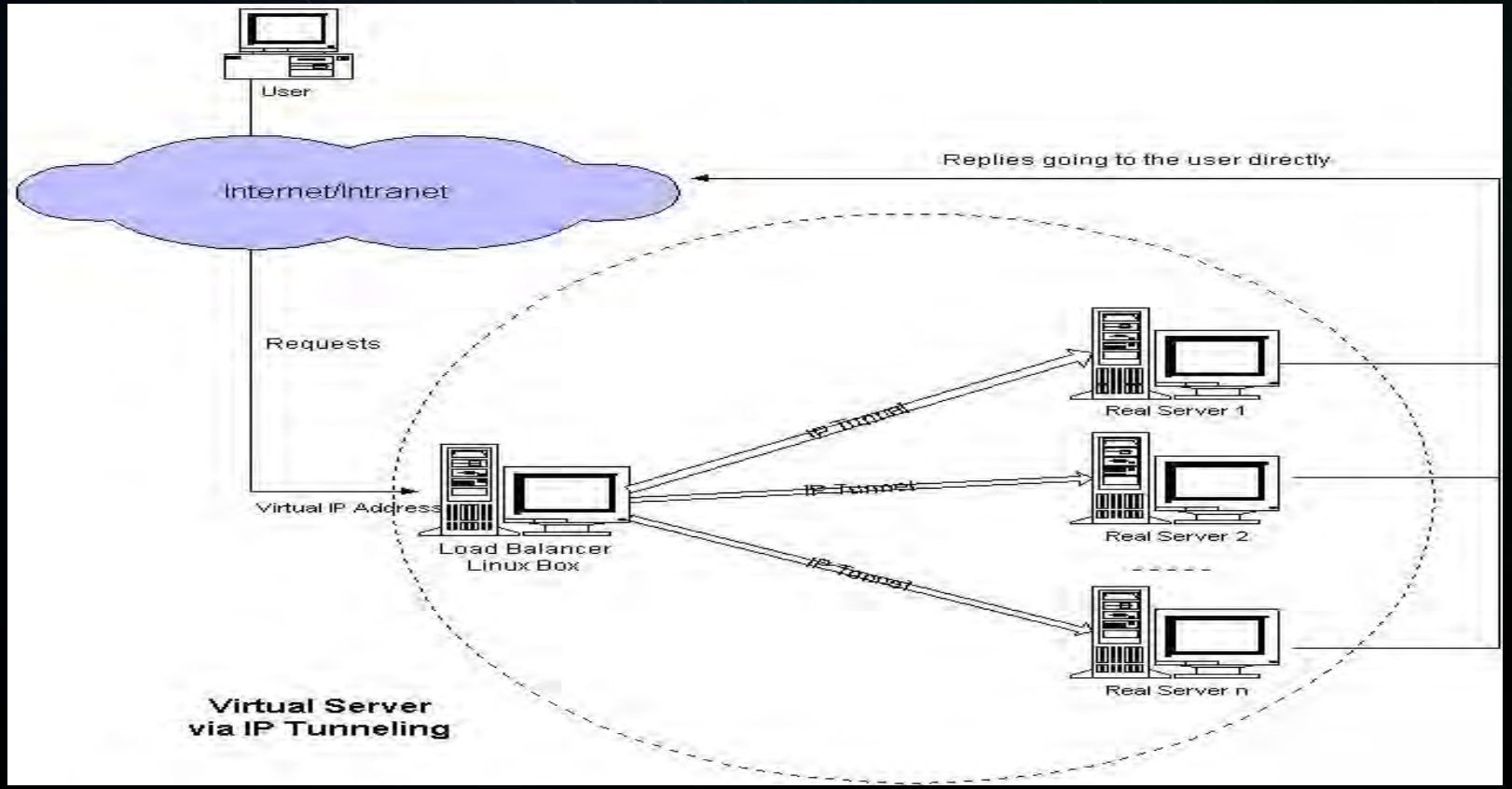
支持三种LB模式: Direct Routing(DR), Tunneling, NAT

- DR模式工作在L2，最快，但不支持端口映射
- Tunneling模式用IP包封装IP包，不支持端口映射
- DR和Tunneling模式，回程报文不会经过IPVS Director
- NAT模式支持端口映射，回程报文经过IPVS Director - 内核原生版本只做DNAT，不做SNAT

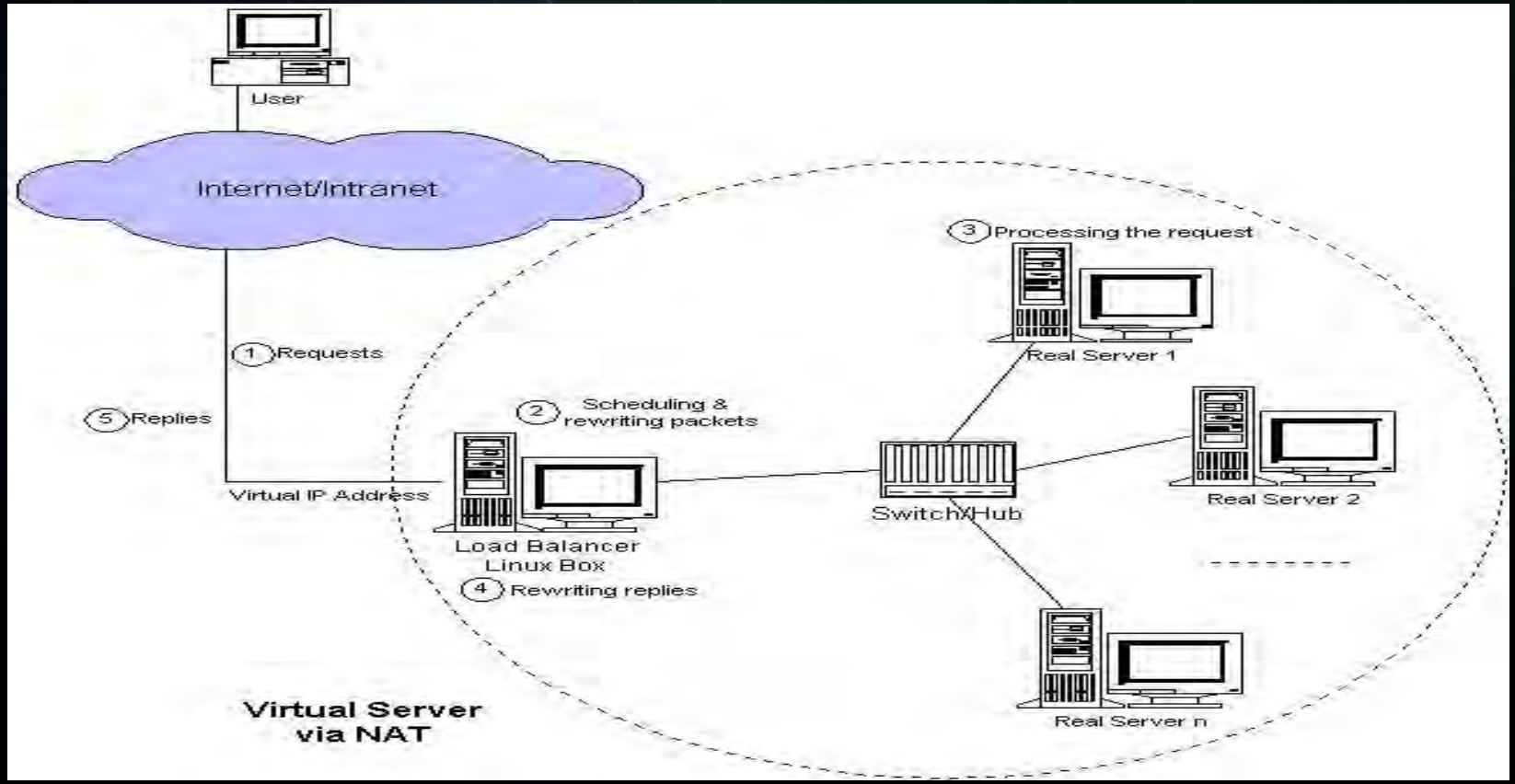
DR


User
Internet/Intranet
Replies going to the user directly
Requests
Virtual IP Address
LinuxDirector
Internal Network
Real Server 1
Real Server 2
Real Server n
Must be in a
physical segment
Virtual Server
via Direct Routing

Tunneling


User
Internet/Intranet
Replies going to the user directly
Requests
Virtual IP Address
Load Balancer
Linux Box
IP Tunnel
IP Tunnel
IP Tunnel
Real Server 1
Real Server 2
Real Server n
Virtual Server
via IP Tunneling

NAT


User
Internet/Intranet
1 Requests
5 Replies
Virtual IP Address
2 Scheduling & rewriting packets
Load Balancer Linux Box
4 Rewriting replies
3 Processing the request
Real Server 1
Switch/Hub
Real Server 2
Real Server n
Virtual Server via NAT

IPVS做流量转发

- **绑定VIP**
  - ✓ **dummy网卡**

```
# ip link add dev dummy0 type dummy
# ip addr add 192.168.2.2/32 dev dummy0
```

  - ✓ **本地路由表**

```
# ip route add to local 192.168.2.2/32 dev eth0 proto kernel
```

  - ✓ **网卡别名**

```
# ifconfig eth0:1 192.168.2.2 netmask 255.255.255.255 up
```

- **IPVS Virtual Server**

```
# ipvsadm -A -t 192.168.60.200:80 -s rr -p 600
```

- **IPVS Real Server**

```
# ipvsadm -a -t 192.168.60.200:80 -r 172.17.1.2:80 –m
# ipvsadm -a -t 192.168.60.200:80 -r 172.17.2.3:80 –m
```

IPVS实现Kubernetes Service


```
Name:              nginx-service
Type:              ClusterIP
IP:                10.102.128.4
Port:              http    80/TCP
Endpoints:         10.244.0.235:8080,10.244.1.237:8080
Session Affinity:   10800

# ip addr
73: kube-ipvs0: <BROADCAST,NOARP> mtu 1500 qdisc noop state DOWN qlen 1000
    link/ether 1a:ce:f5:5f:c1:4d brd ff:ff:ff:ff:ff:ff
    inet 10.102.128.4/32 scope global kube-ipvs0
       valid_lft forever preferred_lft forever

# ipvsadm -ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  10.102.128.4:80 rr persistent 10800
  -> 10.244.0.235:8080            Masq    1      0          0
  -> 10.244.1.237:8080            Masq    1      0          0
```

Kubernetes支持IPVS模式

社区1.8 Alpha特性, Owner @m1093782566

社区1.9进beta, Owner @m1093782566

支持ClusterIP，NodePort，External IP，Load Balancer...类型Service – iptables模式的特性，IPVS模式都支持！

兼容Network Policy

依赖iptables做SNAT和访问控制

目录

CONTENTS

Iptables vs. IPVS 增加规则时延

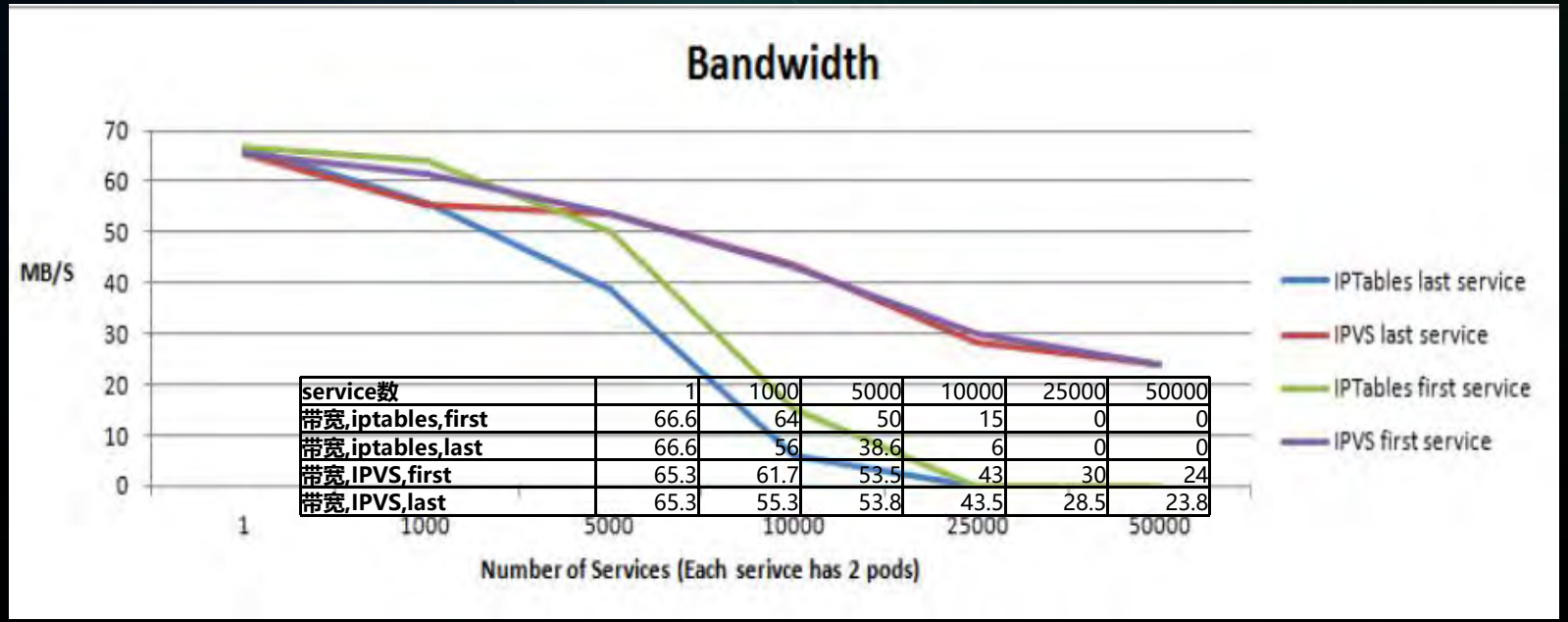
Service基数	1	5000	20000
Rules基数	8	40000	160000
增加1条Iptables规则	50 us	11 min	5 hours
增加1条IPVS规则	30 us	50 us	70 us

增加一条Iptables的时延，随着规则数的增加“指数”上升

增加一条IPVS的时延，规则基数对其几乎没影响

Iptables vs. IPVS 网络带宽

- 使用iperf测量
- 每个Service暴露4个端口（KUBE-SERVICES下挂4条KUBE-SVC-*）



service数	1	1000	5000	10000	25000	50000
带宽,iptables,first	66.6	64	50	15	0	0
带宽,iptables,last	66.6	56	38.6	6	0	0
带宽,IPVS,first	65.3	61.7	53.5	43	30	24
带宽,IPVS,last	65.3	55.3	53.8	43.5	28.5	23.8

Iptables vs. IPVS CPU/内存消耗

Metrics	number of service	IPVS	Iptables
Memory Usage	1000	386 MB	1.1 G
	5000	N/A	1.9 G
	10000	542 MB	2.3 G
	15000	N/A	OOM
	50000	1272 MB	OOM
CPU Usage	1000	0%	N/A
	5000		50% - 85%
	10000		50%-100%
	15000		N/A
	50000		N/A

Iptables vs. IPVS

Iptables

✓灵活，功能强大

✓在prerouting, postrouting, forward, input, output不同阶段都能对包进行操作

IPVS

✓更好的性能(hash vs. chain)

✓更多的负载均衡算法

- rr, wrr, lc, wlc, ip hash...

✓连接保持

- IPVS service更新期间，保持连接不断开

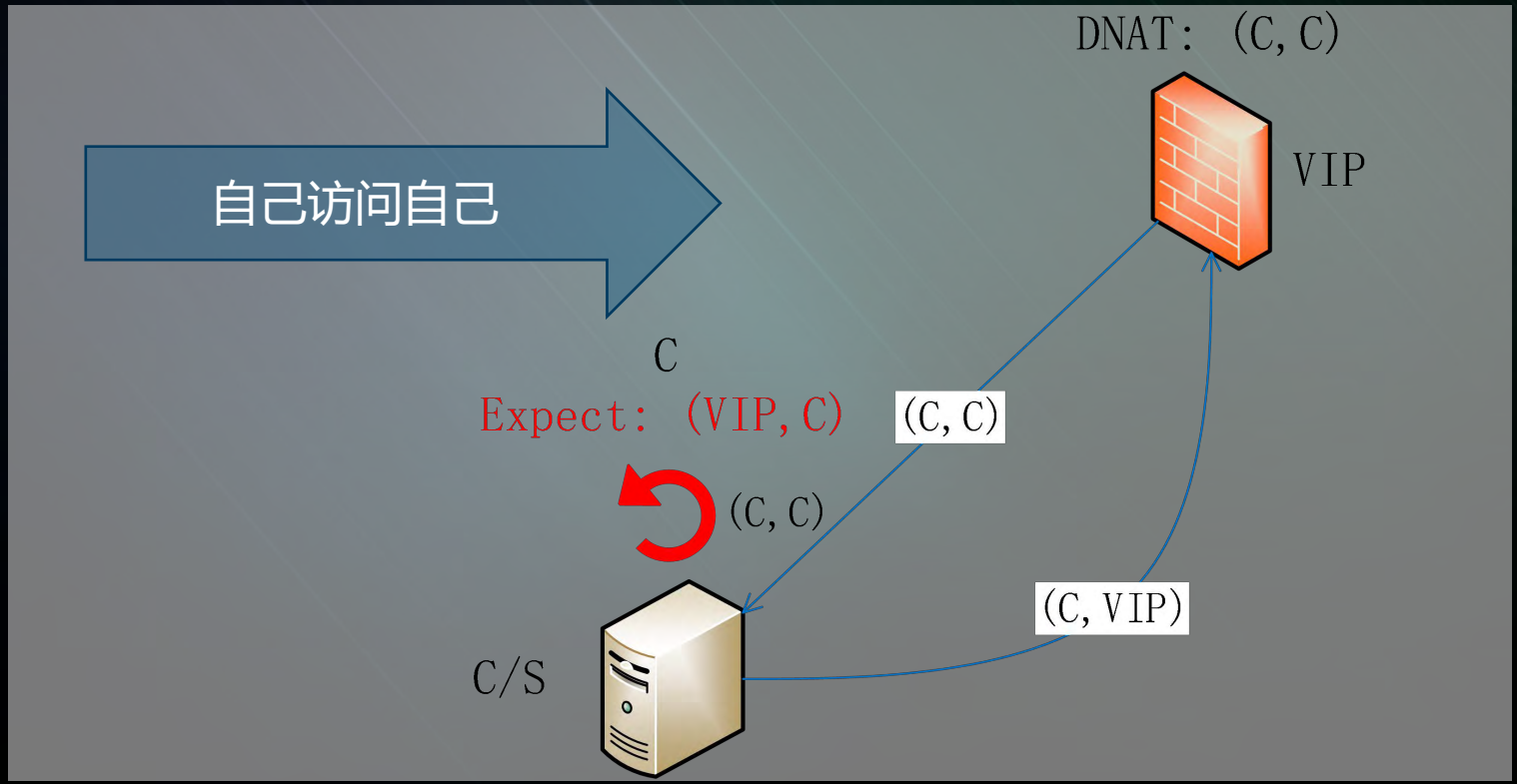
✓预先加载内核模

- nf_conntrack_ipv4, ip_vs, ip_vs_rr, ip_vs_wrr, ipvs_sh...

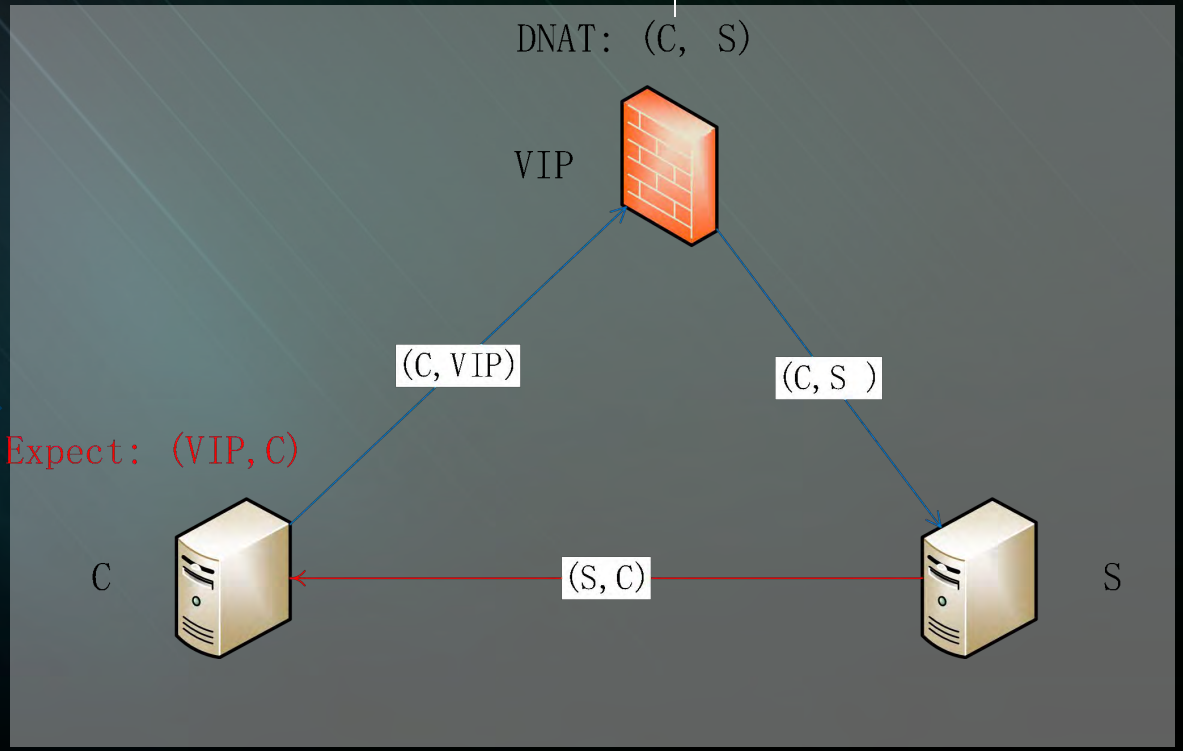
✓# echo 1 > /proc/sys/net/ipv4/vs/conntrack

✓**不支持127.0.0.1作为VIP**

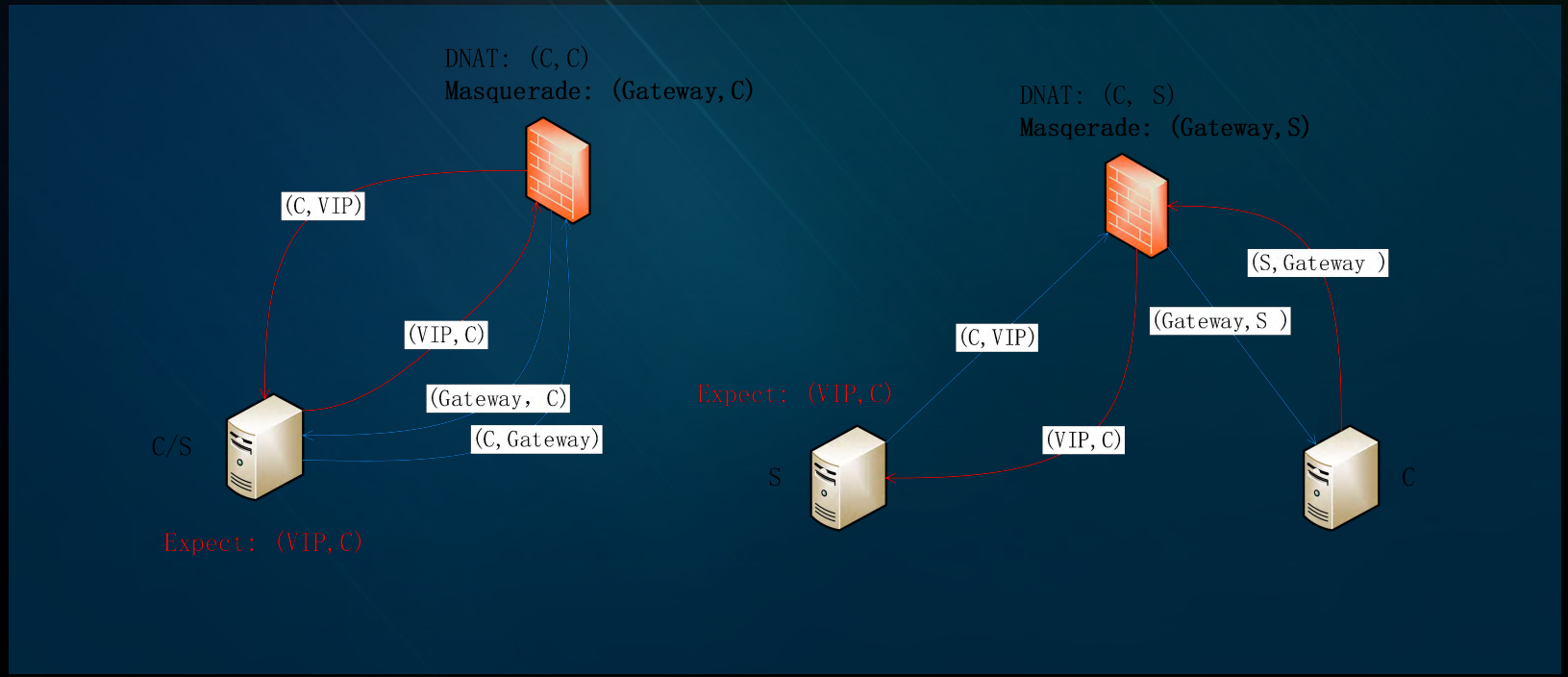
Why we still need iptables?



从Host发起跨节点访问



Iptables: Masquerade



但，不想要太多iptables...

目录

CONTENTS

IPSet - 把O(N)的iptables规则降为O(1)

pset create KUBE-LOOP-BACK hash:ip,port,ip
ipset add KUBE-LOOP-BACK **192.168.1.1**,udp:53,192.168.1.1
ipset add KUBE-LOOP-BACK **192.168.1.2**,tcp:80,192.168.1.2 **O(N)**

iptables -t nat -A POSTROUTING -m set --match-set KUBE-LOOP-BACK dst,dst,src -j MASQUERADEOUTING **O(1)**

而且：

ipset支持“增量”式增/删/改，而非iptables式全量更新

欢迎关注，华为云！

感谢聆听

Thanks !