Making the Kubernetes Service Abstraction Scale using BPF

Daniel Borkmann, Martynas Pumputis
Linux Plumbers Conference, 2019

Problem statement

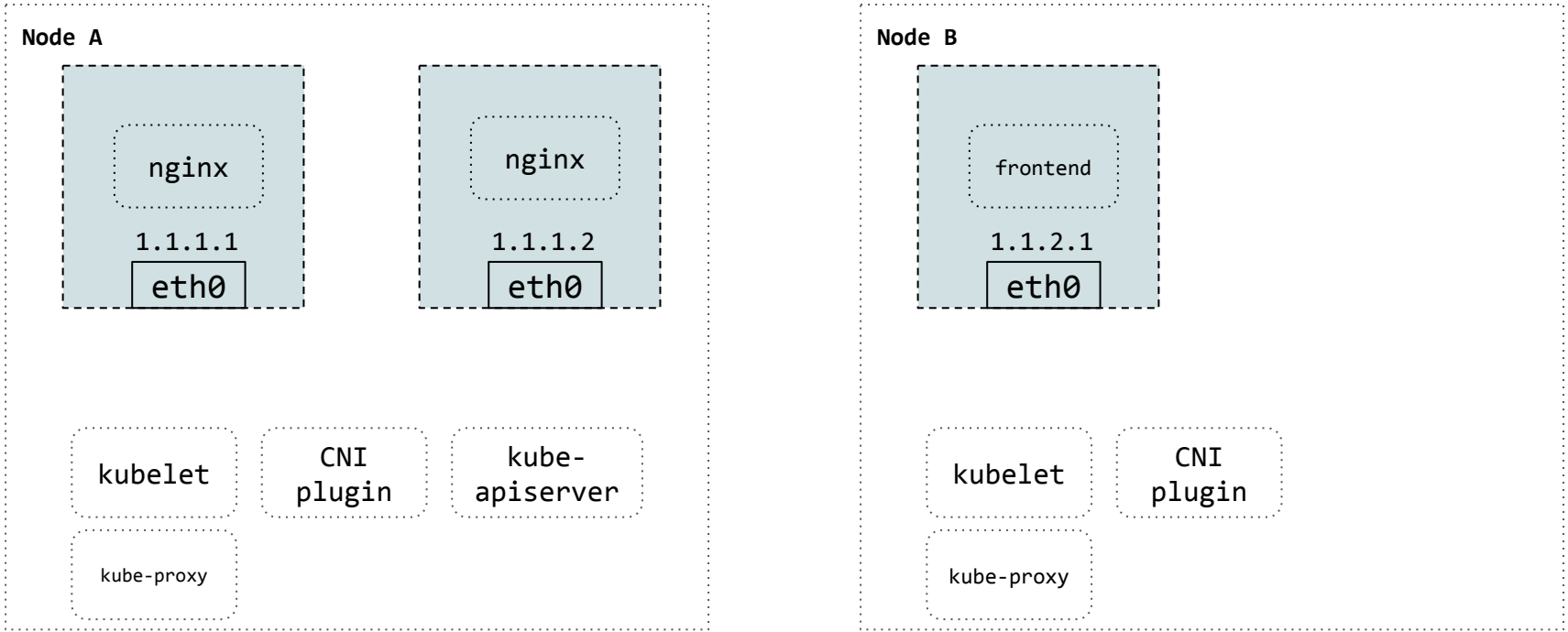
Kubernetes relies (a lot) on iptables:

- Low / unpredictable packet latency
- Slow update time
- Reliability issues

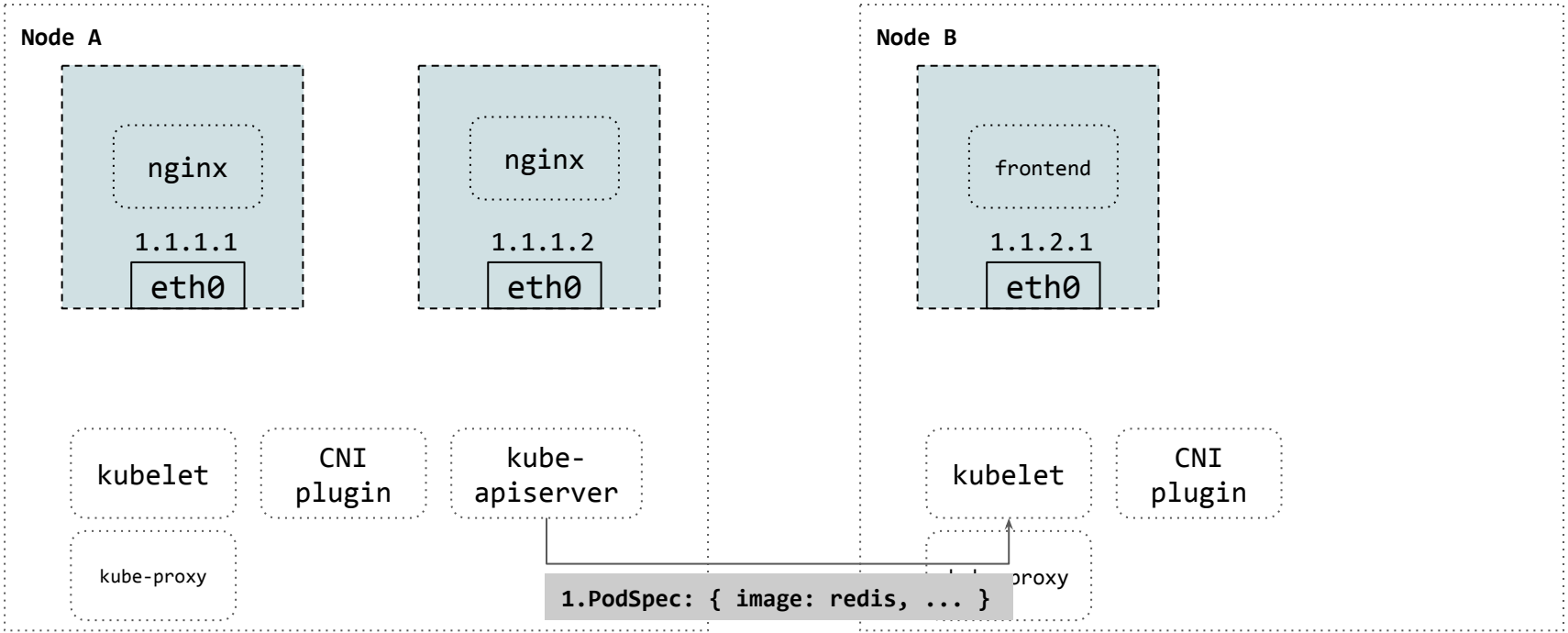
https://github.com/kubernetes/community/blob/master/sig-scalability/blogs/k8s-services-scalability-issues.md

Kubernetes (k8s) networking

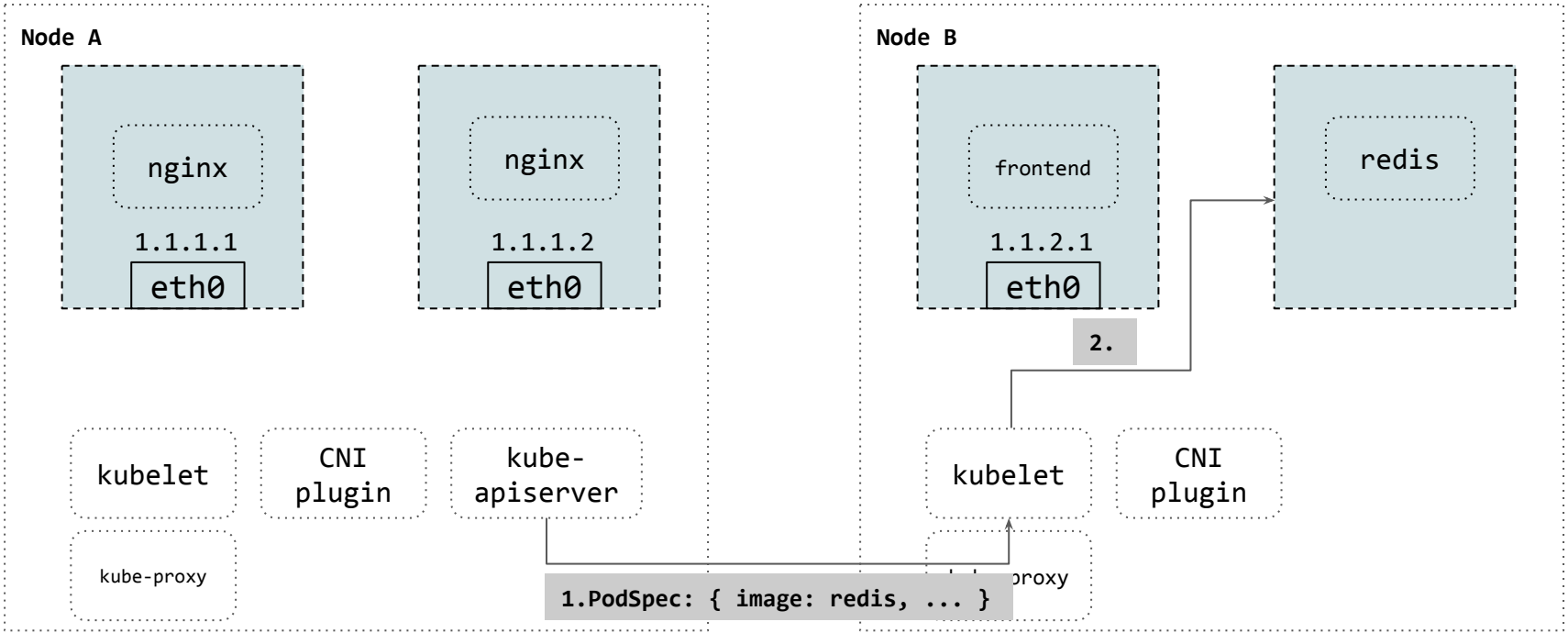
k8s networking: overview



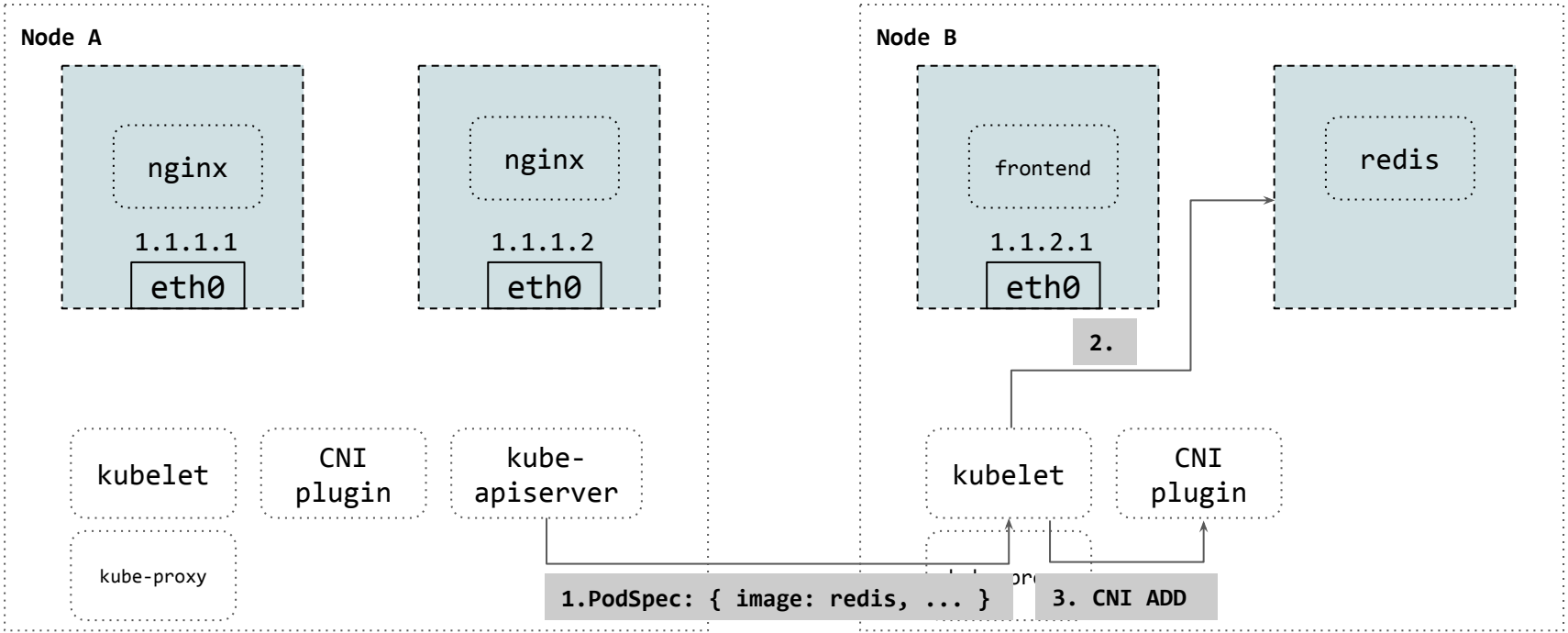
k8s networking: overview



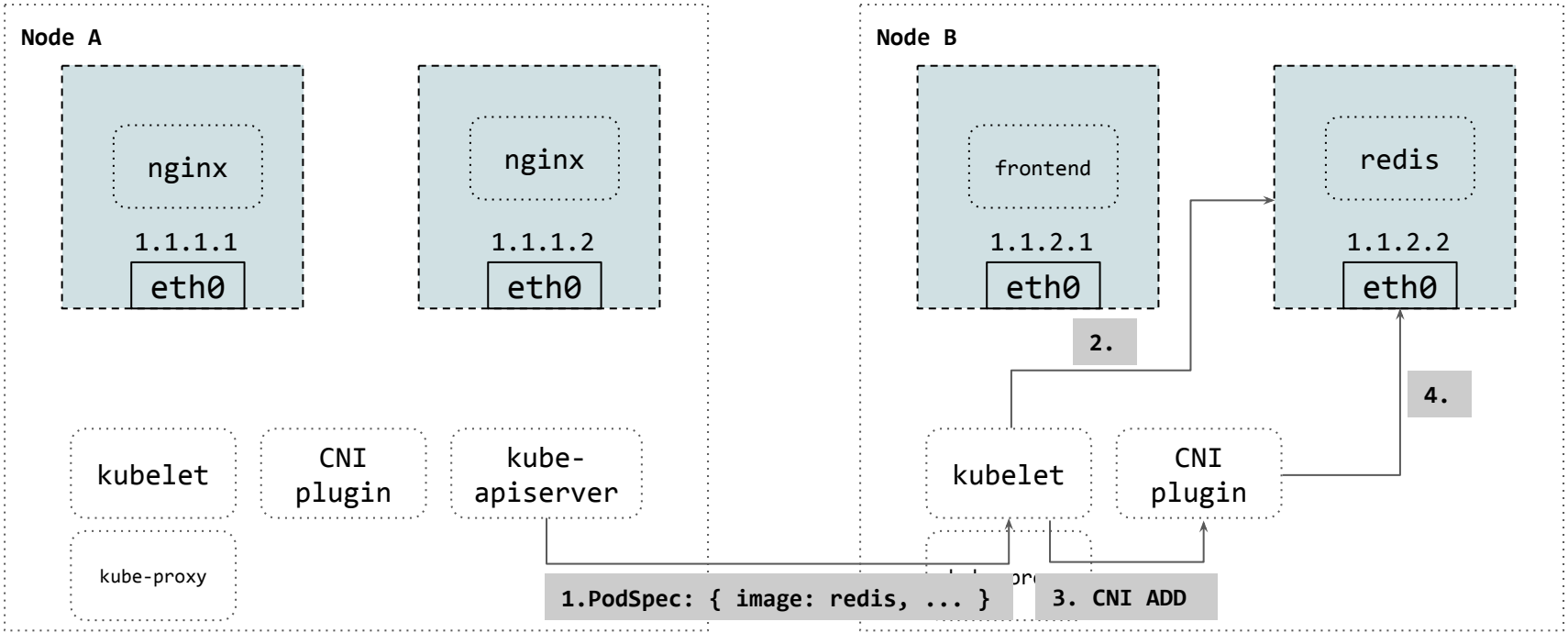
k8s networking: overview



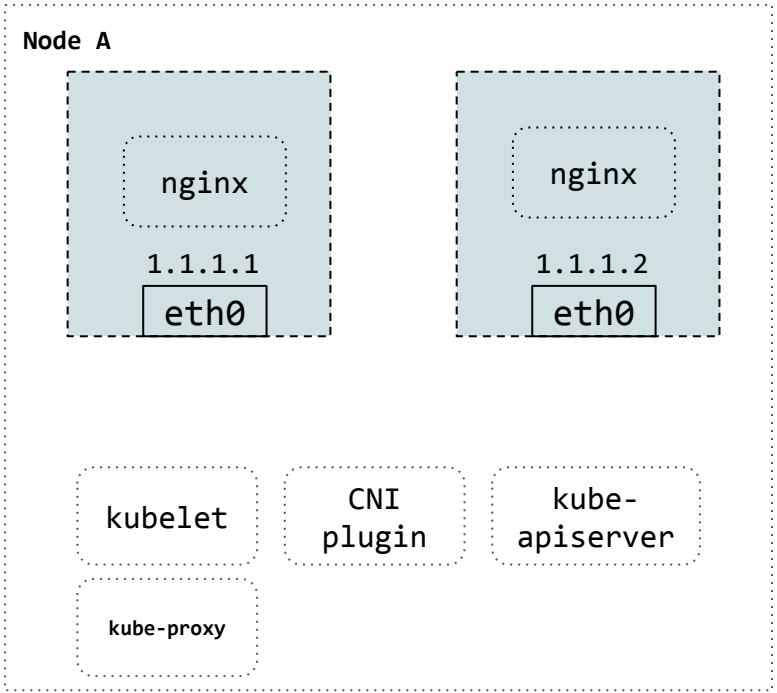
k8s networking: overview



k8s networking: overview

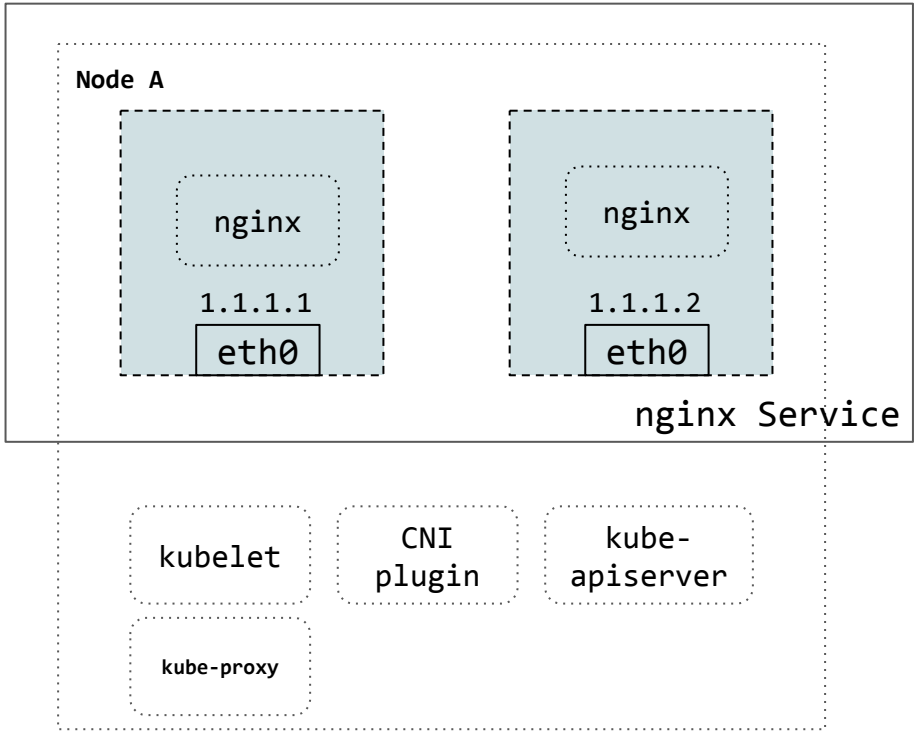


k8s networking: services




```
$ cat nginx-svc.yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80

$ kubectl get service nginx
CLUSTER-IP   PORT(S)
3.3.3.3      80/TCP

$ kubectl get endpoints nginx
ENDPOINTS
1.1.1.1:80, 1.1.1.2:80
```

k8s networking: services




```
$ cat nginx-svc.yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80

$ kubectl get service nginx
CLUSTER-IP   PORT(S)
3.3.3.3      80/TCP

$ kubectl get endpoints nginx
ENDPOINTS
1.1.1.1:80, 1.1.1.2:80
```

k8s networking: services

Service types:

- ClusterIP
- NodePort
- LoadBalancer
- ExternalName

kube-proxy:

- Proxies requests to services and their backend pods

k8s networking: ClusterIP

- ➔ Virtual IP to any endpoint (pod)
- ➔ Only in-cluster access

```
-t nat -A {PREROUTING, OUTPUT} -m conntrack --ctstate NEW -j KUBE-SERVICES

-A KUBE-SERVICES ! -s 1.1.0.0/16 -d 3.3.3.3/32 -p tcp -m tcp --dport 80 -j KUBE-MARK-MASQ
-A KUBE-SERVICES -d 3.3.3.3/32 -p tcp -m tcp --dport 80 -j KUBE-SVC-NGINX

-A KUBE-SVC-NGINX -m statistic --mode random --probability 0.50 -j KUBE-SEP-NGINX1
-A KUBE-SVC-NGINX -j KUBE-SEP-NGINX2

-A KUBE-SEP-NGINX1 -s 1.1.1.1/32 -j KUBE-MARK-MASQ
-A KUBE-SEP-NGINX1 -p tcp -m tcp -j DNAT --to-destination 1.1.1.1:80
-A KUBE-SEP-NGINX2 -s 1.1.1.2/32 -j KUBE-MARK-MASQ
-A KUBE-SEP-NGINX2 -p tcp -m tcp -j DNAT --to-destination 1.1.1.2:80
```

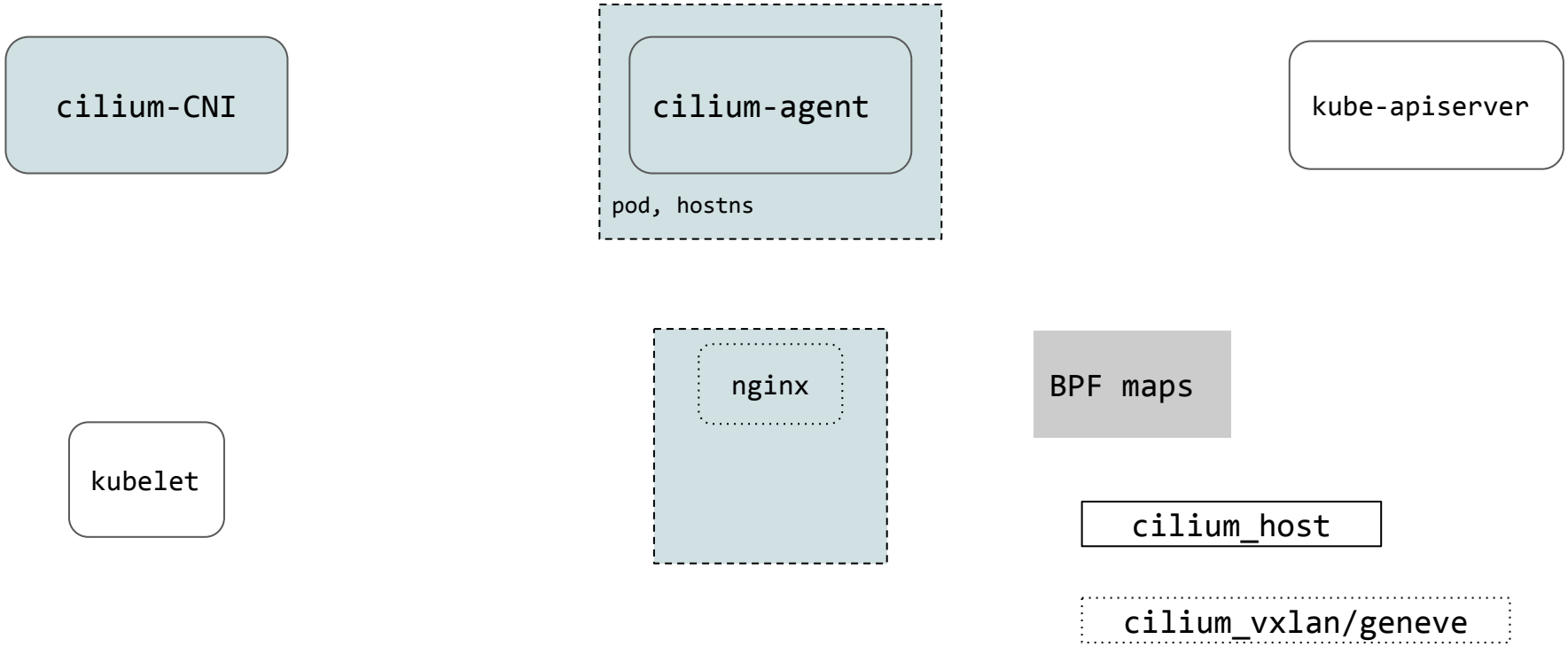
k8s networking: NodePort

- ➔ External node IP + port in NodePort range to any endpoint (pod), e.g. 10.0.0.1:31000
- ➔ Enables access from outside

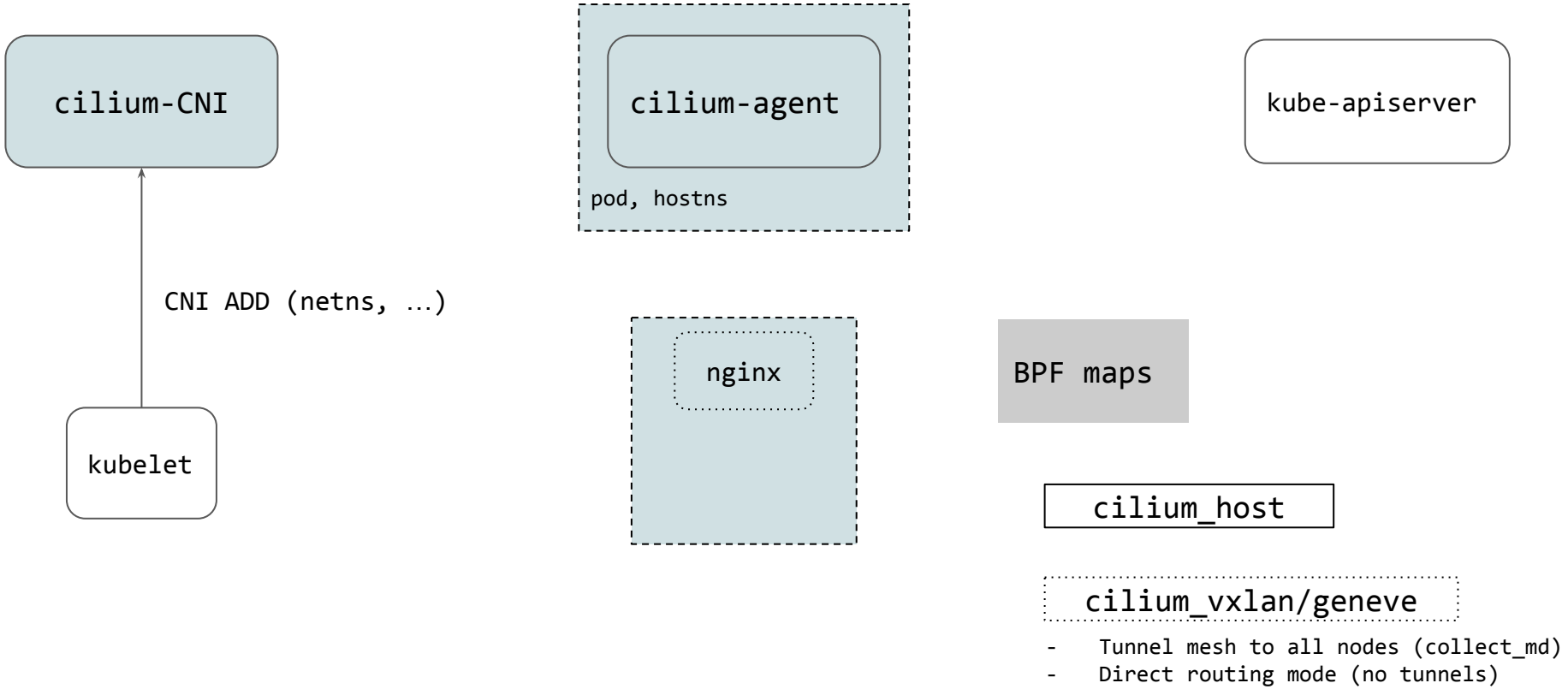
```
-t nat -A {PREROUTING, OUTPUT} -m conntrack --ctstate NEW -j KUBE-SERVICES

-A KUBE-SERVICES ! -s 1.1.0.0/16 -d 3.3.3.3/32 -p tcp -m tcp --dport 80 -j KUBE-MARK-MASQ
-A KUBE-SERVICES -d 3.3.3.3/32 -p tcp -m tcp --dport 80 -j KUBE-SVC-NGINX
-A KUBE-SERVICES -m addrtype --dst-type LOCAL -j KUBE-NODEPORTS

-A KUBE-NODEPORTS -p tcp -m tcp --dport 31000 -j KUBE-MARK-MASQ
-A KUBE-NODEPORTS -p tcp -m tcp --dport 31000 -j KUBE-SVC-NGINX

-A KUBE-SVC-NGINX -m statistic --mode random --probability 0.50 -j KUBE-SEP-NGINX1
-A KUBE-SVC-NGINX -j KUBE-SEP-NGINX2
```

Replacing kube-proxy with Cilium and BPF

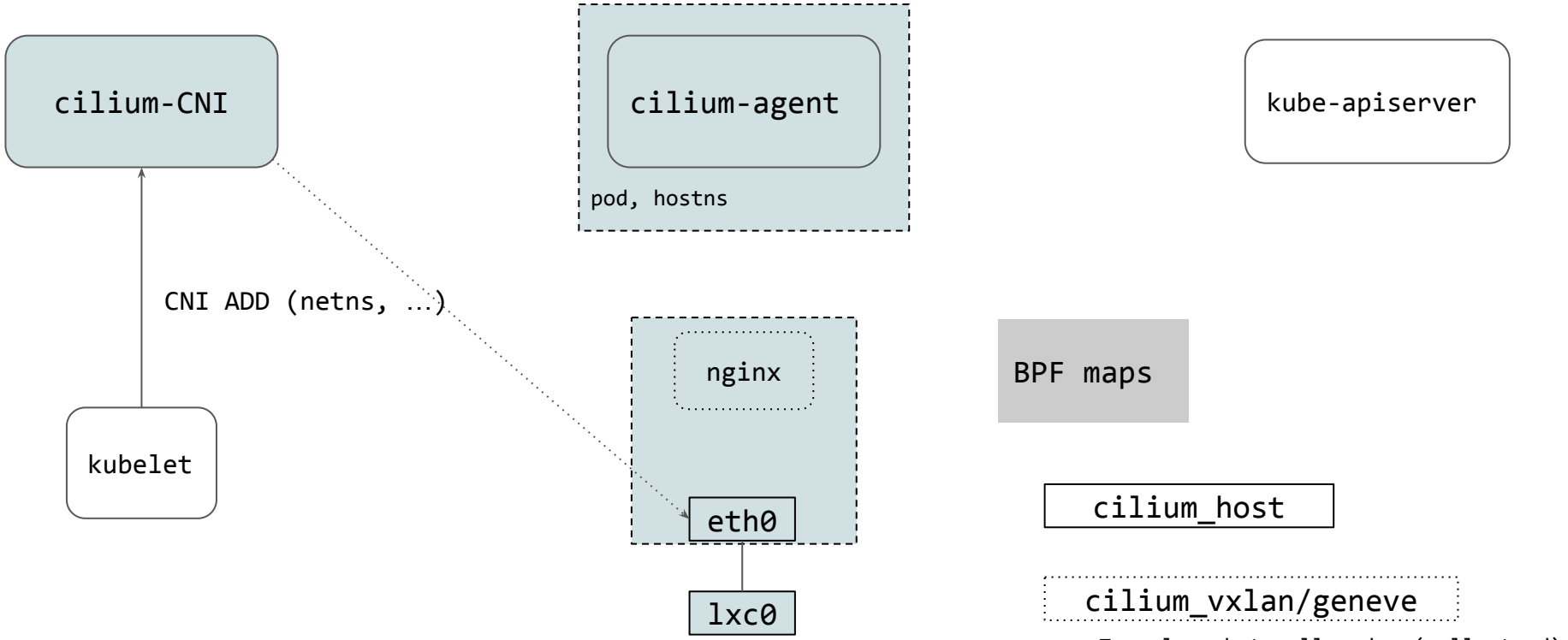
Cilium: overview



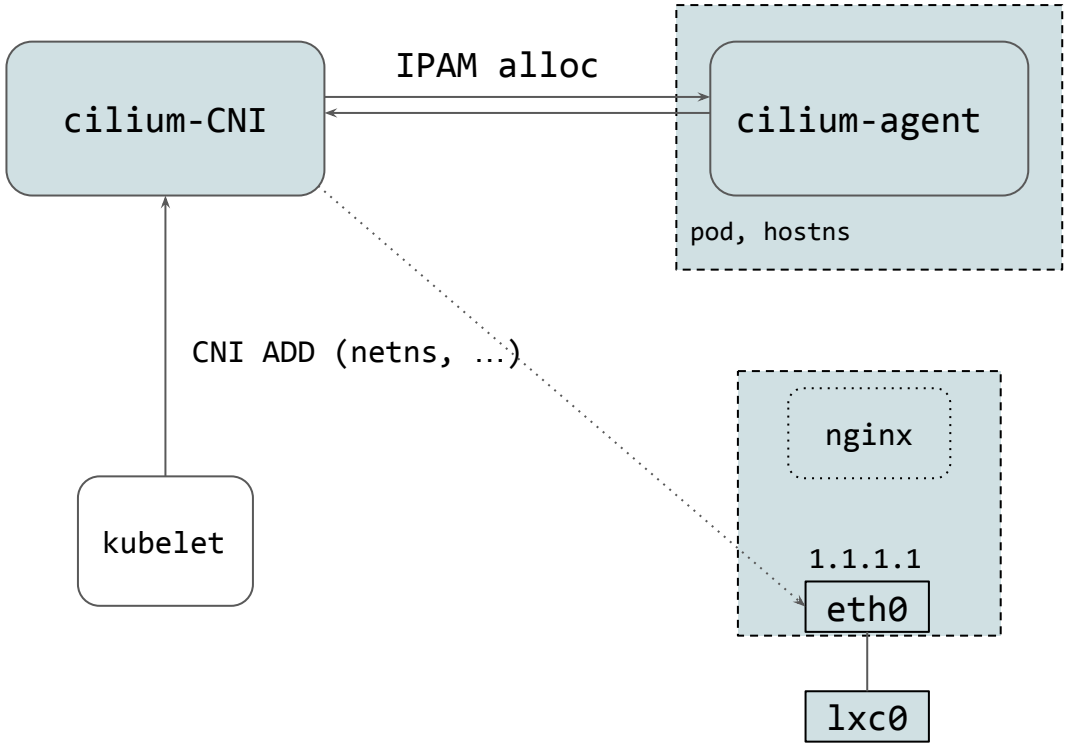
Cilium: overview



Cilium: overview



Cilium: overview



kube-apiserver

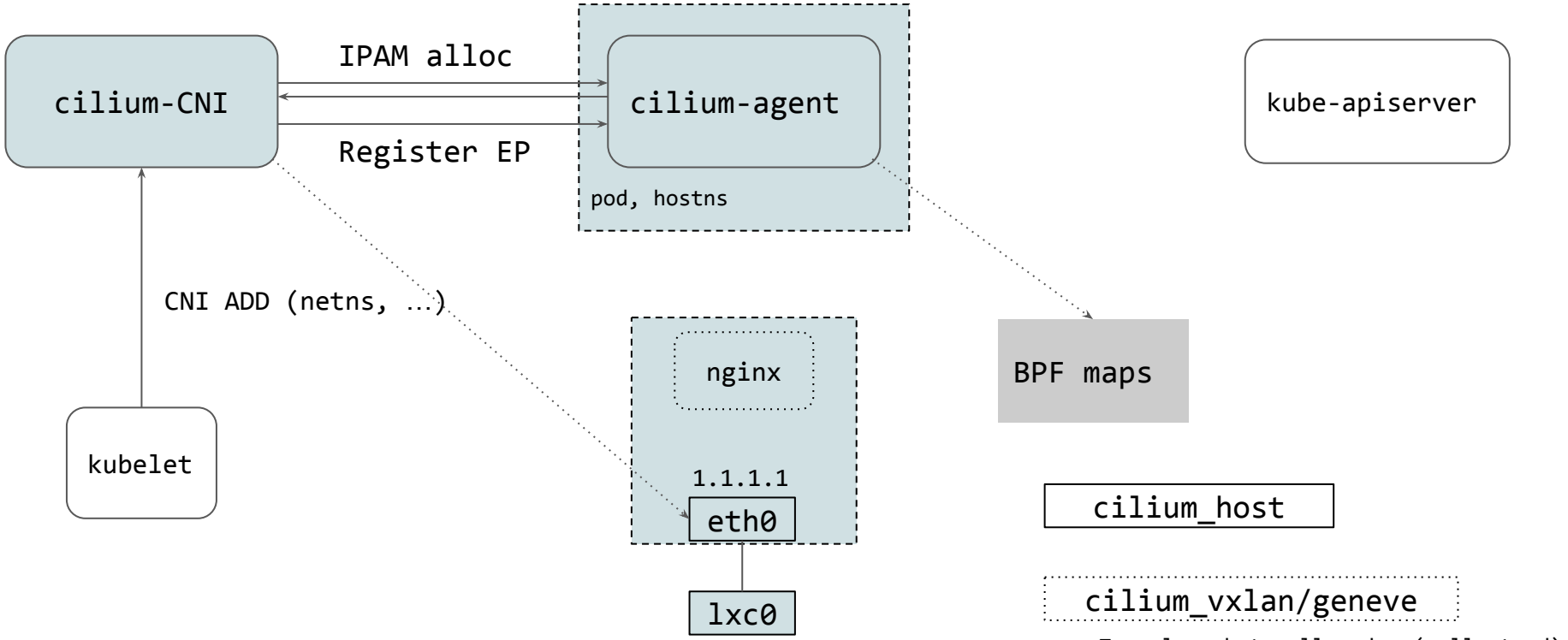
BPF maps

cilium_host

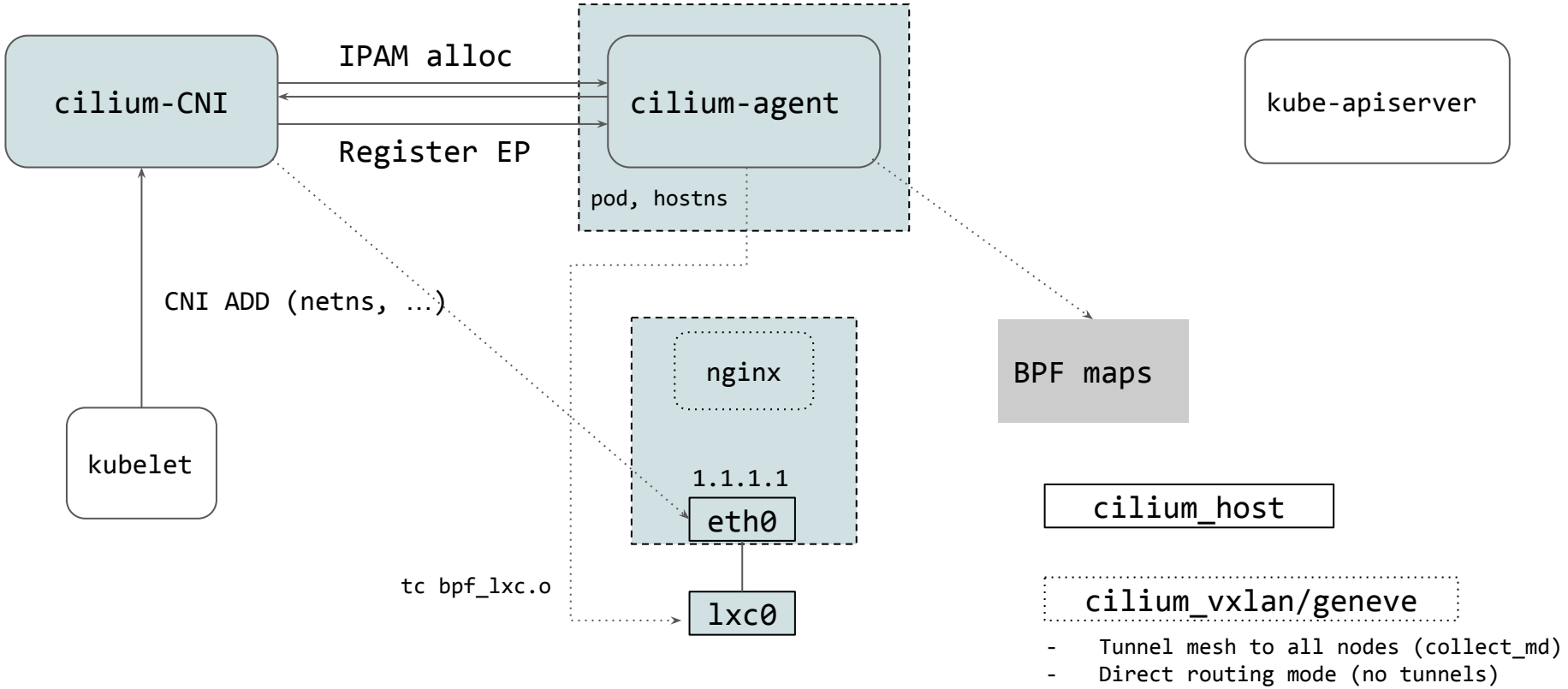
cilium_vxlan/geneve

- Tunnel mesh to all nodes (collect_md)
- Direct routing mode (no tunnels)

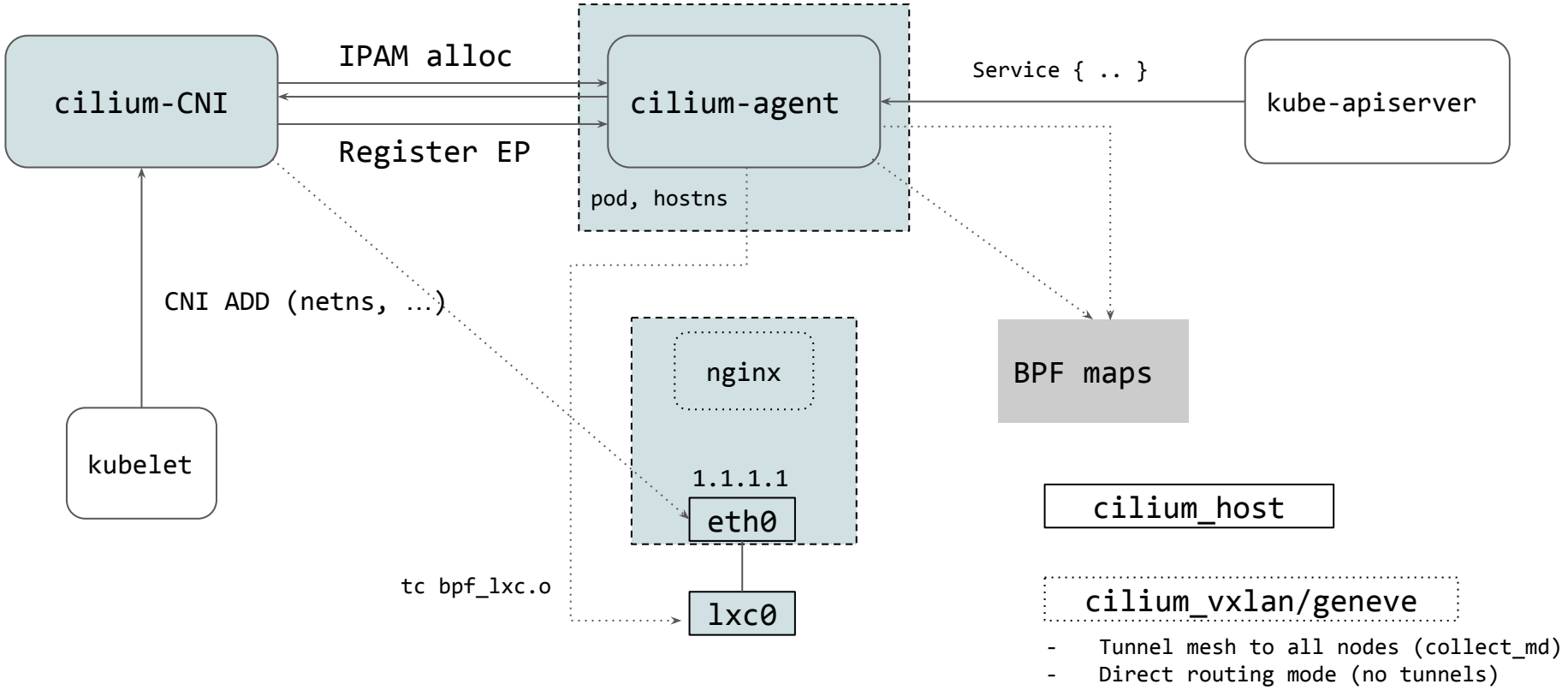
Cilium: overview



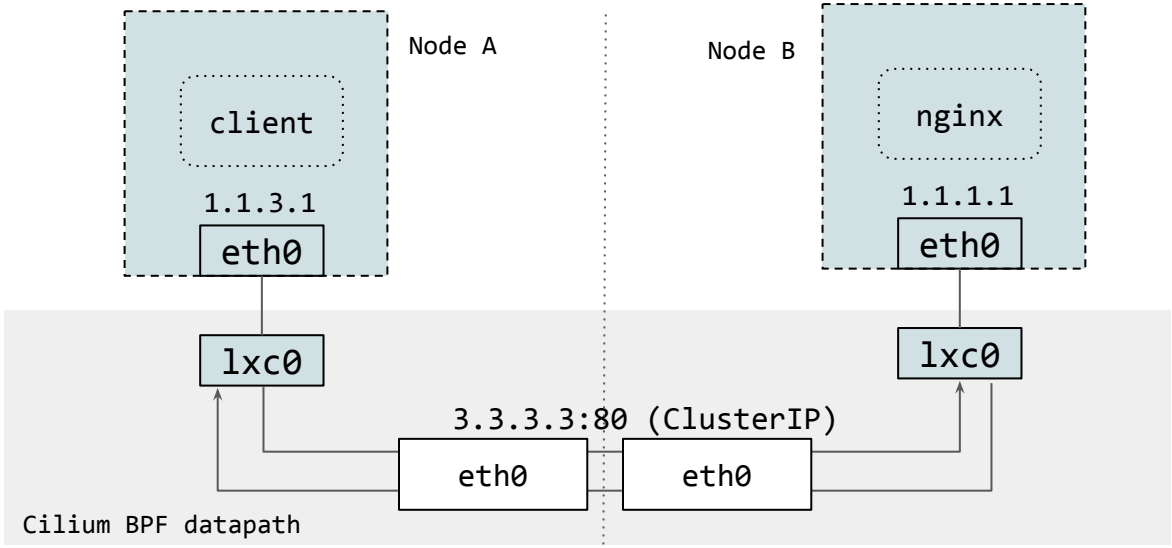
Cilium: overview



Cilium: overview

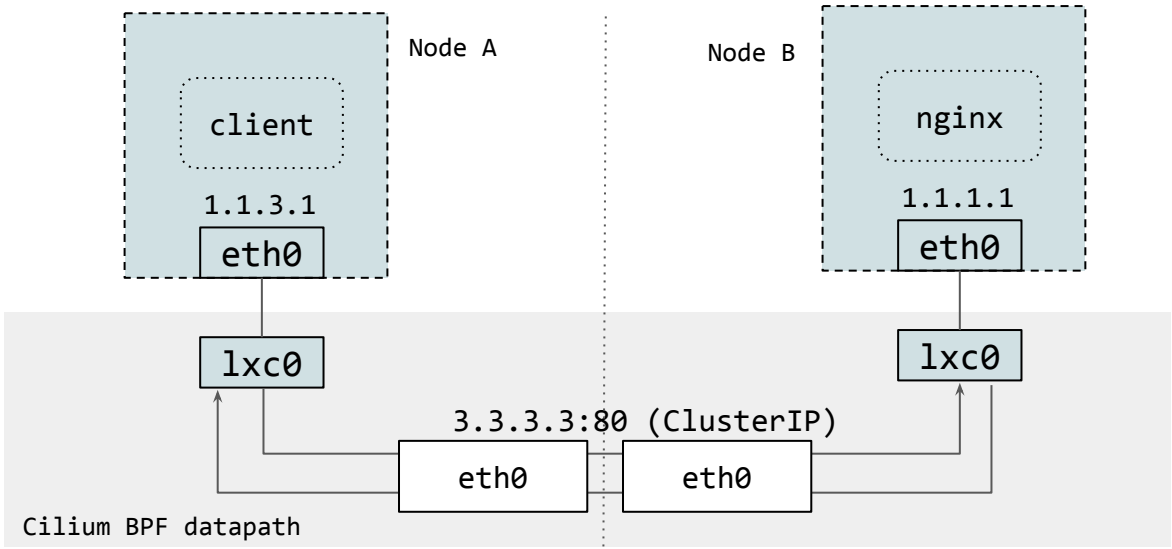


Cilium pre-v1.6: BPF ClusterIP



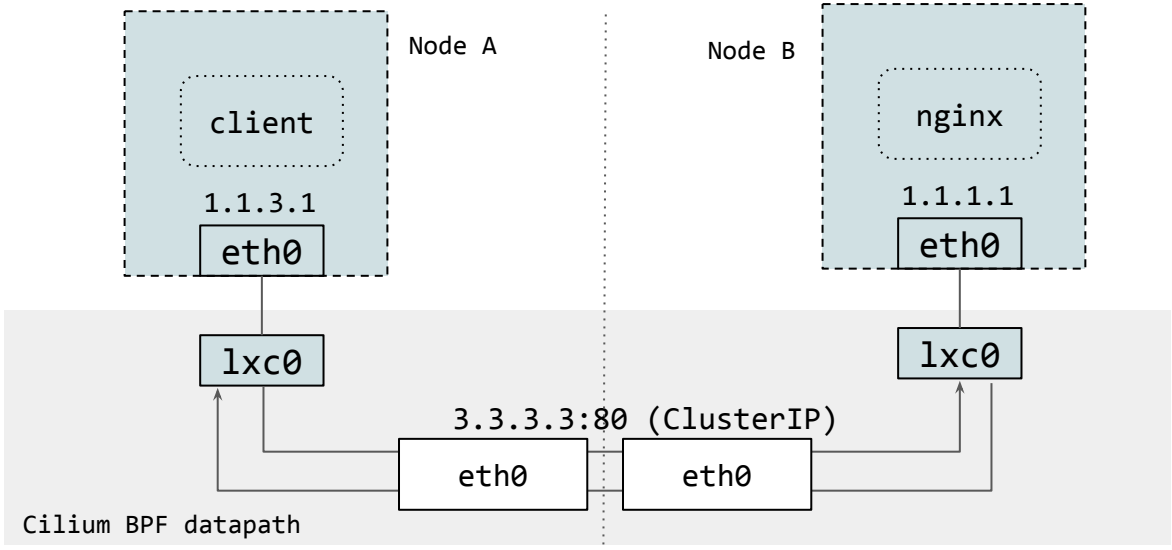
Cilium pre-v1.6: BPF ClusterIP

```
tc ingress bpf on lxc0:
 1.   Lookup SVC
 2.   If found:
       a.   Create SVC CT
       b.   DNAT
 3.   Create Egress CT

tc ingress bpf on eth0 (host):
 1.   Lookup Egress CT
 2.   If found:
       a.   Rev-NAT xlation
 3.   redirect to lxc0
```



Cilium pre-v1.6: BPF ClusterIP

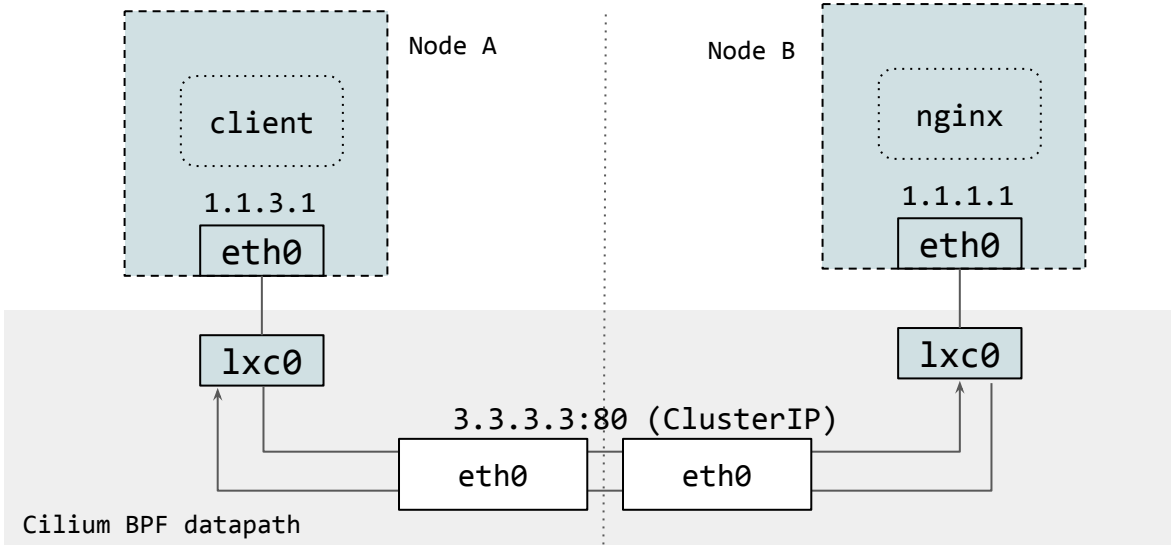
```
tc ingress bpf on lxc0:
 1.   Lookup SVC
 2.   If found:
       a.   Create SVC CT
       b.   DNAT
 3.   Create Egress CT

tc ingress bpf on eth0 (host):
 1.   Lookup Egress CT
 2.   If found:
       a.   Rev-NAT xlation
 3.   redirect to lxc0
```




```
                BPF SVC map
 SVC IP Port NR => ID EID Endpoint IP Port Count
-----------------------------------------------
3.3.3.3 80   0  => 1   0     0.0.0.0    0      2
3.3.3.3 80   1  => 1   4     1.1.1.1    80     0
3.3.3.3 80   2  => 1   5     1.1.1.2    80     0
```

Cilium pre-v1.6: BPF ClusterIP

```
tc ingress bpf on lxc0:
 1.   Lookup SVC
 2.   If found:
       a.   Create SVC CT
       b.   DNAT
 3.   Create Egress CT

tc ingress bpf on eth0 (host):
 1.   Lookup Egress CT
 2.   If found:
       a.   Rev-NAT xlation
 3.   redirect to lxc0
```



BPF SVC map

SVC IP	Port	NR	=>	ID	EID	Endpoint IP	Port	Count
3.3.3.3	80	0	=>	1	0	0.0.0.0	0	2
3.3.3.3	80	1	=>	1	4	1.1.1.1	80	0
3.3.3.3	80	2	=>	1	5	1.1.1.2	80	0

BPF conntrack LRU map

srcIP	sPort	dstIP	dPort	Type	=>	EID|SVCID
1.1.3.1	4321	3.3.3.3	80	SVC	=>	4
1.1.3.1	4321	1.1.1.1	80	Egress	=>	1
1.1.1.1	80	1.1.3.1	4321	Ingress	=>	

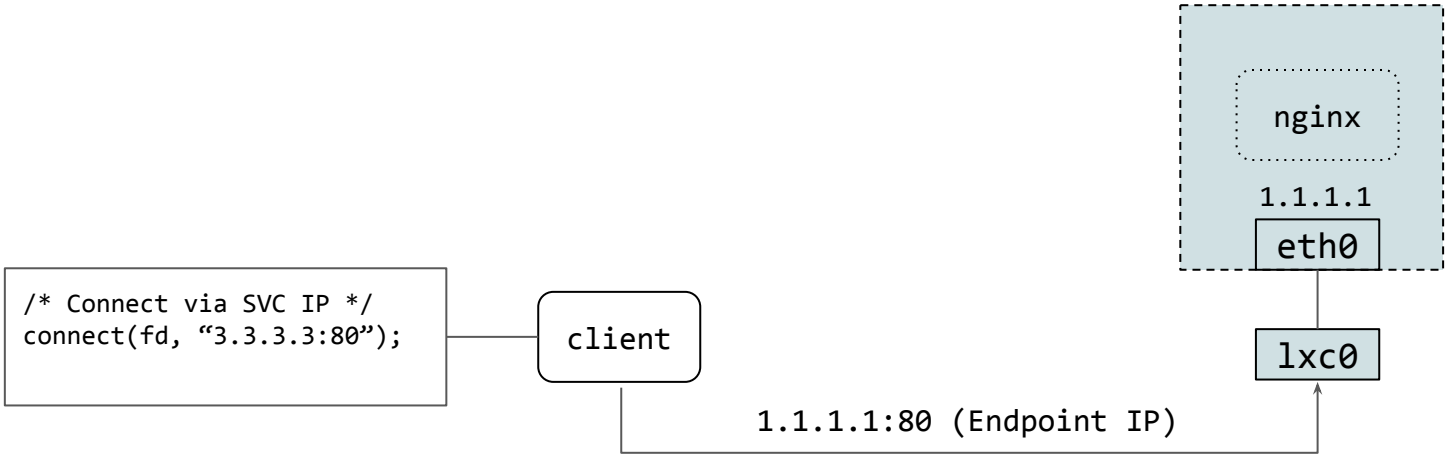
Cilium pre-v1.6

- ✓ Pod netns -> ClusterIP
- × Host netns -> ClusterIP
  - kube-proxy iptables
- × Any -> Public IP + NodePort
  - kube-proxy iptables

Cilium v1.6

- ✓ Pod netns -> ClusterIP
- ✓ Host netns -> ClusterIP
- ✓ Any -> Public IP + NodePort

Cilium v1.6: BPF ClusterIP



Cilium v1.6: BPF ClusterIP

- Transparent socket-based load-balancing
- Attach on the cgroupv2 root mount **BPF_PROG_TYPE_CGROUP_SOCK_ADDR**:
  - **BPF_CGROUP_INET{4,6}_CONNECT** - TCP, connected UDP
  - **BPF_CGROUP_UDP{4,6}_SENDMSG** - UDP
  - Pseudo code:

```
int sock4_xlate(struct bpf_sock_addr *ctx) {
      struct lb4_svc_key key = { .dip = ctx->user_ip4, .dport = ctx->user_port };
      svc = lb4_lookup_svc(&key)
      if (svc) {
            ctx->user_ip4 = svc->endpoint_addr;
            ctx->user_port = svc->endpoint_port;
      }
      return 1;
}
```

- No conntrack, no packet header mangling
- Cost for LB paid only once for TCP and connected UDP
- Works in both - host netns and pod netns

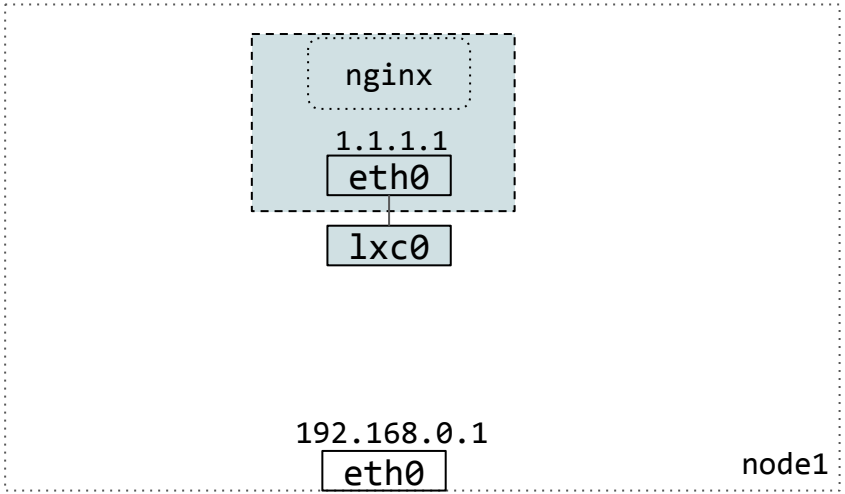
Cilium v1.6: BPF ClusterIP

- New attachment type **BPF_CGROUP_UDP{4,6}_RECVMSG**
- Create map for UDP rev-NAT xlation

```
                BPF rev NAT map
 Cookie Endpoint IP Port => Service ID IP        Port
-------------------------------------------------------
 42     1.1.1.1     80   => 1          3.3.3.30  80
```

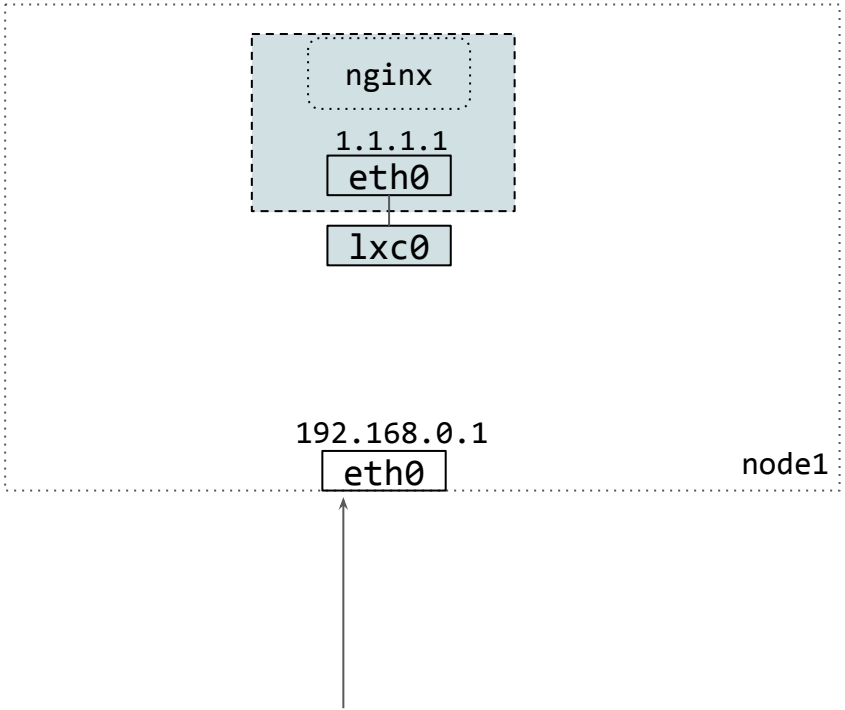
- Socket cookie to avoid collisions via bpf_get_socket_cookie()
- Insert/update map upon **connect(2)** or **sendmsg(2)**
- Do reverse NAT xlation upon **recvmsg(2)**

Cilium v1.6: BPF NodePort



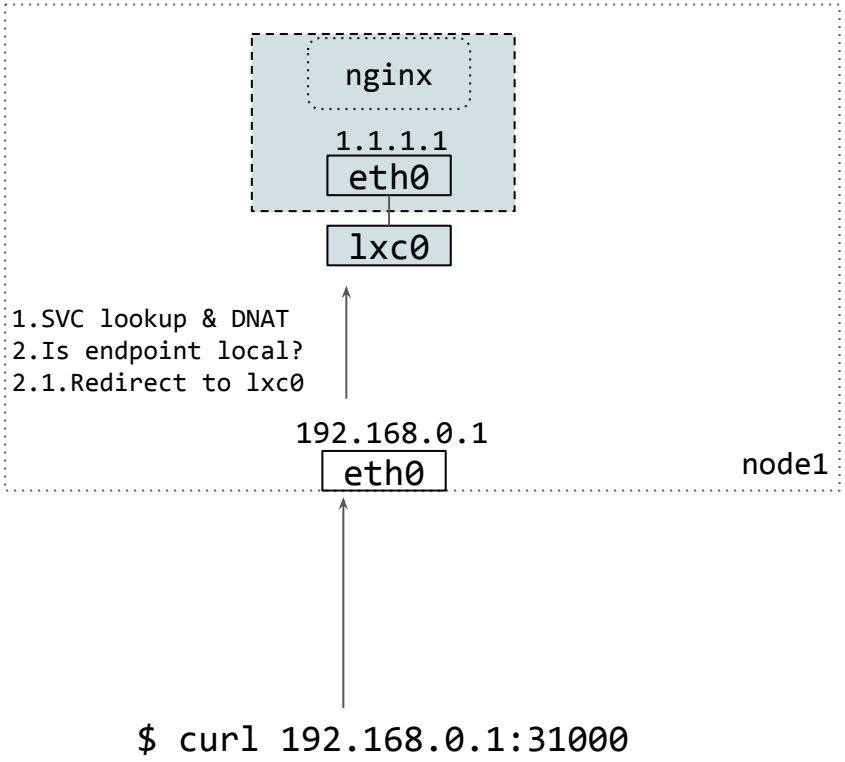

```
$ cat nginx-np-svc.yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
  type: NodePort

$ kubectl get service nginx
CLUSTER-IP   PORT(S)
3.3.3.3      80:31000/TCP
```

Cilium v1.6: BPF NodePort



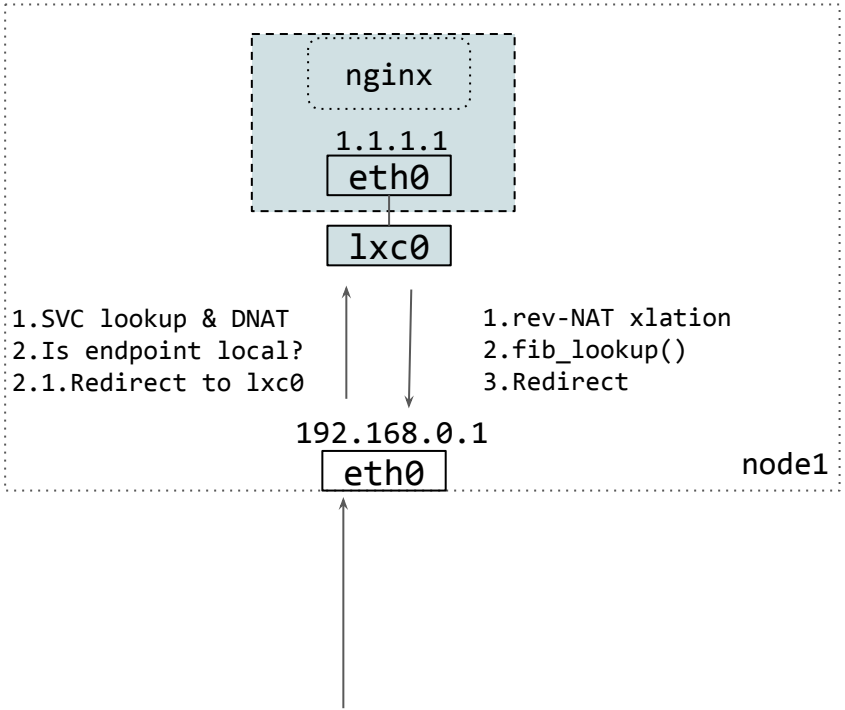

```
$ cat nginx-np-svc.yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
  type: NodePort

$ kubectl get service nginx
CLUSTER-IP   PORT(S)
3.3.3.3      80:31000/TCP
```

Cilium v1.6: BPF NodePort



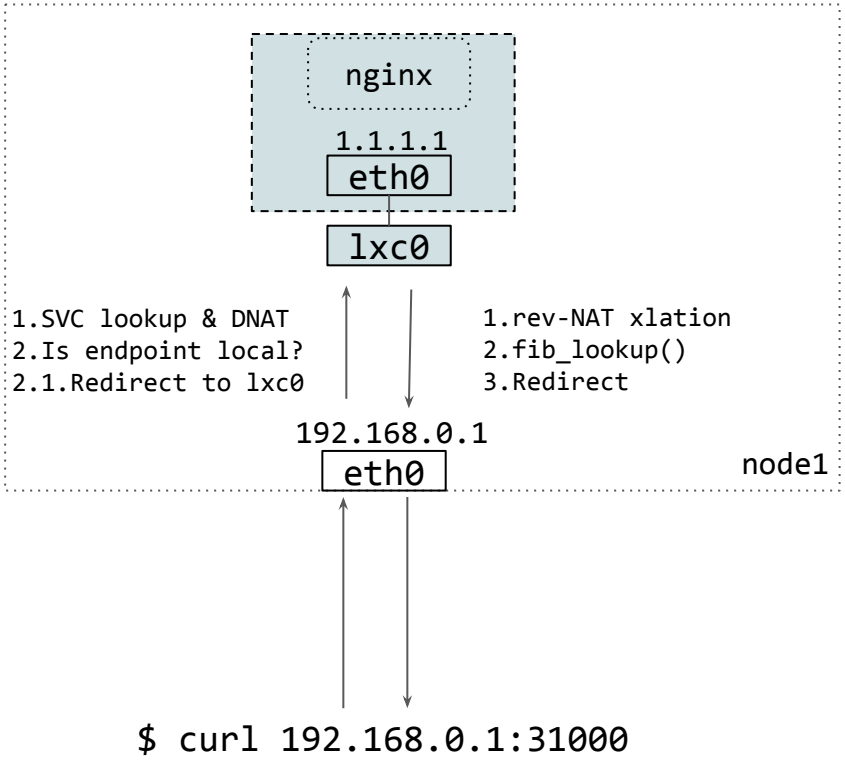

```
$ cat nginx-np-svc.yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
  type: NodePort

$ kubectl get service nginx
CLUSTER-IP   PORT(S)
3.3.3.3      80:31000/TCP
```

Cilium v1.6: BPF NodePort



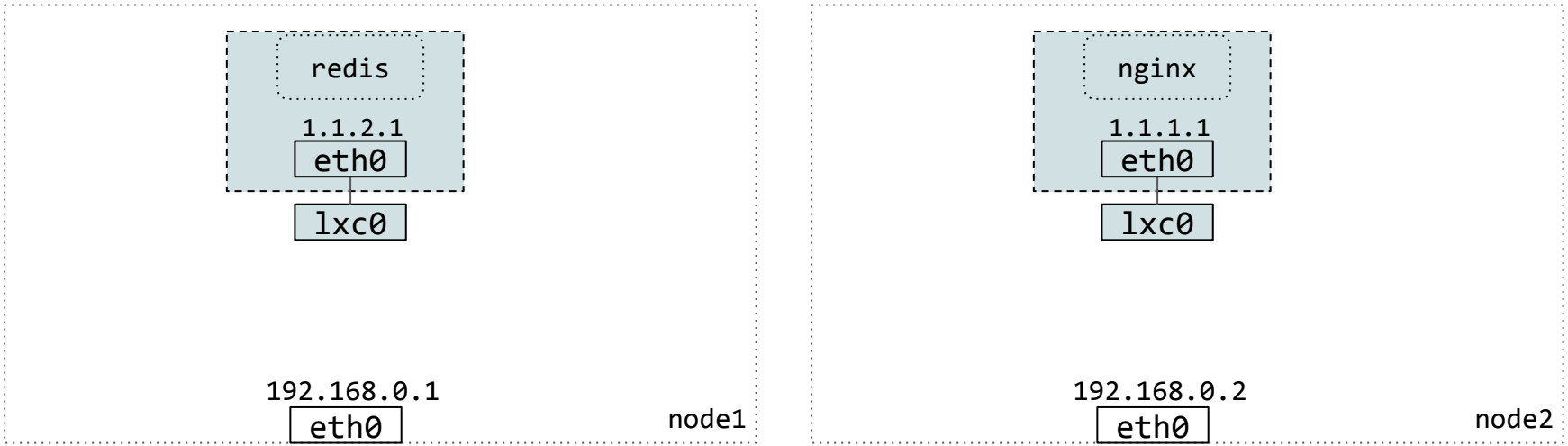

```
$ cat nginx-np-svc.yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
  type: NodePort

$ kubectl get service nginx
CLUSTER-IP   PORT(S)
3.3.3.3      80:31000/TCP
```

Cilium v1.6: BPF NodePort

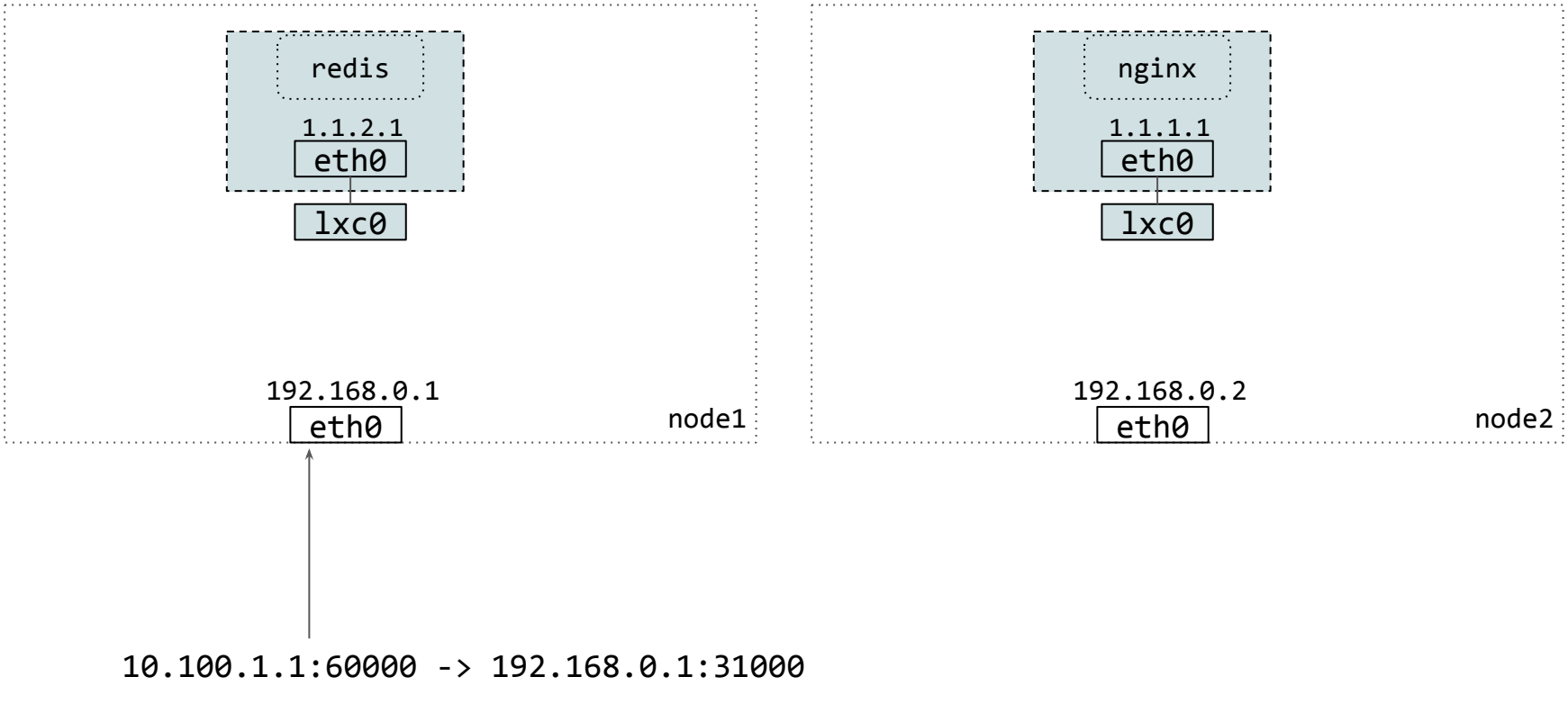



```
$ cat nginx-np-svc.yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
  type: NodePort

$ kubectl get service nginx
CLUSTER-IP   PORT(S)
3.3.3.3      80:31000/TCP
```

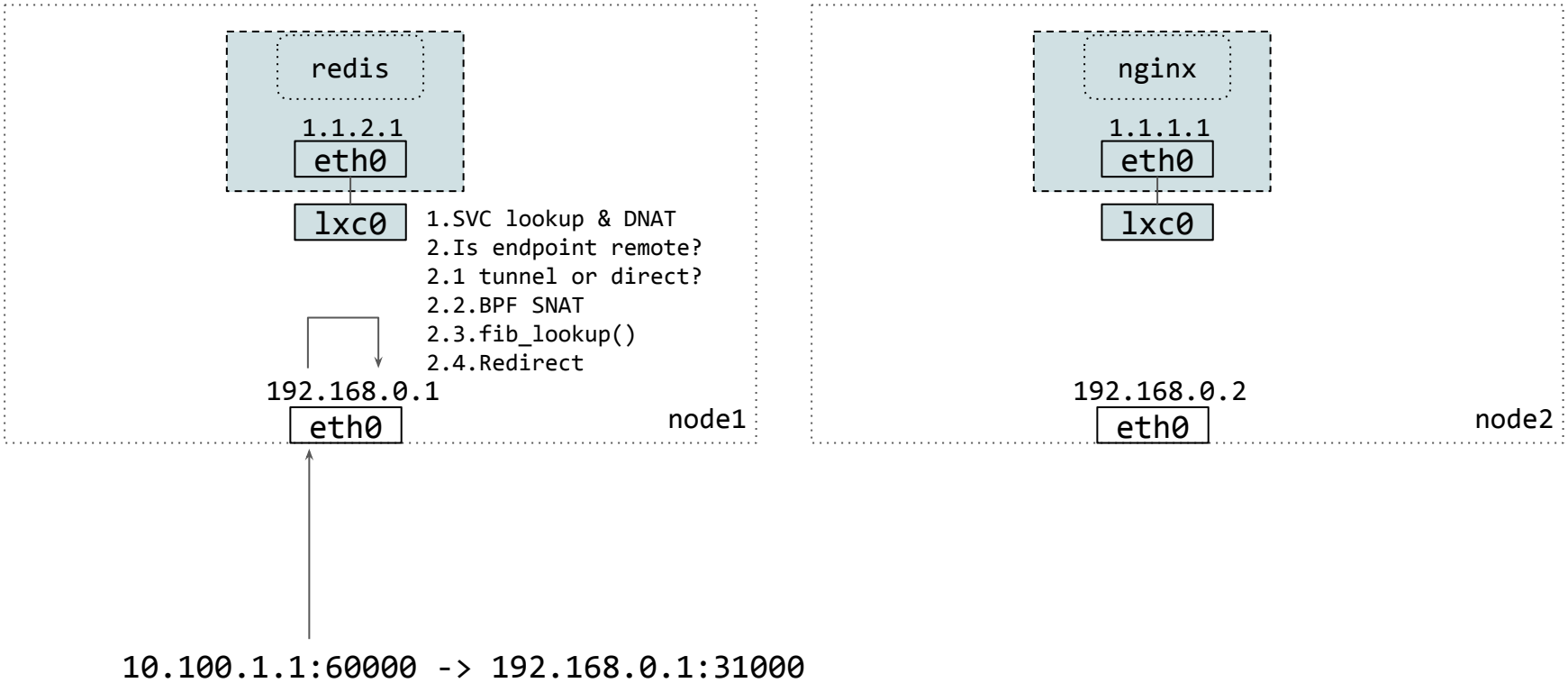
Cilium v1.6: BPF NodePort



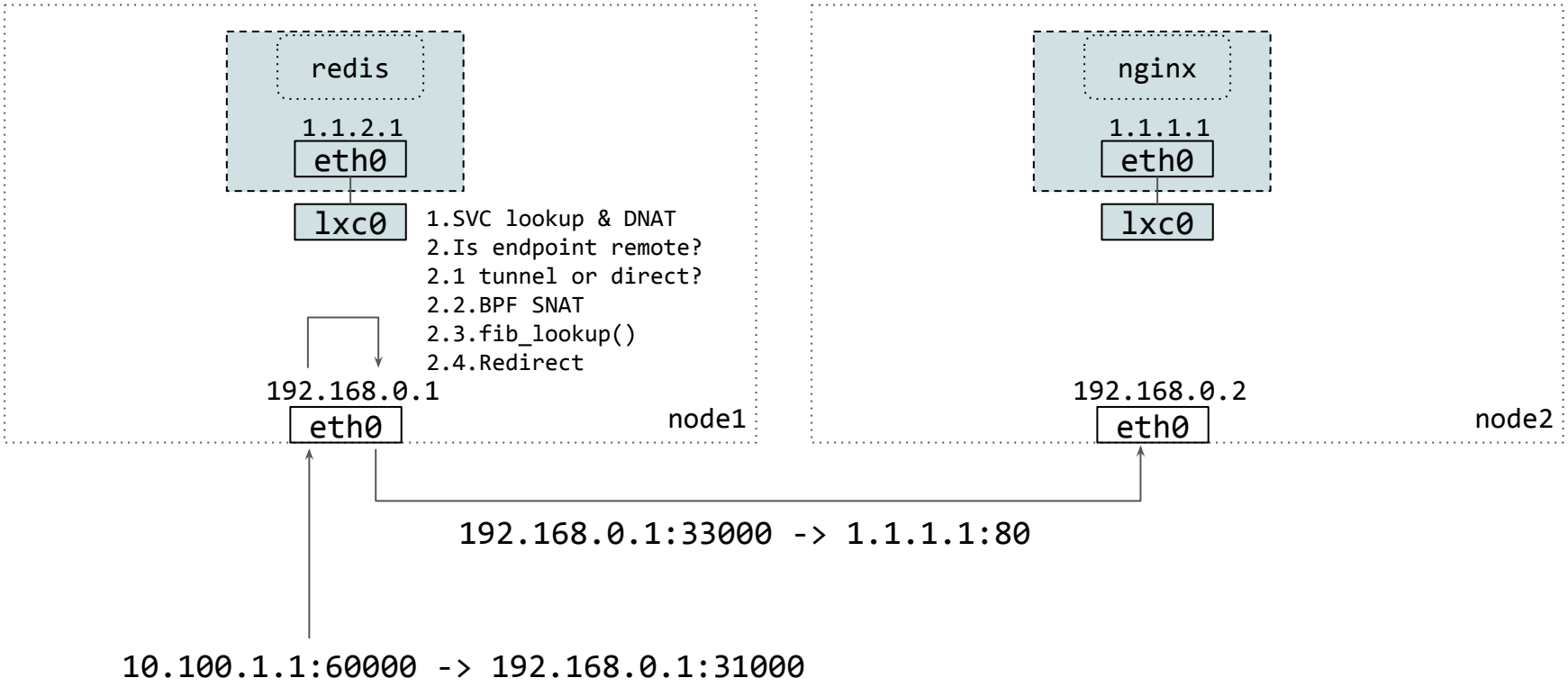
Cilium v1.6: BPF NodePort



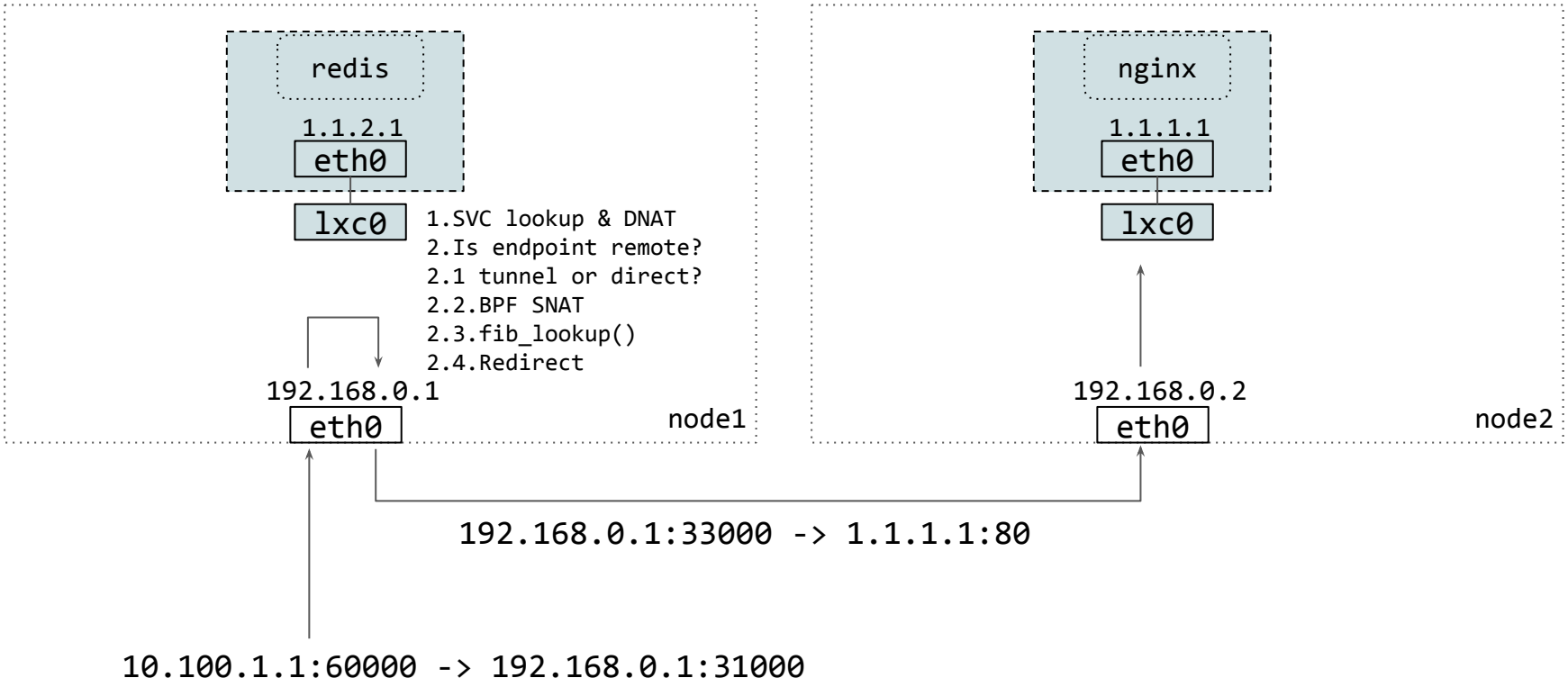
Cilium v1.6: BPF NodePort



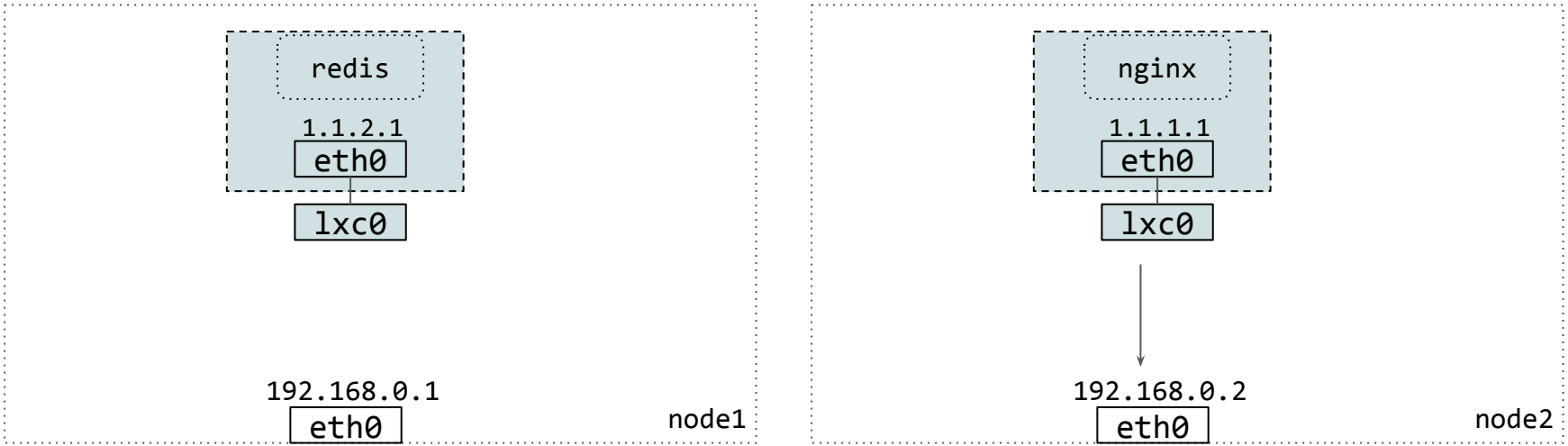
Cilium v1.6: BPF NodePort



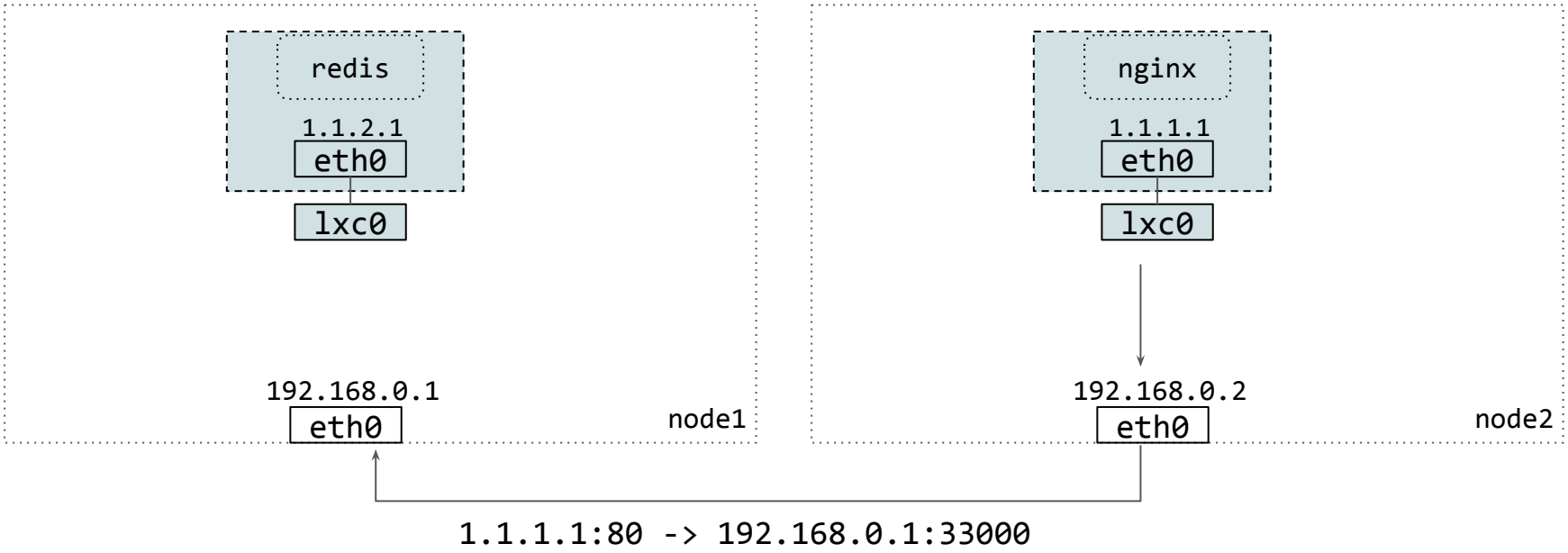
Cilium v1.6: BPF NodePort



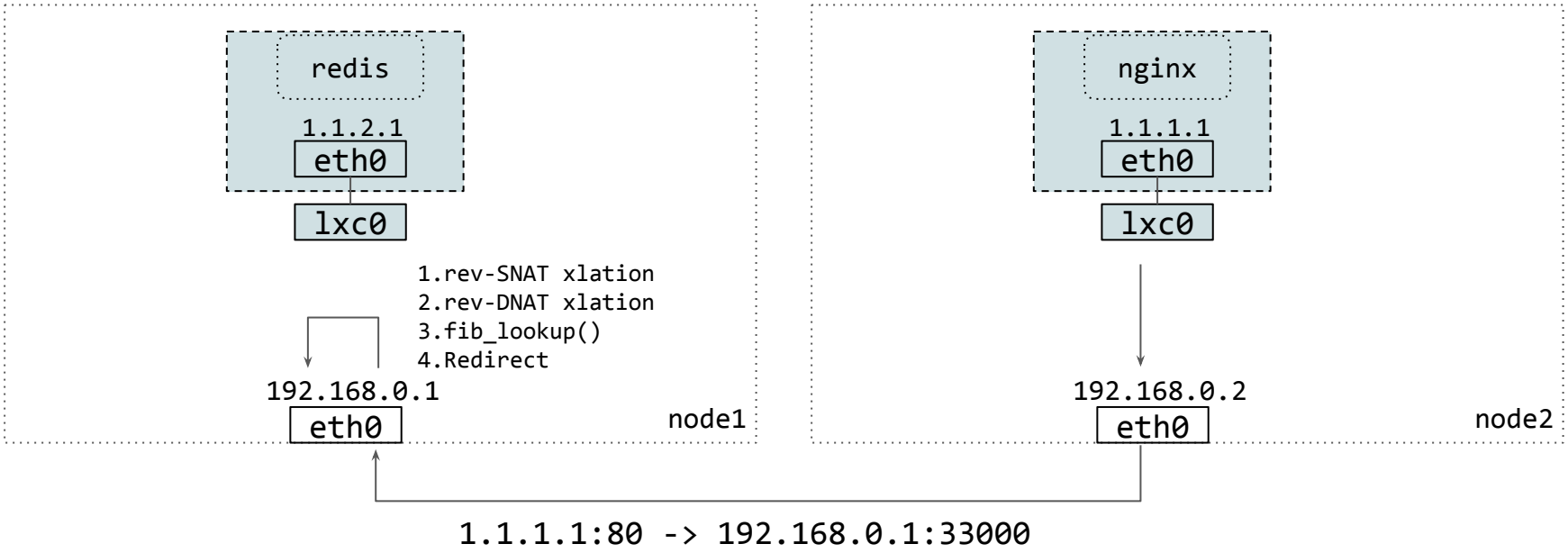
Cilium v1.6: BPF NodePort



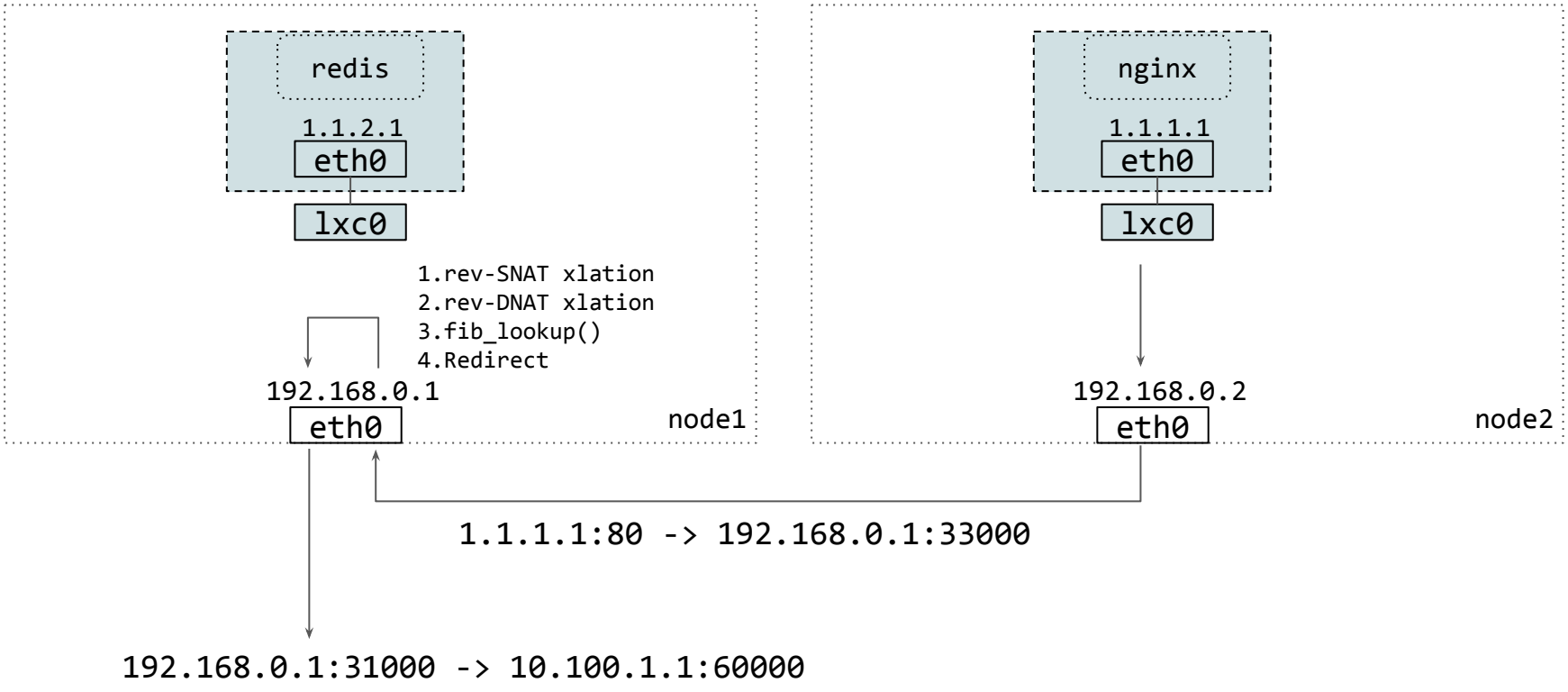
Cilium v1.6: BPF NodePort



Cilium v1.6: BPF NodePort



Cilium v1.6: BPF NodePort



Cilium v1.6: BPF NodePort

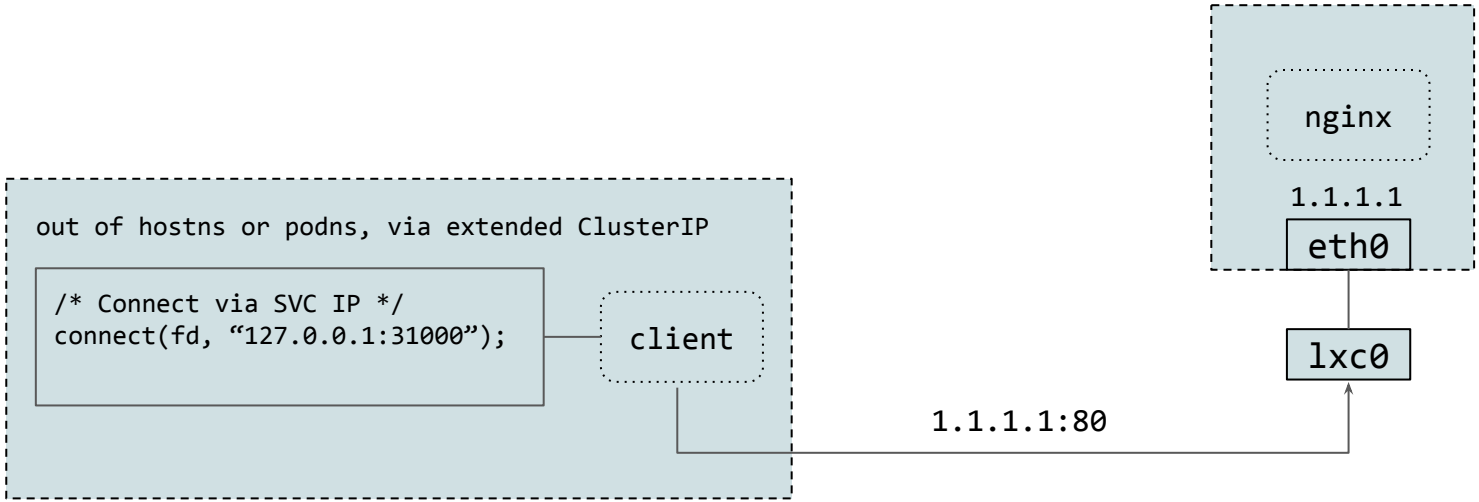
SNAT:

- BPF map to store SNAT mappings

```
 saddr     sport  daddr       dport dir     => addr        dport
-------------------------------------------------------------------
10.100.1.1 60000  1.1.1.1     80    Egress  => 192.168.0.1 33000
1.1.1.1    80     192.168.0.1 33000 Ingress => 10.100.1.1  66000
```

- #pragma unroll: hash and prandom() for sport collision resolution
- Additional NAT mapping for BPF conntrack for tracking local flows

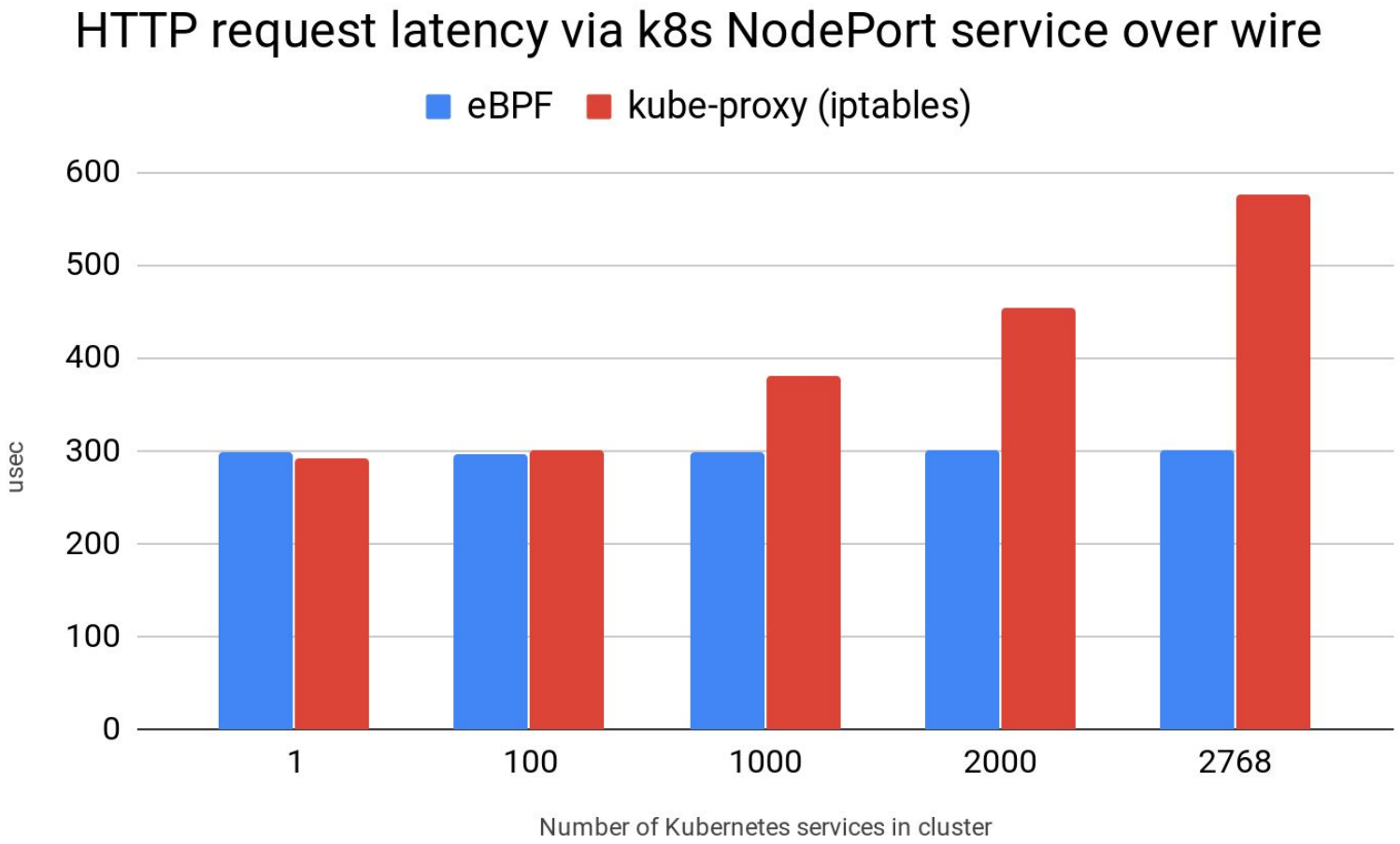
Cilium v1.6: BPF NodePort



iptables rules (5k services)

iptables-save | grep ‘\-A KUBE’ | wc -l:

- With kube-proxy: 25401
- With BPF: 4

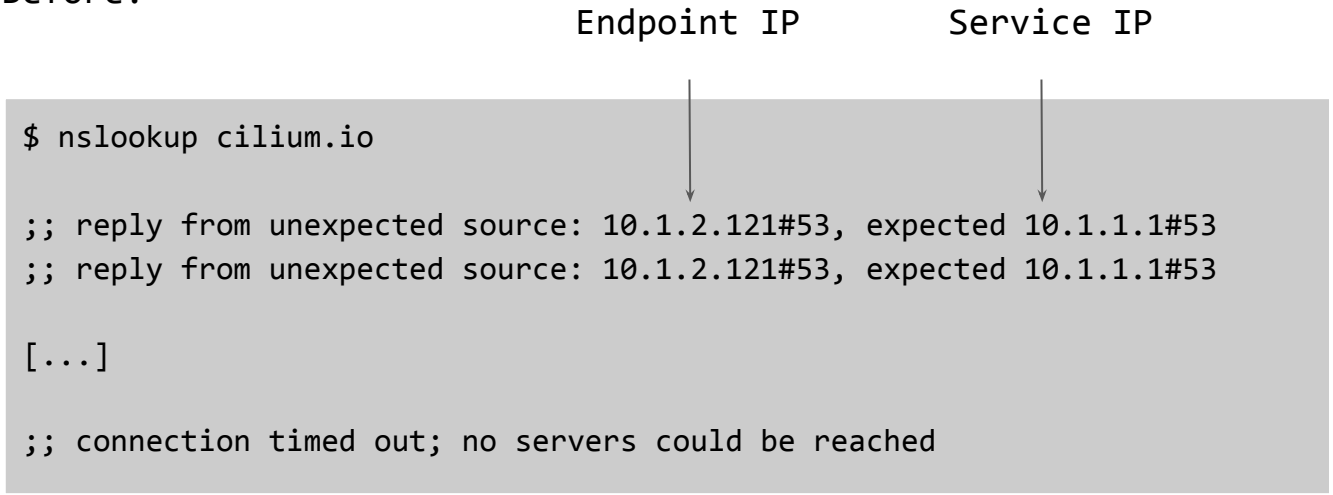

HTTP request latency via k8s NodePort service over wire
eBPF
kube-proxy (iptables)
600
500
400
300
200
100
0
usec
1
100
1000
2000
2768
Number of Kubernetes services in cluster

Related BPF & Cilium improvements

BPF UDP recvmsg hook

- Plain attachment to UDP sendmsg insufficient: e.g. breaks DNS
- 983695fa6765 (“bpf: fix unconnected udp hooks”) added recvmsg handling
- BPF ClusterIP performs reverse mapping without packet rewrite

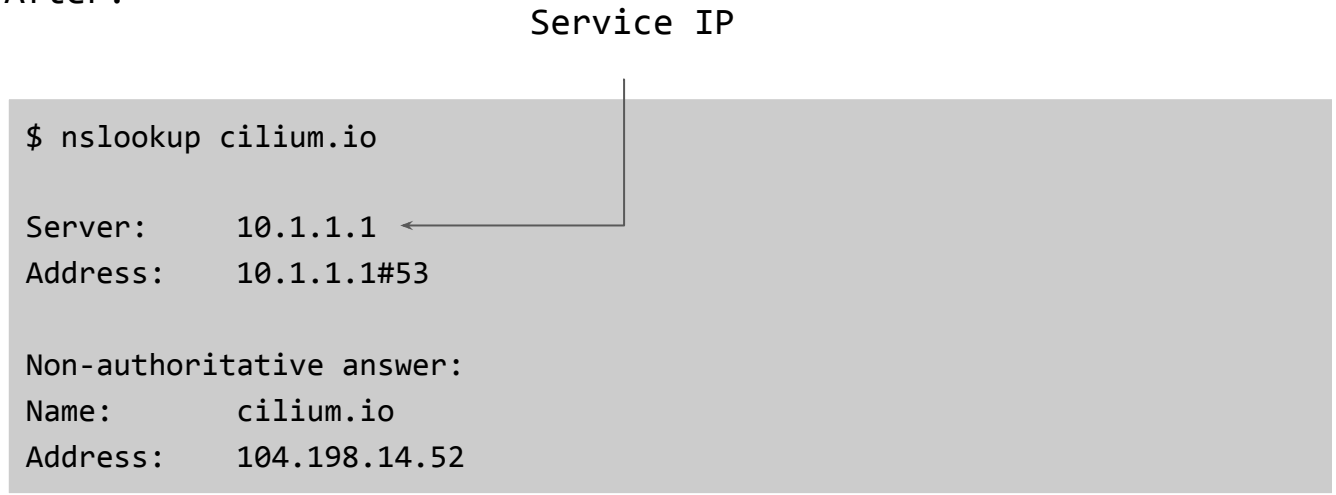
- Before:



BPF UDP recvmsg hook

- Plain attachment to UDP sendmsg insufficient: e.g. breaks DNS
- 983695fa6765 (“bpf: fix unconnected udp hooks”) added recvmsg handling
- BPF ClusterIP performs reverse mapping without packet rewrite

- After:



Global socket cookies

- BPF ClusterIP holds LRU reverse mapping table for UDP
- Socket cookie part of the key, but only unique per netns due to cookie generator
- Fixed via cd48bdda4fb8 (“sock: make cookie generation global instead of per netns”)

Managed neighbor entries for backends 1/2

- Cilium orchestration layer pushes services down into BPF datapath
- When BPF NodePort service backend is remote, skb at BPF tc ingress is eventually pushed back out via redirect()
  - Tunneling: BPF datapath zeroes inner src/dst mac
  - Direct routing: fib_lookup() needed for src/dst/ifindex
    - Fails if no neigh entry for backend, no ARP probe either

Managed neighbor entries for backends 1/2

- Workaround to overcome failing fib_lookup()
  - Orchestration layer resolves backend and pushes NUD_PERMANENT down
  - Seems okay since fixed/controlled number of entries
- New NUD_* type where orchestration layer only pushes down L3 address and kernel takes over its maintenance
  - Orchestration layer won’t need to deal with L2 anymore

Managed neighbor entries for backends 2/2

- Same issue for original NodePort requests from outside node
- Given it’s all pre-routing, we track L3->L2 mapping in BPF LRU map
- More suitable the BPF way due to potential neigh table overflow

LRU BPF callback on entry eviction

- CT map: LRU type is chosen if kernel supports it, else plain HT (down to 4.9)
- NAT map: separate from CT to avoid traffic disruptions in up/downgrade paths
- Agent has GC which kicks in at dynamic intervals based on stale entry load
  - 50b045a8c0cc (“bpf, lru: avoid messing with eviction heuristics upon syscall lookup”) fixed map walking from user space
  - GC in CT LRU case removes NAT mappings as well
  - Cost for GC can be avoided entirely with callback to invalidate NAT entry

LRU BPF eviction zones

- Partitioning of LRU recycling for global CT map depending of traffic type
  - E-W traffic in zone 1 (ClusterIP services, direct pod-{pod,host} traffic)
  - N-S traffic in zone 2 (NodePort services, ExternalName services)
  - Goal: single map for lookups, but guarantee that load in zone 2 cannot bring down zone 1 due to LRU evictions
- Could be realized via zone attribute on map update
- Specifying partitioning on map creation not too clear

BPF atomic ops

- NAT mapping state update for fast recycling of entries
- BPF spinlock possible, BUT ideally proper atomic ops avoiding 2 helper calls and storing spinlock into map (additionally avoids up/downgrade issues)
- Today only BPF_XADD to limited extend, and spinlocks for maps
- In future
  - READ_ONCE/WRITE_ONCE access semantics and BPF_XCHG, BPF_CMPXCHG
  - (Integration into klitmus suite)

BPF getpeername hook

- Missing coverage from the set of syscall BPF hooks on sockets
  - Use case: TCP, connected UDP
- Returns backend IP instead of service IP today

Improved mapping collision resolution

- Low hanging fruit for improving BPF SNAT implementation (Cilium-only)
  - Currently “outsourced” into tail call program to avoid hitting 4k insns limit
- Probing for newer kernels with 1 mio BPF insns/complexity limit + bounded loops to improve search for unused mappings

Thanks! Questions?

cilium.io
github.com/cilium/cilium

Try it out:

```
kubeadm init --pod-network-cidr=10.217.0.0/16 --skip-phases=addon/kube-proxy
kubeadm join [...]
helm template cilium \
      --namespace kube-system --set global.nodePort.enabled=true \
      --set global.k8sServiceHost=$API_SERVER_IP \
      --set global.k8sServicePort=$API_SERVER_PORT \
      --set global.tag=v1.6.1 > cilium.yaml
kubectl apply -f cilium.yaml
```