BPF as a revolutionary technology for the container landscape

Daniel Borkmann, Cilium.io
FOSDEM’20
Landscape: continuously decreasing lifetime

The short life of containers

Comparing container lifespans year over year, we found that the number of containers that are alive for 10 seconds or less has doubled to 22%. In fact, the number of containers that live for 5 minutes or less grew by 2X as well.

Source: sysdig ‘19 container usage report
Landscape: continuously increasing density

Containers-per-host density increases 100%

Over the past year, the median number of containers per host doubled to 30, compared to 15 in 2018 and 10 in 2017.

Source: sysdig ’19 container usage report
Landscape: Kubernetes as main orchestrator

Container orchestration: Kubernetes dominates

It's no surprise that as the de facto container orchestration tool, Kubernetes takes a whopping 77% share of orchestrators in-use. That number expands to 89% when you add in Red Hat OpenShift and Rancher – both built with Kubernetes. Here's the current breakdown:

Source: sysdig '19 container usage report
Landscape: Linux kernel as common denominator

Must provide building blocks for ...

- Isolation (namespaces)
- Resource management (cgroups)
- Network connectivity
- Security policies
- [...]  

… AND must withstand ever increasing scalability needs and high churn frequencies ...
Landscape: Linux kernel as common denominator

… while coping with subsystems and user interfaces originally designed long ago and subject to the “never break user space” paradigm.

Few examples in networking: tc, iptables/netfilter

Both designed for extensibility in general, but within inflexible overall framework for today’s needs.

Processing pipeline becomes part of the API contract.

Complex rules then significantly slow down fast-path.

Source: reddit.com/r/ArchitecturePorn/
Landscape: Linux kernel as common denominator

Given the need to support wide range of kernels, system software often stuck in such framework.

Policy logic then gets deeply baked into codebase, significant effort to rewrite.

Random pick, libnetwork:

```go
args := []string{
    "!(" + bridgeName + ", " + bridgeName + ", " + proto + ", " + strconv.Itoa(destPort) + ", " + destAddr + ", " + strconv.Itoa(destPort) + ", " + destAddr + ", " + action + ", " + filter + ")", "-j", "ACCEPT",
}
if err := ProgramRule(Filter, c.Name, action, args); err != nil {
    return err
}
```
Landscape: Linux kernel as common denominator

... but also Kubernetes itself relies a lot on iptables/netfilter for its Service implementation.

Issues in face of container scalability needs:

- Low and unpredictable packet latency
- Slow update time
- Reliability issues
- Inflexibility

https://github.com/kubernetes/community/blob/master/sig-scalability/blogs/k8s-services-scalability-issues.md (Jan 2020)
# perf top -a -e cycles:k

PerfTop:   16326 irqs/sec (all, 4 CPUs)

---------------------------------------------------------------------
  8.79% [kernel]    [k] native_sched_clock
  4.99% [ip_tables] [k] ipt_do_table
  3.09% [e1000e]    [k] e1000_irq_enable
  2.51% [nf_conntrack] [k] __nf_conntrack_find_get
  2.03% [kernel]    [k] fib_table_lookup
  1.98% [kernel]    [k] sched_clock_cpu
  1.75% [nf_conntrack] [k] tcp_packet
  1.65% [nf_conntrack] [k] nf_conntrack_tuple_taken
  [...]

Performance
DNS intermittent delays of 5s #56903

@ Closed mikksoone opened this issue on Dec 6, 2017 • 230 comments

mikksoone commented on Dec 6, 2017 • edited

Is this a BUG REPORT or FEATURE REQUEST?
/kind bug

What happened:
DNS lookup is sometimes taking 5 seconds.

What you expected to happen:
No delays in DNS.

Root cause: May 27, 2018
Patches submitted: Aug 5, 2018
Patches merged: Feb 11, 2019
Reliability

**DNS intermittent delays of 5s #56903**

- **Closed**
- **mikksoone** opened this issue on Dec 6, 2017 • 230 comments

**mikksoone** commented on Dec 6, 2017 • edited

*Is this a BUG REPORT or FEATURE REQUEST?*
/kind bug

*What happened:*
DNS lookup is sometimes taking 5 seconds.

*What you expected to happen:*
No delays in DNS.

---

**First occurrence of bug**

Nov 11, 2010

**Patches merged**

Feb 11, 2019
Compatiblity issues along the way

kube-proxy currently incompatible with `iptables >= 1.8`
#71305

drags opened this issue on Nov 21, 2018 · 75 comments · May be fixed by #82966 or #84420

Ensure iptables tooling does not use the nftables backend

In Linux, nftables is available as a modern replacement for the kernel’s iptables subsystem. The `iptables` tooling can act as a compatibility layer, behaving like iptables but actually configuring nftables. This nftables backend is not compatible with the current kubeadm packages: it causes duplicated firewall rules and breaks `kube-proxy`.

If your system’s `iptables` tooling uses the nftables backend, you will need to switch the `iptables` tooling to 'legacy' mode to avoid these problems. This is the case on at least Debian 10 (Buster), Ubuntu 19.04, Fedora 29 and newer releases of these distributions by default. RHEL 8 does not support switching to legacy mode, and is therefore incompatible with current kubeadm packages.

Debuggability

```bash
# iptables-save -c

*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
[1:10] -A FORWARD -i eth0 -s 172.17.0.0/16 -j DROP
```
Debuggability

Source: reddit.com/r/networkingmemes/comments/8u7jyz/container_networking/
Packet flow

1. alloc skb
2. TC ingress
3. raw PREROUTING
4. conntrack
5. mangle FORWARD
6. FIB lookup
7. nat PREROUTING
8. mangle PREROUTING
9. filter FORWARD
10. mangle POSTROUTING
11. nat POSTROUTING
12. TC egress

host

pod

Source: commons.wikimedia.org/w/index.php?curid=122201
ClusterIP with iptables

```
$ kubectl get svc nginx
NAME     TYPE     CLUSTER-IP     EXTERNAL-IP   PORT(S)
nginx    ClusterIP  3.3.3.3      <none>        80/TCP

$ kubectl get endpoints nginx
NAME     ENDPOINTS
nginx    1.1.1.1:80, 1.1.2.2:80
```

```
-t nat
-A PREROUTING
-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 1.1.2.2/32 -j KUBE-MARK-MASQ
-A KUBE-SEP-NGINX2 -p tcp -m tcp -j DNAT --to-destination 1.1.2.2:80
```
Packet flow

alloc skb → TC ingress

mangle FORWARD
FIB lookup

raw PREROUTING

conntrack

mangle PREROUTING

PREROUTING

mangle FORWARD
mangle POSTROUTING

filter FORWARD

nat PREROUTING

POSTROUTING

POSTROUTING

TC egress

lxc0

eth0

host

pod
Packet flow

Source: commons.wikimedia.org/w/index.php?curid=122201
SEC("to_netdev")

int handle(struct sk_buff *skb) {
    ...
    if (tcp->dport == 80)
        redirect(lxc0);
    return DROP_PACKET;
}

clang -target bpf [...]

foo.o

BPF loader

bpf(BPF_MAP...)

agent

bpf(BPF_PROG_LOAD, ...)

BPF maps

native code

JIT

BPF verifier

lxc0

eth0

userspace

kernelspace
BPF as a radical shift towards full programmability

Freedom to let user tinker with the kernel through BPF programs, but with safety-belt on.

Main use-cases in networking, tracing and security subsystems, e.g. in networking, allows to fully define the forwarding pipeline.

Stable API guarantees as with syscalls. Native speed as with kernel modules. Atomic program updates on live kernel without service disruption. Designed for performance and solving production use-cases.
287 contributors (Jan 2016 to Jan 2020):

- 466 Daniel Borkmann (Cilium; maintainer)
- 290 Andrii Nakryiko (Facebook)
- 279 Alexei Starovoitov (Facebook; maintainer)
- 217 Jakub Kicinski (Facebook, formerly Netronome)
- 173 Yonghong Song (Facebook)
- 168 Martin KaFai Lau (Facebook)
- 159 Stanislav Fomichev (Google)
- 148 Quentin Monnet (Cilium, formerly Netronome)
- 148 John Fastabend (Cilium)
- 118 Jesper Dangaard Brouer (Red Hat)
- [...]

Large-scale users:
BPF in Kubernetes networking and security: enter Cilium

- Datapath implemented in BPF
- Networking
  - Cilium-CNI or chaining on top of most other CNIs
- Kubernetes Services implementation
- Network Policies
  - Identity-based, DNS aware, API aware
- Multicluster, Encryption
- Native Envoy and Istio Integration
  - Transparent Envoy injection (per-node or sidecar)
  - Accelerated proxy redirection, Transparent SSL visibility
- All Open Source at github.com/cilium/cilium
Path towards replacing kube-proxy with BPF in Cilium

$ kubectl -n kube-system delete ds kube-proxy
kube-proxy

1. ClusterIP
   - In-cluster access via virtual IP

2. NodePort
   - Access from outside / inside via node IP + port

3. ExternalIP
   - Access from outside via external IP

4. LoadBalancer
   - Access from outside via external LB
ClusterIP (pod to pod) in Cilium

1. Lookup dst in SVC map
2. If found:
   a. Select EP
   b. DNAT
   c. Create SVC CT
   d. Create Egress CT

1. Lookup Egress CT
2. If found:
   a. Rev-DNAT
   b. Redirect to lxc0

<table>
<thead>
<tr>
<th>SVC IP</th>
<th>Port</th>
<th>NR</th>
<th>ID</th>
<th>EID</th>
<th>Endpoint</th>
<th>IP</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.3.3</td>
<td>80</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1.1.1.1</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>3.3.3.3</td>
<td>80</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>1.1.1.2</td>
<td>80</td>
<td></td>
</tr>
</tbody>
</table>
Cilium service maps

kube-apiserver

```
apiVersion: v1
category: Service
metadata:
  name: nginx
spec:
  selector:
    app: nginx
  ports:
  - protocol: TCP
    port: 80
  clusterIP: 3.3.3.3
```

```
apiVersion: v1
category: Endpoints
metadata:
  name: nginx
spec:
  subsets:
  - addresses:
    - ip: 1.1.1.1
  ports:
  - port: 80
    protocol: TCP
```

```
bpf_map_update_element(...)
```

eBPF SVC hash map

<table>
<thead>
<tr>
<th>SVC IP</th>
<th>Port</th>
<th>NR</th>
<th>ID</th>
<th>EID</th>
<th>Endpoint IP</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.3.3</td>
<td>80</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1.1.1.1</td>
<td>80</td>
</tr>
<tr>
<td>3.3.3.3</td>
<td>80</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>1.1.1.2</td>
<td>80</td>
</tr>
</tbody>
</table>
ClusterIP (host or pod to pod) in Cilium

1. Lookup dst in SVC map
2. If found:
   a. Change dst addr and port in socket

```
import "net/http"
func main() {
    r, err := http.Get("3.3.3.3")
    ...
}
```
ClusterIP (host or pod to pod) in Cilium

import "net/http"
func main() {
    r, err := http.Get("nginx")
    ...}

1. Lookup dst in SVC map
2. If found:
   a. Change dst addr and port in socket
   b. Create rev NAT entry

1. Lookup src in rev NAT map
2. If found:
   a. Change src addr and port

TCP
UDP
sendmsg()
recvmsg()
NodePort with service endpoint on local node in Cilium

1. SVC lookup & DNAT
2. Is endpoint local?
   2.1. Redirect to lxc0

1. rev-DNAT xlation
   2. Redirect to eth0

10.100.1.1:60000 -> 192.168.0.1:31000

Node A

1.1.1.1
eth0

192.168.0.1

client
NodePort with service endpoint on remote node in Cilium

1. SVC lookup & DNAT
2. Is endpoint remote?
   2.1. eBPF SNAT
   2.2. Redirect

192.168.0.1:60000 -> 1.1.1.1:80

10.100.1.1:60000 -> 192.168.0.1:31000
NodePort with service endpoint on remote node in Cilium

192.168.0.1:31000 -> 10.100.1.1:60000
NodePort externalTrafficPolicy=Local

10.100.1.1:60000 -> 192.168.0.1:31000
NodePort (DSR) in Cilium

1. SVC lookup & DNAT
2. Is endpoint remote?
   2.1. Append SVC addr into IP hdr
   2.2. Redirect
NodePort (DSR) in Cilium

1.1.2.1
eth0
lxc0

192.168.0.1
eth0
Node A

192.168.0.1:31000 -> 10.100.1.1:60000

client

1.1.1.1
eth0
lxc0

192.168.0.2
eth0
Node B

1.rev-DNAT xlation
2.Redirect
Performance (lower is better)

TCP_CRR to direct backend via NodePort latency (μsec per tx)

Number of services

BPF  | ipvs  | iptables
Performance (lower is better)

TCP_RR to remote backend via NodePort latency (μsec per tx)

Number of services:
- 1
- 100
- 1000
- 2000
- 2768

Graph categories:
- BPF DSR
- BPF
- ipvs
- iptables
WIP for Cilium: XDP for hop to remote node (DSR, SNAT)

Native XDP finally supported by all 3 major cloud providers.

tl;dr

Performance
- Better performance and latency over kube-proxy (ipvs and iptables)
- Fast service updates

Reliability
- Less LOC in datapath
- No need to wait for a new kernel release to fix a bug

Visibility
- Better tooling for introspection and troubleshooting

Compatibility
- No more exec iptables

Customization
- Ability to change datapath behaviour on the fly
- Fully integrated with rest of Cilium BPF datapath features
Want to liberate yourself from kube-proxy?

Howto: https://cilium.link/kubeproxy-free
Code: https://github.com/cilium/cilium
Slack: https://cilium.io/slack