Exploration of Linux Container Network Monitoring and Visualization

Alban Crequy

https://goo.gl/iDL8te
Alban Crequy

- Worked on the rkt container run-time
- Contributed to systemd

https://github.com/alban
Berlin-based software company building **foundational** Linux technologies

Some examples of what we work on...

- rkt
- systemd
- OSTree
Find out more about us...

**Blog:** [https://kinvolk.io/blog](https://kinvolk.io/blog)

**Github:** [https://github.com/kinvolk](https://github.com/kinvolk)

**Twitter:** [https://twitter.com/kinvolkio](https://twitter.com/kinvolkio)

**Email:** hello@kinvolk.io
Plan

- First use-case: visualizing tcp connections
  - Microservices application with containers: Weave Socks
  - CoreOS Linux, Kubernetes, Weave Scope
- Using /proc & conntrack
  - Limitations
  - proc connector, eBPF & kprobes
- Next use cases:
  - L7, HTTP: eBPF & kprobes
  - Simulating degraded networks with traffic control
The demo application
microservices-demo

https://github.com/microservices-demo/microservices-demo
Some micro-services
Orchestrating containers With Kubernetes
Kubernetes Replica Sets

- Kubernetes node 1
  - front-end
  - catalogue
- Kubernetes node 2
  - front-end
  - orders
- Kubernetes node 3
  - catalogue
  - orders
Kubernetes Services

- Kubernetes node 1
- Kubernetes node 2
- Kubernetes node 3

front-end

orders

orders service
Kubernetes Services

Proxying the traffic from the virtual service IP to a Kubernetes pod

Several implementations possible:

- Userspace proxy in kube-proxy
- Iptables rules (Destination NAT) installed by kube-proxy
- Cilium implements a load balancer based on eBPF (tc level)
Weave Scope
Weave Scope demo
procfs
procfs files

- /proc/$PID
- /proc/$PID/ns/net network namespace
- /proc/$PID/fd/ file descriptors
- /proc/$PID/net/tcp tcp connections
procfs files

```
$ nc 85.239.127.90 80

$ ls -l /proc/$(pidof nc)/ns/net
lrwxrwxrwx. 1 alban alban 0 Oct 2 16:42 /proc/25343/ns/net -> 'net:4026531969'

$ ls -l /proc/$(pidof nc)/fd/
total 0
lrwx------. 1 alban alban 64 Oct 2 16:35 0 -> /dev/pts/2
lrwx------. 1 alban alban 64 Oct 2 16:35 2 -> /dev/pts/2
lrwx------. 1 alban alban 64 Oct 2 16:35 3 -> 'socket:[8707290]'

$ cat /proc/$(pidof nc)/net/tcp
    sl  local_address  rem_address  st  tx_queue  rx_queue  tr tm->when retransmt  uid  timeout inode
25: 0C00A8C0:1984A 5A7FEF55:0050 01 00000000:00000000 00:00000000 00000000 1000 0 8707290 1 ffff88026285e300 23 0 0 10 -1

$ printf '%02X' 90 127 239 85 ; echo -n ':' ; printf '%04X' 80 ; echo
5A7FEF55:0050
```
procfs limitations

- No notifications
- Need to read procfs for
  - new processes
  - new network namespaces
  - new sockets
  - every second?
- CPU intensive for systems with high number of processes
- Missing short-lived connections
- Issues with packet modifications (e.g. DNAT)
Packet modifications

Protocol layer

Network layer

Link layer

Local process

NAT

Traffic control, egress

Kubernetes node 1

Socket lookup

Traffic control, ingress

Kubernetes node 2
Netlink
Netlink sockets

socket(AF_NETLINK, SOCK_RAW, NETLINK_...);

Several Netlink sockets:

- NETLINK_ROUTE
- NETLINK_INET_DIAG
- NETLINK_SELINUX
- NETLINK_CONNECTOR
- NETLINK_NETFILTER
- ...
conntrack -E

- Use NETLINK_NETFILTER sockets to subscribe to Conntrack events from the kernel
- Is aware of NAT rewritings
conntrack limitations

- Conntrack events don't include:
  - Process ID
  - Network namespace ID
    - Conntrack zones included but not necessary used by container run-times
- So harvesting procfs regularly still necessary
Other kind of Netlink sockets?
NETLINK_INET_DIAG

socket(AF_NETLINK, SOCK_RAW, NETLINK_INET_DIAG);

- Fetch information about sockets
  - Used by ss (“another utility to investigate sockets”)
  - Basic bytecode to filter the sockets (e.g. “INET_DIAG_BC_JMP”)
- But no notification mechanism
  - Patch “sock_diag: notify packet socket creation/deletion” (2013) rejected
Kernel Connector

socket(AF_NETLINK, SOCK_RAW, NETLINK_CONNECTOR);

Several Kernel Connector agents:

- Device mapper
- HyperV
- Proc connector
Proc connector

bind(sockfd, ...CN_IDX_PROC...);
sendmsg(sockfd, ...PROC_CN_MCAST_LISTEN...)

- Since Linux v2.6.15 (January 2006)

Notifications for:

- fork
- exec
- exit
Proc connector

Missing:

- network namespace
  - RFC patch “proc connector: add namespace events” last month
    https://lkml.org/lkml/2016/9/8/588
- Sockets

So harvesting procfs regularly still necessary
Proc connector

demo
BPF
Classic BPF (cBPF)

- `setsockopt(sockfd, SOL_SOCKET, SO_ATTACH_FILTER, &bpf, sizeof(bpf));`
- `recvfrom()`
Extended BPF (or eBPF)

Program type:

- BPF_PROG_TYPE_SOCKET_FILTER
- BPF_PROG_TYPE_KPROBE
- BPF_PROG_TYPE_SCHED_CLS
- BPF_PROG_TYPE_SCHED_ACT
- BPF_PROG_TYPE_TRACEPOINT (Linux >= 4.7)
- BPF_PROG_TYPE_XDP
eBPF classifier for qdiscs

```c
if (skb->protocol...) return TC_H_MAKE(TC_H_ROOT, mark);
```

userspace

```
compilation clang... -march=bpf
```

kernel

```
BPF_JMP... BPF_LD... BPF_RET...
```

upload

- bpf()
- Netlink

x86_64 code

JIT compilation
eBPF maps

- Keep context between calls
- Report statistics to userspace
Tracepoints with eBPF

- BPF_PROG_TYPE_TRACEPOINT since Linux 4.7
- Find the list of tracepoints in /sys/kernel/debug/tracing/events
- Stable API
- But limited tracepoints

```
# cat /sys/kernel/debug/tracing/events/skb/kfree_skb/format
name: kfree_skb
ID: 1135
format:
    field:unsigned short common_type; offset:0; size:2; signed:0;
    field:unsigned char common_flags; offset:2; size:1; signed:0;
    field:unsigned char common_preempt_count; offset:3; size:1; signed:0;
    field:int common_pid; offset:4; size:4; signed:1;

    field:void * skbaddr; offset:8; size:8; signed:0;
    field:void * location; offset:16; size:8; signed:0;
    field:unsigned short protocol; offset:24; size:2; signed:0;

print fmt: "skbaddr=%p protocol=%u location=%p", REC->skbaddr, REC->protocol, REC->location
```
kprobes with eBPF

- BPF_PROG_TYPE_KPROBE since Linux 4.1
- No ABI guarantees
- Probe any kernel function
Socket events with kprobe / eBPF

- BPF Compiler Collection (BCC)
  - bcc/examples/tracing/tcpv4connect.py
  - Iago’s tcp4tracer (WIP)
    - Get connection tuple, pid, netns
    - tcp_v4_connect
    - tcp_close
    - inet_csk_accept

<table>
<thead>
<tr>
<th>TYPE</th>
<th>PID</th>
<th>COMM</th>
<th>SADDR</th>
<th>DADDR</th>
<th>SPORT</th>
<th>DPORT</th>
<th>NETNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>connect</td>
<td>7736</td>
<td>nc</td>
<td>192.168.35.25</td>
<td>185.46.139.24</td>
<td>45426</td>
<td>80</td>
<td>4026531969</td>
</tr>
<tr>
<td>close</td>
<td>7736</td>
<td>nc</td>
<td>192.168.35.25</td>
<td>185.46.139.24</td>
<td>45426</td>
<td>80</td>
<td>4026531969</td>
</tr>
</tbody>
</table>
Packet modifications

Kubernetes node 1

Protocol layer → Network layer → Link layer

Local process
NAT
Traffic control, egress

Kubernetes node 2

Local process
Socket lookup
Traffic control, ingress

packet
tcp4tracer & NAT

- The connection tuple from the process’ point of view is not enough
  - NAT
  - Kubernetes Services
- Iago’s tcp4tracer (WIP)
  - nf_nat_ipv4_manip_pkt
  - nf_nat_tcp_manip_pkt

<table>
<thead>
<tr>
<th>TYPE</th>
<th>PID</th>
<th>COMM</th>
<th>SADDR</th>
<th>DADDR</th>
<th>SPORT</th>
<th>DPORT</th>
<th>NETNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>connect</td>
<td>19727</td>
<td>nc</td>
<td>172.16.28.11</td>
<td>185.46.139.27</td>
<td>55366</td>
<td>80</td>
<td>4026532470</td>
</tr>
<tr>
<td>tcp_nat</td>
<td>19727</td>
<td>nc</td>
<td>0.0.0.0</td>
<td>0.0.0.0</td>
<td>55366</td>
<td>80</td>
<td>0</td>
</tr>
<tr>
<td>ip_nat</td>
<td>19727</td>
<td>nc</td>
<td>192.168.35.25</td>
<td>185.46.139.27</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
More metrics
Weave Scope architecture

- Scope Probe
- Scope App
- Scope Probe
- Firefox
- Kubernetes node 1
- Kubernetes node 2
Weave Scope plugins

Kubernetes node 1

Scope Probe

Plugin

Plugin

Kubernetes node 2

Scope Probe

Scope Probe

Plugin

Plugin

Firefox
HTTP requests plugin

- Number of HTTP requests per second
- Without instrumenting the application
- eBPF kprobe on skb_copy_datagram_iter

**Diagram:**

**HTTP client**

- `sendmsg()`

**HTTP server**

- `recvfrom()`
- `GET / HTTP/1.1`
- `skb_copy_datagram_iter()` copies the skb into the iovec
HTTP responses plugin

- Number of HTTP responses by category (404, etc.)
- Without instrumenting the application
- eBPF kprobe on skb_copy_datagram_from_iter
- Using an eBPF map to track the context between kprobe & kretprobe

Userspace

recvfrom()

Kernel

sendmsg()

HTTP/1.0 200 OK

skb_copy_datagram_from_iter() copies the iovec into the skb
Testing degraded networks
Traffic control, why?

- fair distribution of bandwidth
- reserve bandwidth to specific applications
- avoid bufferbloat
Queuing disciplines (qdisc)

- Network scheduling algorithm
  - which packet to emit next?
  - when?
- Configurable at run-time:
  - /sbin/tc
  - Netlink
- Default on new network interfaces: sysctl net.core.default_qdisc
Stochastic Fairness Queueing (sfq)

- eth0
- FIFO 0
- FIFO 1
- FIFO n
- round robin
- THE INTERNET
Demo

Reproduce this demo yourself: https://github.com/kinvolk/demo
Network emulator (netem)

- etho
- bandwidth
- latency
- packet loss
- corrupt
- ...
Testing with containers

Testing framework

configure “netem” qdiscs: bandwidth, latency, packet drop...

container 1

etho

container 2

etho
Add latency on a specific connection

- payment
- catalogue
- orders-db
- orders
- front-end
- Firefox

latency=100ms
How to define classes of traffic

dest_ip=10.0.4.*  dest_ip=10.0.5.*  other

etho  interface

netem  latency=100ms
u32: filter on content

interface

root qdisc (type = HTB)

root class (type = HTB)

leaf qdiscs (type = netem)

leaf classes (type = HTB)

filters (type=u32)

ip=10.0.4.*

ip=10.0.5.*

other

latency=10ms
Filtering with cBPF/eBPF

if (skb->protocol...) 
return TC_H_MAKE(TC_H_ROOT, mark);

BPF

compilation
clang... -march=bpf

upload in the kernel:
- bpf()
- Netlink

x86_64 code
JIT compilation

netem

kernel

userspace

eth0
eBPF maps

eBPF map

tc

userspace

kernel

eth0

BPF

netem

x86_64 code

netem
Questions?

The slides: https://goo.gl/iDL8te