Packet mark in a Cloud Native world

Joe Stringer

Cilium.io


Google
the internet is held together with
the internet is held together with duct tape Remove
the internet is held together with bubble gum Remove
the internet is held together with baling wire Remove
the internet is held together with popsicle sticks Remove
the internet is held together with pixie dust Remove
the internet is held together with prayers Remove
the internet is held together with skb->mark Remove

Overview

Mark of the 

- `fw_mark`
- `skb_mark`
- `ct_mark`
- `SO_MARK`
- `xfrm_mark`
- `pkt_mark`

```
struct sk_buff {
    ...
    __u32 mark;
    ...
}
```

So what does the mark represent?

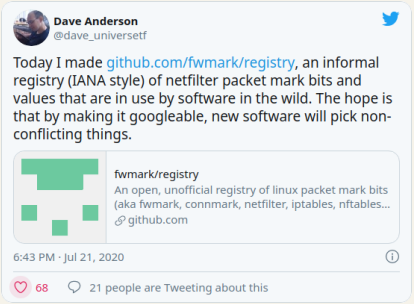
- Nothing..

So what does the mark represent?

- Nothing..
- Anything!

So what does the mark represent?

- Nothing..
- Anything!
- **MAGIC.** ✨



https://twitter.com/dave_universetf/status/1285752332135788544

cilium Star 6092

Enterprise Blog Slack Documentation

eBPF-based Networking, Observability, and Security

Follow @ciliumproject

Join Our Slack

Networking	Observability	Security
Highly Scalable Kubernetes CNI	Identity-aware Network Visibility	Advanced Network Policy
Kube-proxy Load Balancer Replacement	Network Metrics + Troubleshooting	Security Forensics + Audit
Multi-cluster Connectivity	API-aware Network Observability	Transparent Encryption

Cloud Native networking

Methodology

1. Look at CNCF landscape[1]
2. Find the project on GitHub
3. Search for $mark_name
4. ???
5. Knowledge!

[1] https://landscape.cncf.io/category=cloud-native-network&format=card-mode&grouping=category

Use cases

Network policy

- 1 bit, two variations:
  - 1 bit -> drop [2]
  - 1 bit -> allow
- Store complex path through rules into mark
- Typically netfilter -> netfilter

[2] Kubernetes default

Transparent encryption

- 2+ bits
  - 1 bit encrypt, 1 bit decrypt
  - Variation: key selector
- { eBPF, netfilter } -> xfrm

Virtual IP services

- 1+ bits, request DNAT
  - 1 bit: route towards bridge for DNAT
  - 30 bits representing hashed 3-tuple
- { eBPF, netfilter } -> routing -> netfilter
- OVS -> routing -> OVS

IP masquerade

- 1+ bits, request SNAT
  - Variation: 1 bit, Skip SNAT
  - Variation: 32 bits for source address selection
- Connection may not originate on the node
- {eBPF, OVS, netfilter} -> netfilter

Multi-homing

- 1 bit, two variations:
  - Reply via primary device
    - Default: Pod communicates via secondary device
    - Inbound connections must reply via primary device
    - Store & restore in connmark
  - Route via management interface
- { socket, netfilter } -> routing

Application identity

- Variable bits
  - 4 bit pattern: “local” traffic
  - 16+ bits: Carry Identity to destination
    - Policy routing
    - Portmap plugin
- { eBPF, netfilter } -> routing -> eBPF

Service proxy

- 1+ bits depending on context
  - 1 bit, route locally
  - 16 bit tproxy port towards proxy
  - 16+ bit Identity from proxy
- eBPF -> { netfilter, routing }
- netfilter -> routing
- socket -> { eBPF, netfilter },

Observations

Marking your territory

- Bitwise usage
  - Simpler interoperability
- Full-mark
  - More values to work with
  - Most usage doesn't make use of this

A tiny bit of overload

- Use every feature: 100+ bits
  - ...but there's only 32 bits to play with?
- Mitigation: Encode meaning in bit range
  - Use [0x0000..0x000F] rather than bits in 0xFFFF
- Mitigation: Overload bits on different paths
  - Ingress / Egress
  - Make semantics dependent on packet fields

One does not simply understand skb mark

- Required reading: network stack diagram
- Distinct bits do not guarantee integration
  - skb, conn matches may steer packets
- Fun: replies disappear
- Proxies: Double the connections, double the fun

Evaluation

Properties of skb mark

- Powerful mechanism for cross-subsystem programming
- Frequent uncertainty whether bits are OK to use
- Can be a crutch
- When you run out of bits, the “fun” starts

Interoperability

- Driven by common deployment scenarios
- The clearer responsibility assignment you have, the better
- Not free (in effort or in complexity)

Mitigating conflicts

- "If only I had more bits..."
- How can we get more?
  - Consolidate subsystem usage
  - Extend the kernel

Summary

- Cloud native use cases
- Common themes
- Challenges for interoperability

Cilium

https://cilium.io
https://cilium.io/slack
https://github.com/cilium/cilium
https://twitter.com/ciliumproject

Mark registry

https://github.com/fwmark/registry