- Update on Cilium with tcx & netkit
- Revamping global socket iterator

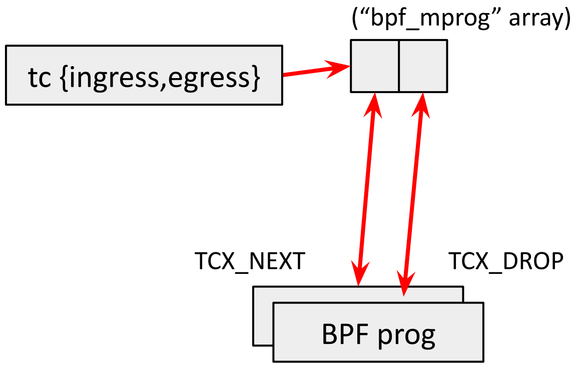
Daniel Borkmann (Cisco)

LSF/MM/BPF 2024

tcx: What’s done

tcx datapath infra was merged and released with 6.6 kernel

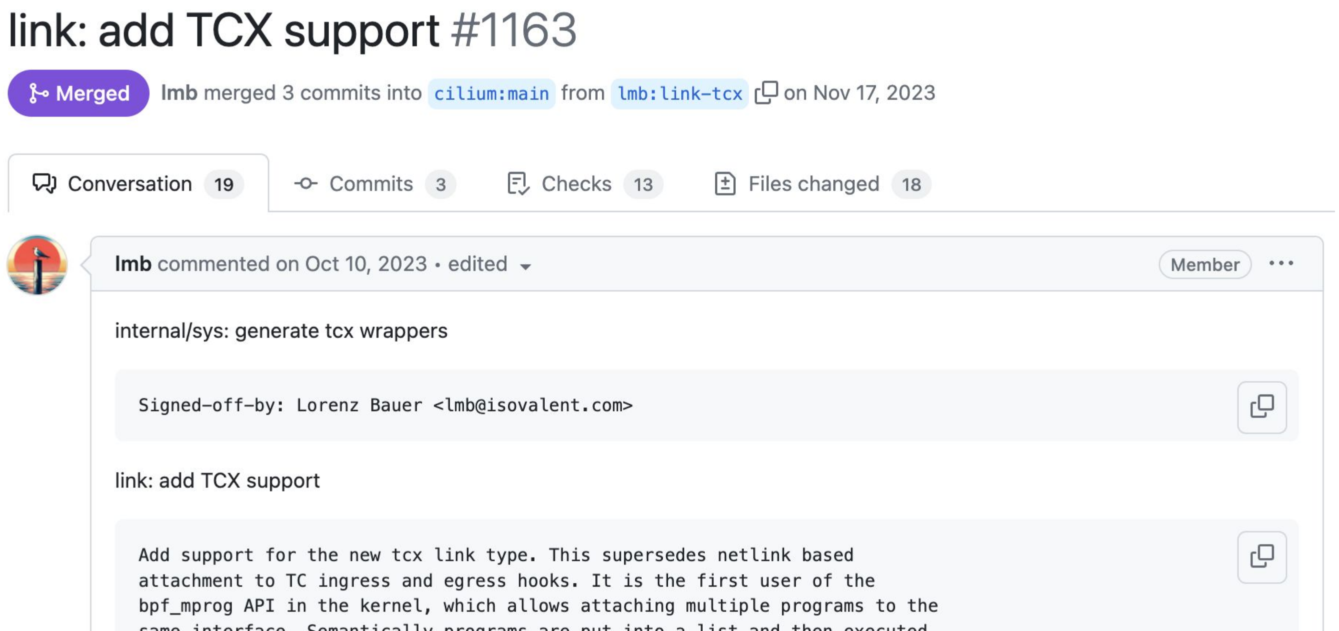
```
static __always_inline struct sk_buff *
sch_handle_ingress(struct sk_buff *skb, struct packet_type **pt_prev, int *ret,
                   struct net_device *orig_dev, bool *another)
{
        struct bpf_mprog_entry *entry = rcu_dereference_bh(skb->dev->tcx_ingress);
        enum skb_drop_reason drop_reason = SKB_DROP_REASON_TC_INGRESS;
        int sch_ret;

        if (!entry)
                return skb;
        if (*pt_prev) {
                *ret = deliver_skb(skb, *pt_prev, orig_dev);
                *pt_prev = NULL;
        }

        qdisc_skb_cb(skb)->pkt_len = skb->len;
        tcx_set_ingress(skb, true);

        if (static_branch_unlikely(&tcx_needed_key)) {
                sch_ret = tcx_run(entry, skb, true);
                if (sch_ret != TC_ACT_UNSPEC)
                        goto ingress_verdict;
        }
        sch_ret = tc_run(tcx_entry(entry), skb, &drop_reason);
ingress_verdict:
        switch (sch_ret) {
        case TC_ACT_REDIRECT:
                /* skb_mac_header check was done by BPF, so we can safely
```



tcx: What’s done

cilium/ebpf support was merged (thanks to Lorenz!)

- Goal: BPF program management for direct or link-based attachment



link: add TCX support #1163

Merged lmb merged 3 commits into cilium:main from lmb:link-tcx on Nov 17, 2023

Conversation 19 | Commits 3 | Checks 13 | Files changed 18

lmb commented on Oct 10, 2023 • edited · Member

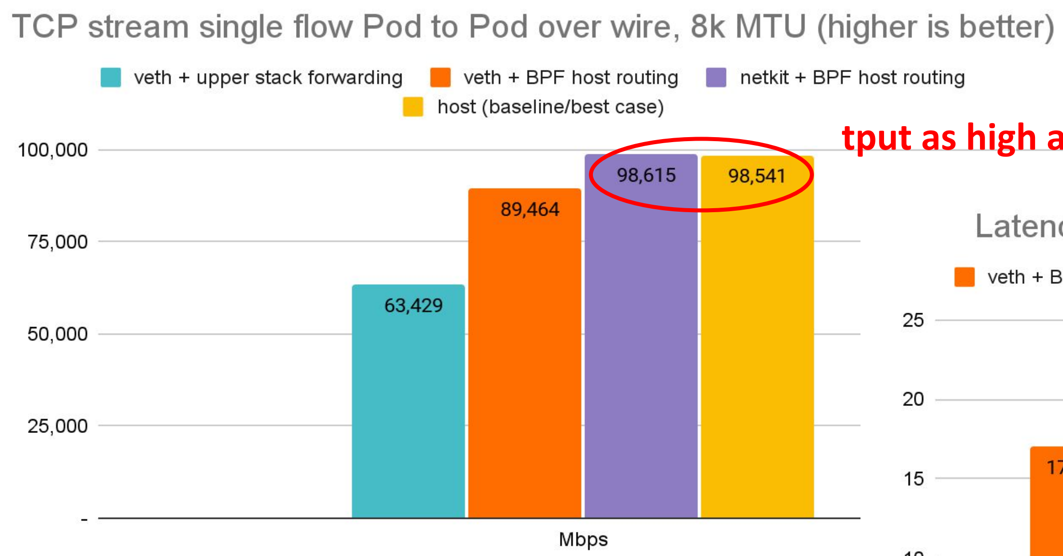
internal/sys: generate tcx wrappers

```
Signed-off-by: Lorenz Bauer <lmb@isovalent.com>
```

link: add TCX support

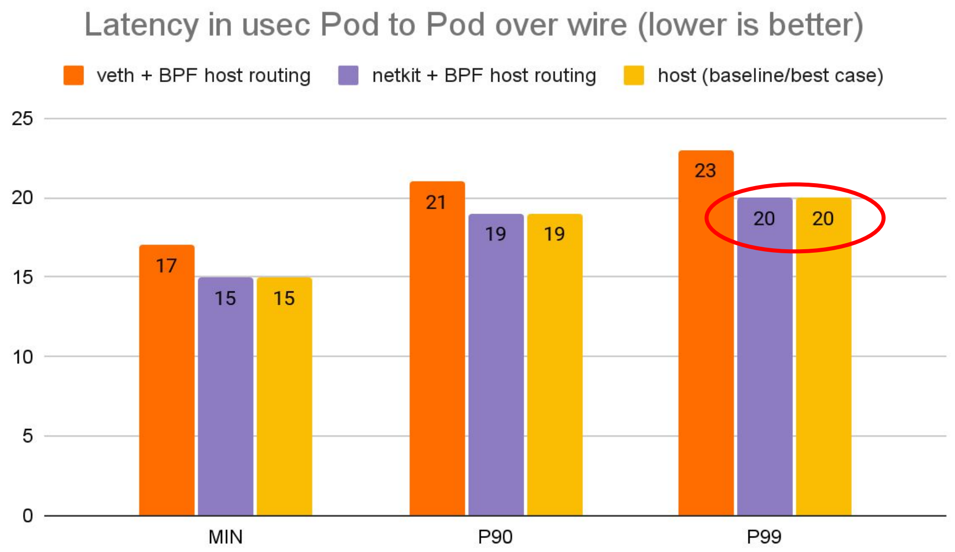
```
Add support for the new tcx link type. This supersedes netlink based
attachment to TC ingress and egress hooks. It is the first user of the
bpf_mprog API in the kernel, which allows attaching multiple programs to the
```

netkit: What's done



tput as high as host

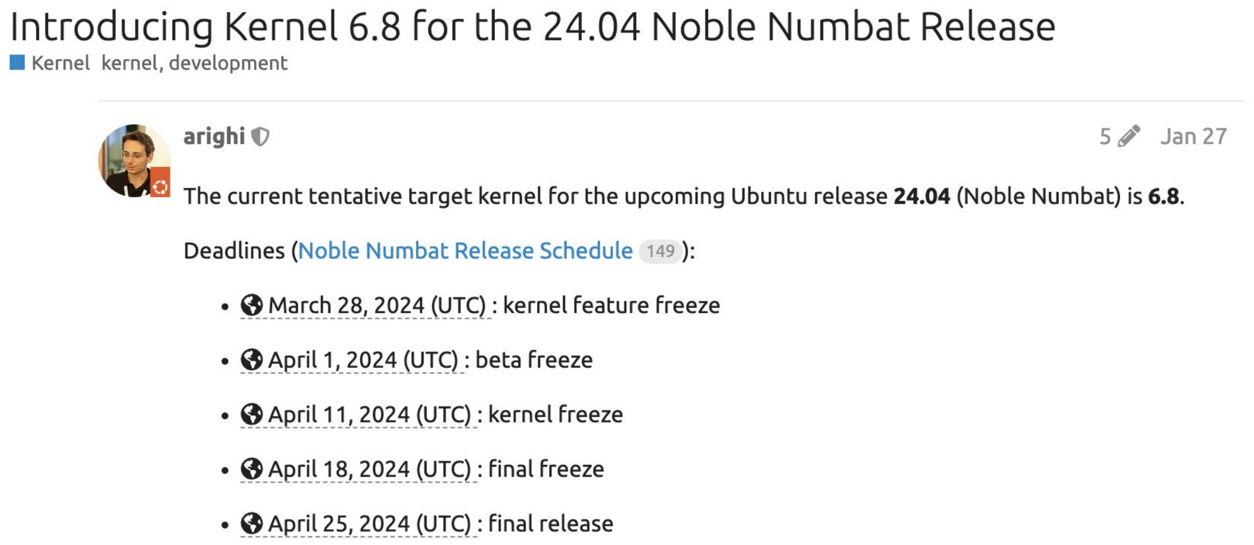
latency as low as host



netkit: What's done

netkit driver was merged and released with 6.7 kernel

CONFIG_NETKIT=y (bool) is set by default in latest **Ubuntu 24.04 LTS** !



Introducing Kernel 6.8 for the 24.04 Noble Numbat Release

Kernel kernel, development

arighi 5 Jan 27

The current tentative target kernel for the upcoming Ubuntu release **24.04** (Noble Numbat) is **6.8**.

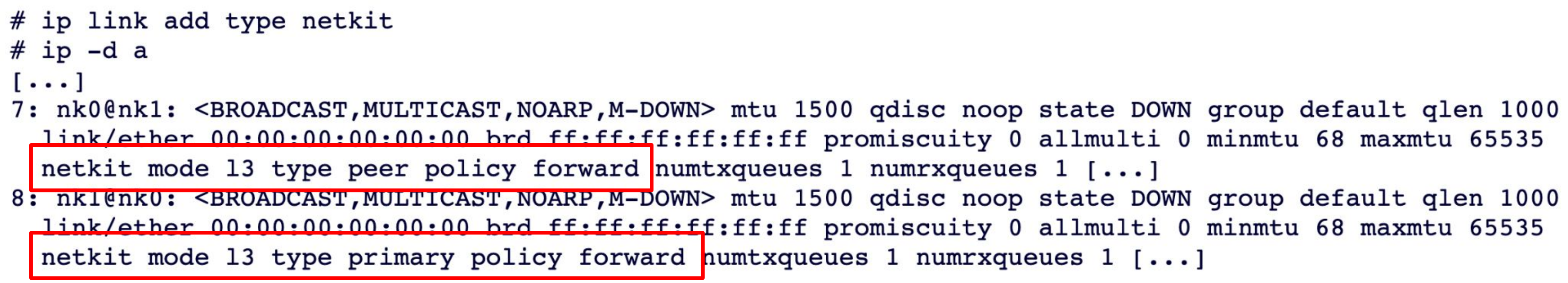
Deadlines (Noble Numbat Release Schedule 149):

- March 28, 2024 (UTC) : kernel feature freeze
- April 1, 2024 (UTC) : beta freeze
- April 11, 2024 (UTC) : kernel freeze
- April 18, 2024 (UTC) : final freeze
- April 25, 2024 (UTC) : final release

netkit: What’s done

iproute2 support was merged and released with iproute2 v6.8.0

- Goal: Basic device setup and introspection support



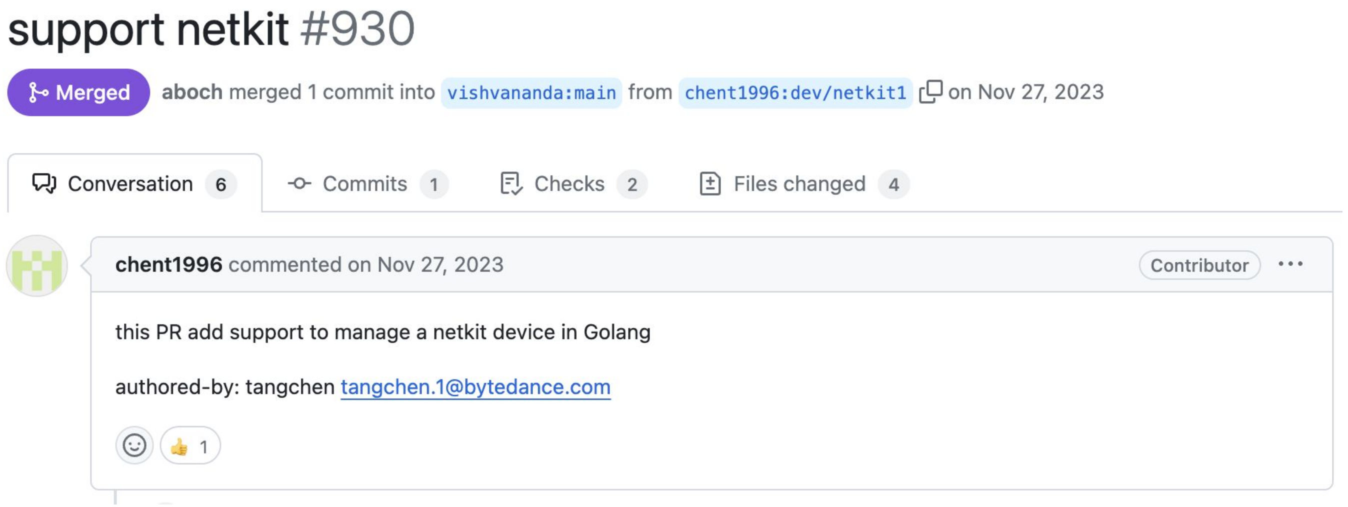
```
# ip link add type netkit
# ip -d a
[...]
7: nk0@nk1: <BROADCAST,MULTICAST,NOARP,M-DOWN> mtu 1500 qdisc noop state DOWN group default qlen 1000
  link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff promiscuity 0 allmulti 0 minmtu 68 maxmtu 65535
  netkit mode l3 type peer policy forward numtxqueues 1 numrxqueues 1 [...]
8: nk1@nk0: <BROADCAST,MULTICAST,NOARP,M-DOWN> mtu 1500 qdisc noop state DOWN group default qlen 1000
  link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff promiscuity 0 allmulti 0 minmtu 68 maxmtu 65535
  netkit mode l3 type primary policy forward numtxqueues 1 numrxqueues 1 [...]
```

- Support base setup and delegate BPF program management to applications (via libbpf, ebpf-go)

netkit: What's done

vishvananda/netlink support was merged (thanks to Bytedance!)

- Goal: Native iproute2 equivalent for Go, that is, basic device setup and introspection support



support netkit #930

aboch merged 1 commit into vishvananda:main from chent1996:dev/netkit1 on Nov 27, 2023

chent1996 commented on Nov 27, 2023

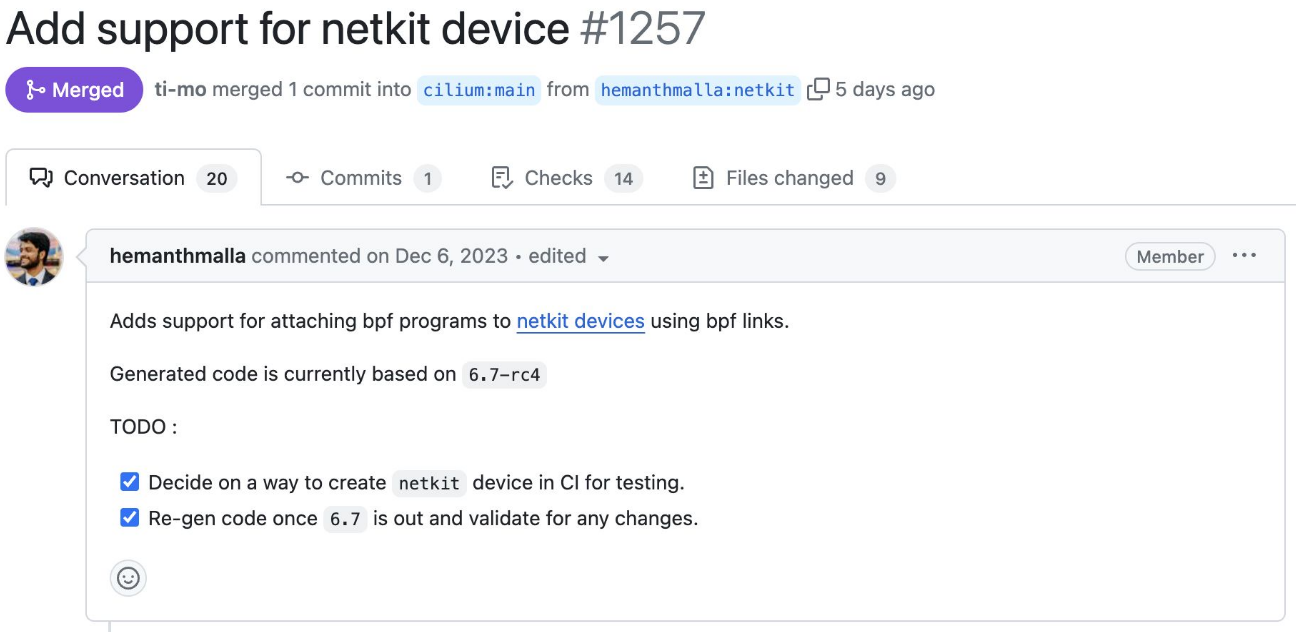
this PR add support to manage a netkit device in Golang

authored-by: tangchen tangchen.1@bytedance.com

netkit: What’s done

cilium/ebpf support was merged (thanks to Datadog!)

- Goal: BPF program management for direct or link-based attachment, same look & feel as tcx



Add support for netkit device #1257

Merged ti-mo merged 1 commit into `cilium:main` from `hemanthmalla:netkit` 5 days ago

Conversation 20 Commits 1 Checks 14 Files changed 9

hemanthmalla commented on Dec 6, 2023 • edited Member

Adds support for attaching bpf programs to netkit devices using bpf links.

Generated code is currently based on `6.7-rc4`

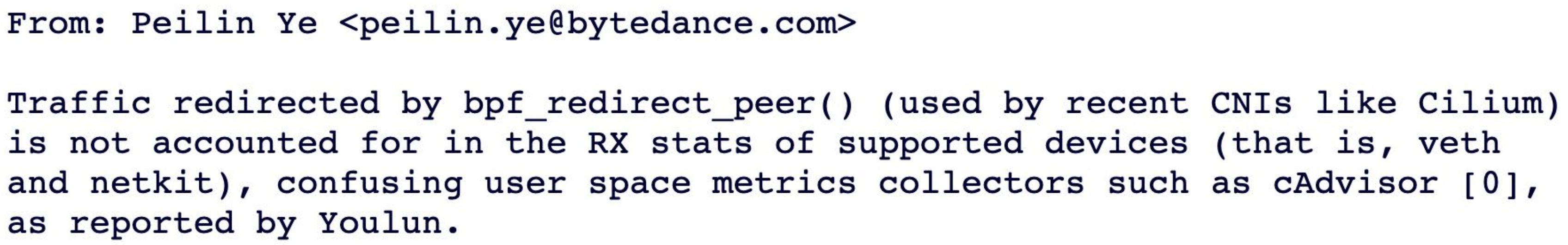
TODO :

- [x] Decide on a way to create `netkit` device in CI for testing.
- [x] Re-gen code once `6.7` is out and validate for any changes.

netkit: What’s done

Fixing networking stats for netkit in general and for peer-redirection

- Goal: Proper network stats accounting for cAdvisor for netkit and veth



```
From: Peilin Ye <peilin.ye@bytedance.com>

Traffic redirected by bpf_redirect_peer() (used by recent CNIs like Cilium)
is not accounted for in the RX stats of supported devices (that is, veth
and netkit), confusing user space metrics collectors such as cAdvisor [0],
as reported by Youlun.
```

- Fix is calling dev_sw_netstats_rx_add() in skb_do_redirect() and move netkit & veth to dev->tstats
- Guard if drivers implementing ndo_get_peer_dev and do not use dev->tstats
- Suggestion from Jakub Kicinski to move {l,t,d}stats allocation into net core

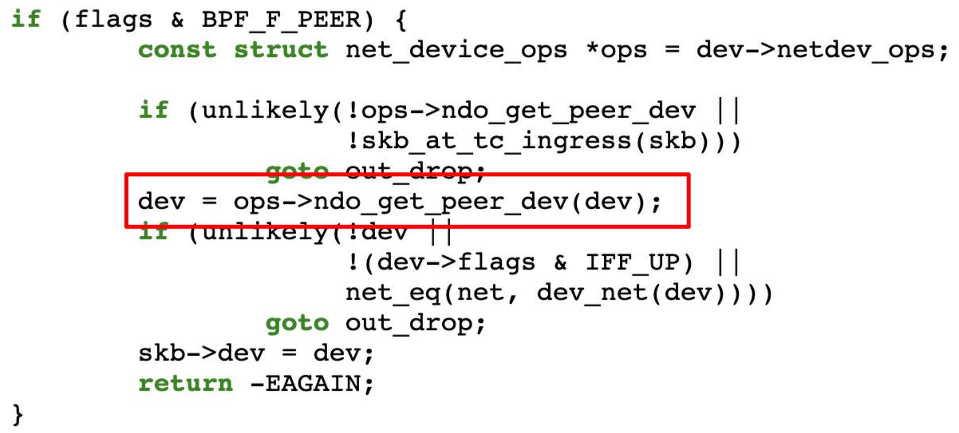
Patches: bpf_redirect_peer fixes, v3 (Daniel Borkmann, Jakub Kicinski, Nikolay Aleksandrov, Peilin Ye)

netkit: What’s done

Fast-path optimising getting peer pointer from struct net_device

- Goal: Get rid of ndo_get_peer_dev entirely and add peer pointer to net_device (suggestion from Jakub Kicinski)

- The latter is only implemented by veth and netkit
- Helps performance for ingress direction due to the current indirect call in skb_do_redirect()



```
if (flags & BPF_F_PEER) {
        const struct net_device_ops *ops = dev->netdev_ops;

        if (unlikely(!ops->ndo_get_peer_dev ||
                     !skb_at_tc_ingress(skb)))
                goto out_drop;
        dev = ops->ndo_get_peer_dev(dev);
        if (unlikely(!dev ||
                     !(dev->flags & IFF_UP) ||
                     net_eq(net, dev_net(dev))))
                goto out_drop;
        skb->dev = dev;
        return -EAGAIN;
}
```

netkit: What’s done

Fast-path optimising getting peer pointer from struct net_device

- Goal: Get rid of ndo_get_peer_dev entirely and add peer pointer to net_device (suggestion from Jakub Kicinski)

- The latter is only implemented by veth and netkit
- Helps performance for ingress direction due to the current indirect call in skb_do_redirect()

```
if (flags & BPF_F_PEER) {
        if (unlikely(!skb_at_tc_ingress(skb)))
                goto out_drop;
        dev = skb_get_peer_dev(dev);
        if (unlikely(!dev ||
                     !(dev->flags & IFF_UP) ||
                     net_eq(net, dev_net(dev))))
                goto out_drop;
        skb->dev = dev;
        dev_sw_netstats_rx_add(dev, skb->len);
        return -EAGAIN;
}
```

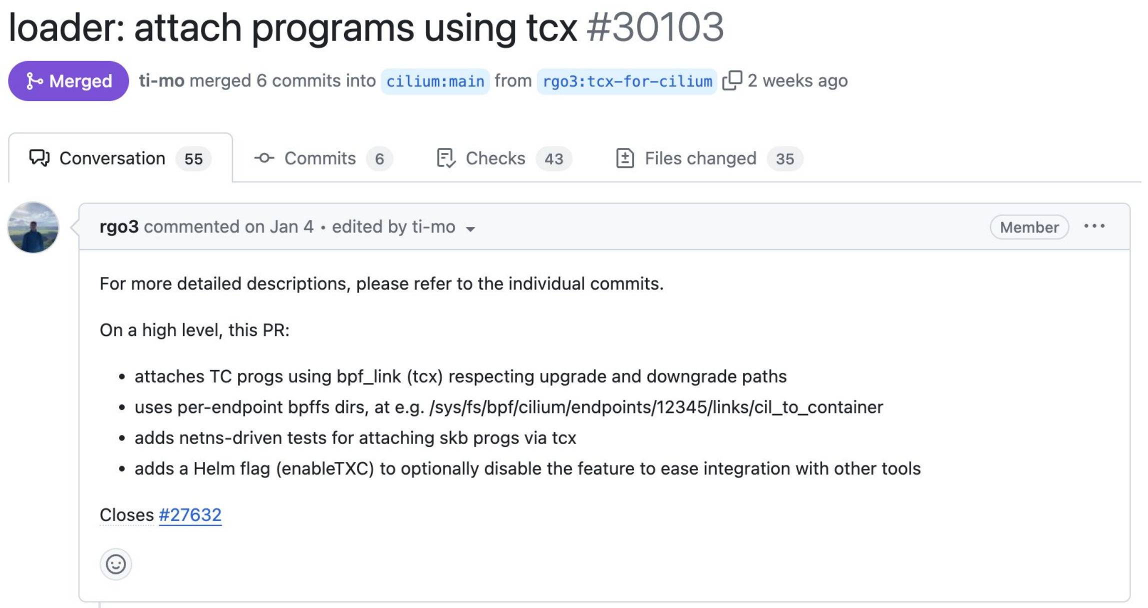
```
static struct net_device *skb_get_peer_dev(struct net_device *dev)
{
        const struct net_device_ops *ops = dev->netdev_ops;

        if (likely(ops->ndo_get_peer_dev))
                return INDIRECT_CALL_1(ops->ndo_get_peer_dev,
                                       netkit_peer_dev, dev);
        return NULL;
}
```

Possible given **CONFIG_NETKIT** is **bool**

Cilium & tcx: What's done

Integration and merge for Cilium 1.16 with tcx complete

- Enabled by default for 6.6+ kernels, opt-out to old style tc possible
- Now all Cilium attachments are BPF link based (XDP, tcx, cgroups)!



loader: attach programs using tcx #30103

Merged ti-mo merged 6 commits into `cilium:main` from `rgo3:tcx-for-cilium` 2 weeks ago

rgo3 commented on Jan 4 • edited by ti-mo

For more detailed descriptions, please refer to the individual commits.

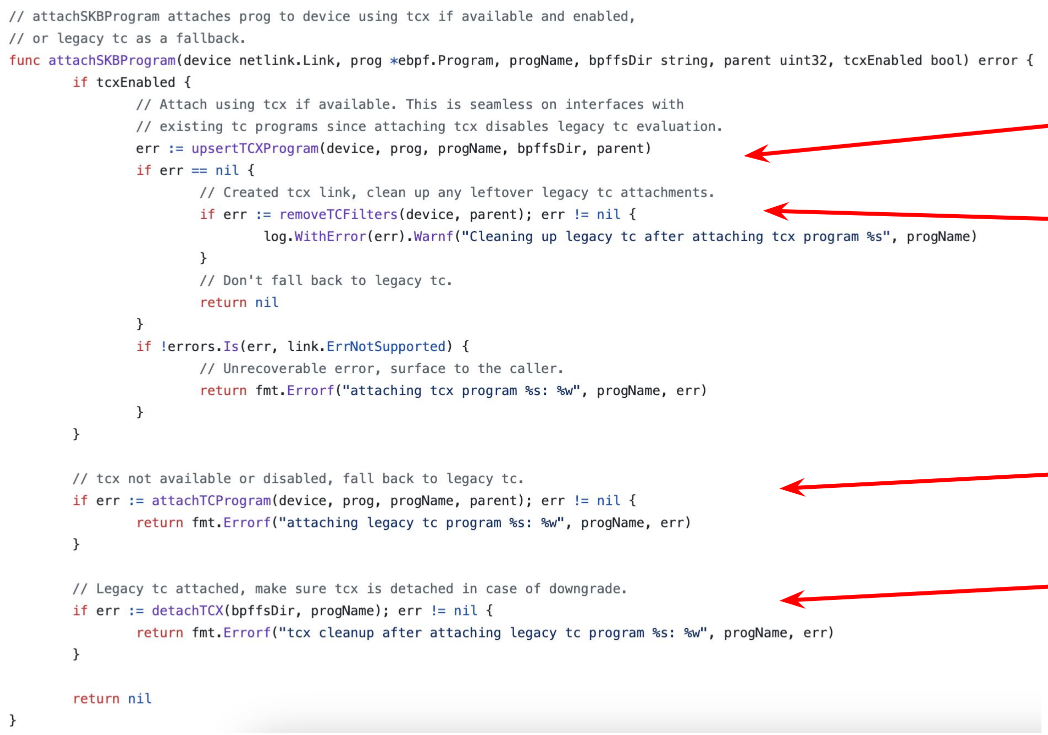
On a high level, this PR:

- attaches TC progs using bpf_link (tcx) respecting upgrade and downgrade paths
- uses per-endpoint bpffs dirs, at e.g. /sys/fs/bpf/cilium/endpoints/12345/links/cil_to_container
- adds netns-driven tests for attaching skb progs via tcx
- adds a Helm flag (enableTXC) to optionally disable the feature to ease integration with other tools

Closes #27632

Cilium & tcx: What’s done

Seamless up/downgrade path:



```go
// attachSKBProgram attaches prog to device using tcx if available and enabled,
// or legacy tc as a fallback.
func attachSKBProgram(device netlink.Link, prog *ebpf.Program, progName, bpffsDir string, parent uint32, tcxEnabled bool) error {
	if tcxEnabled {
		// Attach using tcx if available. This is seamless on interfaces with
		// existing tc programs since attaching tcx disables legacy tc evaluation.
		err := upsertTCXProgram(device, prog, progName, bpffsDir, parent)
		if err == nil {
			// Created tcx link, clean up any leftover legacy tc attachments.
			if err := removeTCFilters(device, parent); err != nil {
				log.WithError(err).Warnf("Cleaning up legacy tc after attaching tcx program %s", progName)
			}
			// Don't fall back to legacy tc.
			return nil
		}
		if !errors.Is(err, link.ErrNotSupported) {
			// Unrecoverable error, surface to the caller.
			return fmt.Errorf("attaching tcx program %s: %w", progName, err)
		}
	}

	// tcx not available or disabled, fall back to legacy tc.
	if err := attachTCProgram(device, prog, progName, parent); err != nil {
		return fmt.Errorf("attaching legacy tc program %s: %w", progName, err)
	}

	// Legacy tc attached, make sure tcx is detached in case of downgrade.
	if err := detachTCX(bpffsDir, progName); err != nil {
		return fmt.Errorf("tcx cleanup after attaching legacy tc program %s: %w", progName, err)
	}

	return nil
}
```

Update or attach tcx link

Removal of old-style tc filters

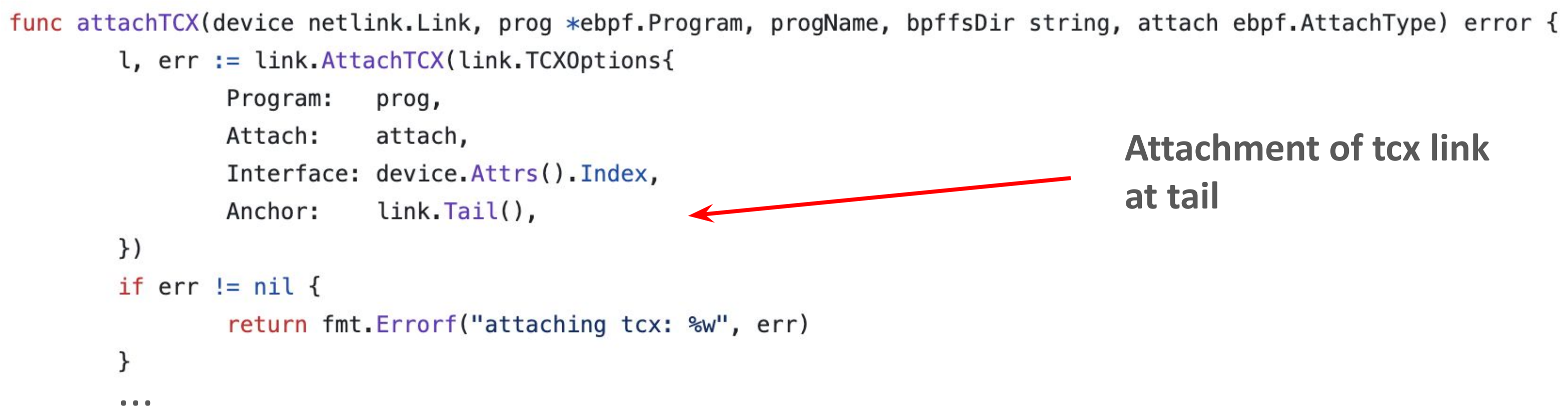
Attachment of old-style tc filters

Removal of tcx link

Cilium & tcx: What’s done

Attachment as “last”:

- Observability programs can attach in front of Cilium
- Cilium terminates tcx and does not enter into legacy tc



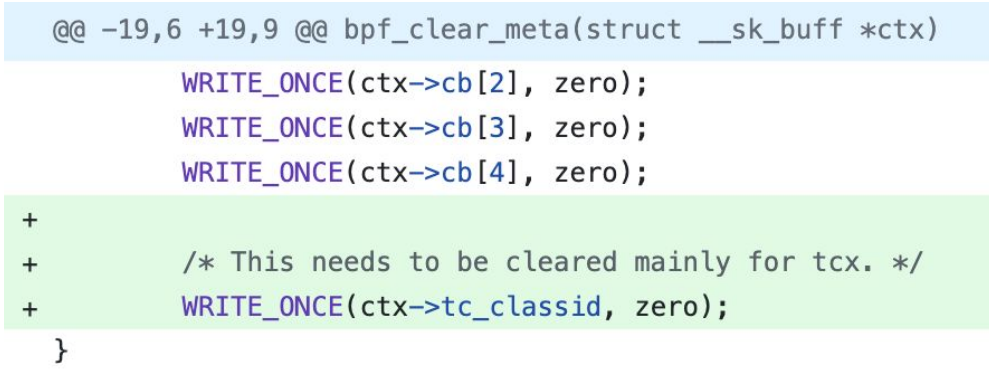
```
func attachTCX(device netlink.Link, prog *ebpf.Program, progName, bpffsDir string, attach ebpf.AttachType) error {
        l, err := link.AttachTCX(link.TCXOptions{
                Program:   prog,
                Attach:    attach,
                Interface: device.Attrs().Index,
                Anchor:    link.Tail(),
        })
        if err != nil {
                return fmt.Errorf("attaching tcx: %w", err)
        }
        ...
```

Attachment of tcx link at tail

Cilium & tcx: What’s done

Minor gotchas:

- Programs worked as-is, only tc_classid had to be zeroed explicitly in our code base
- With that all connectivity tests passed & we were able to merge it



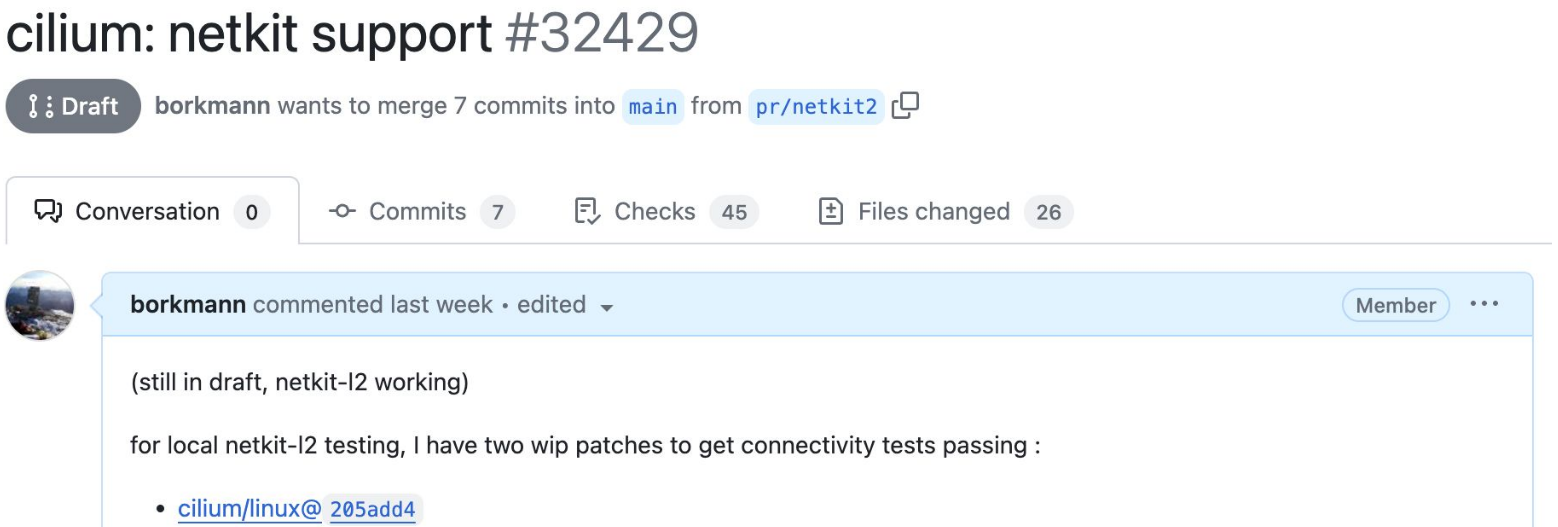
```
@@ -19,6 +19,9 @@ bpf_clear_meta(struct __sk_buff *ctx)
 	WRITE_ONCE(ctx->cb[2], zero);
 	WRITE_ONCE(ctx->cb[3], zero);
 	WRITE_ONCE(ctx->cb[4], zero);
+
+	/* This needs to be cleared mainly for tcx. */
+	WRITE_ONCE(ctx->tc_classid, zero);
 }
```

Cilium & netkit: What's ongoing

Integration and merge for Cilium 1.16 planned

- Goal: Last step of final Cilium integration via **--datapath-mode={veth,netkit,netkit-l2}**

- **netkit:** L3 mode, default peer policy if no BPF is attached: drop
- **netkit-l2:** Same as above but L2 mode



cilium: netkit support #32429

Draft borkmann wants to merge 7 commits into `main` from `pr/netkit2`

borkmann commented last week • edited

(still in draft, netkit-l2 working)

for local netkit-l2 testing, I have two wip patches to get connectivity tests passing :

- cilium/linux@ `205add4`

Cilium & netkit: What’s ongoing

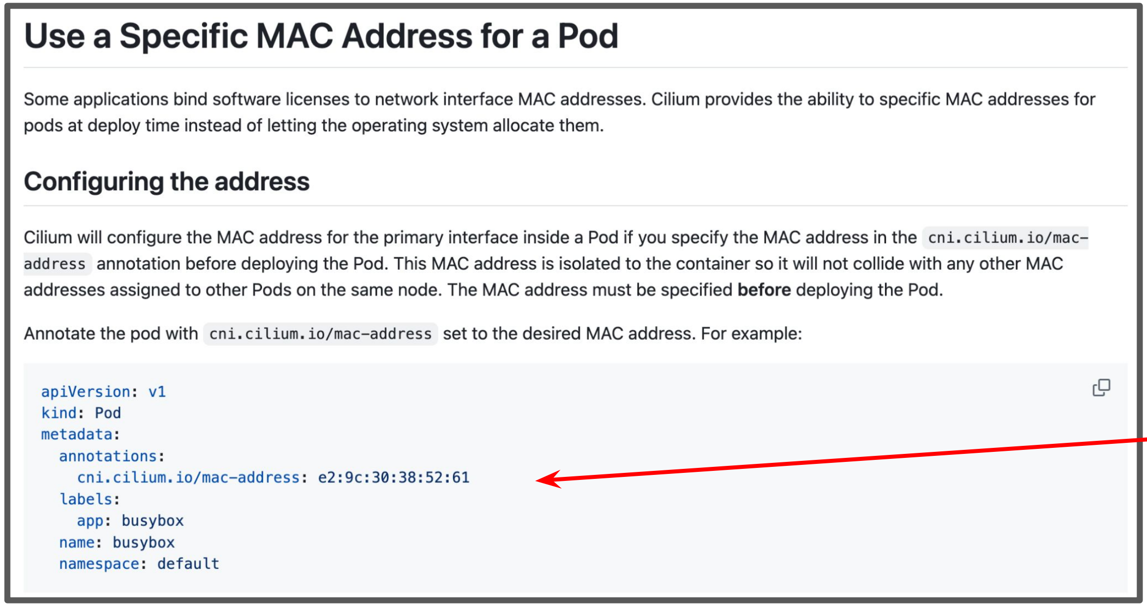
“netkit-l2” mode:

- Working but ran into two issues which needed netkit changes
  - Setting mac addresses in the driver (easy, patch coming)

Cilium & netkit: What's ongoing

"netkit-l2" mode:

- Working but ran into two issues which needed netkit changes
  - Setting mac addresses in the driver (easy, patch coming)



Use a Specific MAC Address for a Pod

Some applications bind software licenses to network interface MAC addresses. Cilium provides the ability to specific MAC addresses for pods at deploy time instead of letting the operating system allocate them.

Configuring the address

Cilium will configure the MAC address for the primary interface inside a Pod if you specify the MAC address in the `cni.cilium.io/mac-address` annotation before deploying the Pod. This MAC address is isolated to the container so it will not collide with any other MAC addresses assigned to other Pods on the same node. The MAC address must be specified **before** deploying the Pod.

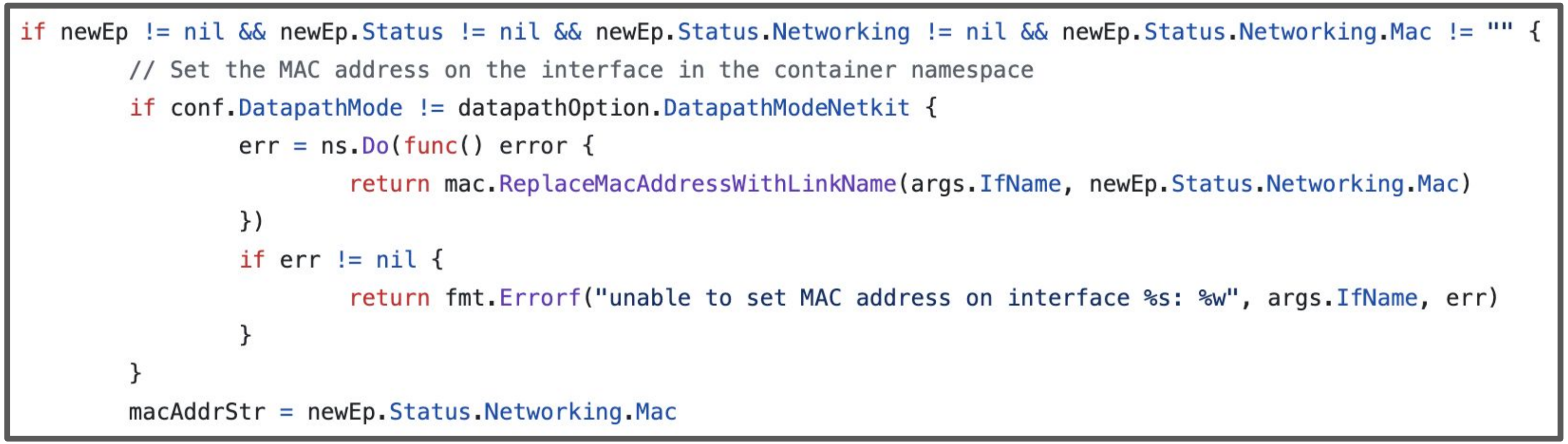
Annotate the pod with `cni.cilium.io/mac-address` set to the desired MAC address. For example:

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    cni.cilium.io/mac-address: e2:9c:30:38:52:61
  labels:
    app: busybox
  name: busybox
  namespace: default
```

Cilium & netkit: What's ongoing

"netkit-l2" mode:

- Working but ran into two issues which needed netkit changes
  - Setting mac addresses in the driver (easy, patch coming)



```
if newEp != nil && newEp.Status != nil && newEp.Status.Networking != nil && newEp.Status.Networking.Mac != "" {
        // Set the MAC address on the interface in the container namespace
        if conf.DatapathMode != datapathOption.DatapathModeNetkit {
                err = ns.Do(func() error {
                        return mac.ReplaceMacAddressWithLinkName(args.IfName, newEp.Status.Networking.Mac)
                })
                if err != nil {
                        return fmt.Errorf("unable to set MAC address on interface %s: %w", args.IfName, err)
                }
        }
        macAddrStr = newEp.Status.Networking.Mac
```

Cilium & netkit: What’s ongoing

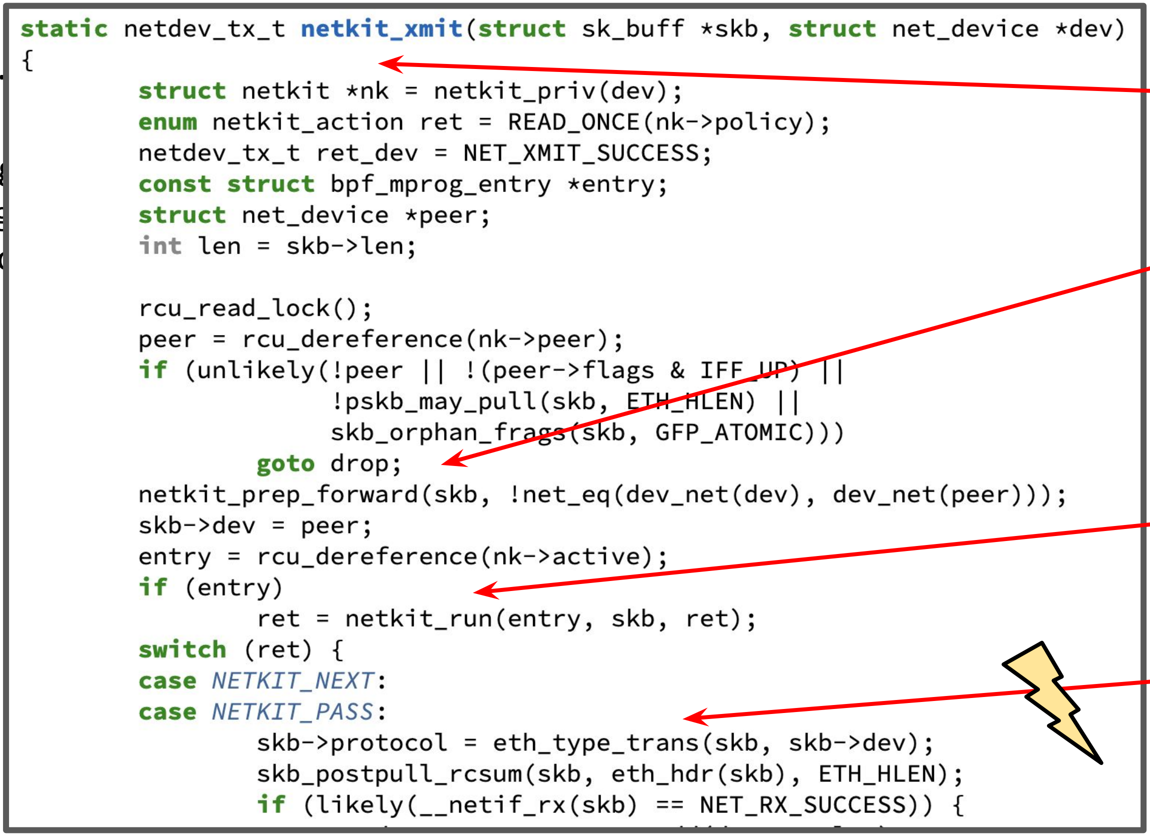
“netkit-l2” mode:

- Working but ran into two issues which needed netkit changes
  - Setting mac addresses in the driver (easy, patch coming)
  - Respecting setting skb->pkt_type from BPF program

Cilium & netkit: What’s ongoing

“netkit-

- Workin
  - Setting
  - Respec



```
static netdev_tx_t netkit_xmit(struct sk_buff *skb, struct net_device *dev)
{
        struct netkit *nk = netkit_priv(dev);
        enum netkit_action ret = READ_ONCE(nk->policy);
        netdev_tx_t ret_dev = NET_XMIT_SUCCESS;
        const struct bpf_mprog_entry *entry;
        struct net_device *peer;
        int len = skb->len;

        rcu_read_lock();
        peer = rcu_dereference(nk->peer);
        if (unlikely(!peer || !(peer->flags & IFF_UP) ||
                     !pskb_may_pull(skb, ETH_HLEN) ||
                     skb_orphan_frags(skb, GFP_ATOMIC)))
                goto drop;
        netkit_prep_forward(skb, !net_eq(dev_net(dev), dev_net(peer)));
        skb->dev = peer;
        entry = rcu_dereference(nk->active);
        if (entry)
                ret = netkit_run(entry, skb, ret);
        switch (ret) {
        case NETKIT_NEXT:
        case NETKIT_PASS:
                skb->protocol = eth_type_trans(skb, skb->dev);
                skb_postpull_rcsum(skb, eth_hdr(skb), ETH_HLEN);
                if (likely(__netif_rx(skb) == NET_RX_SUCCESS)) {
```

pkt_type = PACKET_OUTGOING

pkt_type = PACKET_HOST (via skb scrubbing)

pkt_type = xyz

pkt_type = PACKET_OTHERHOST

Causes L7 proxy test failures in Cilium!

Cilium & netkit: What’s ongoing

“netkit-l2” mode:

- Working but ran into two issues which needed netkit changes
  - Setting mac addresses in the driver (easy, patch coming)
  - Respecting setting skb->pkt_type from BPF program (only relevant here in netkit-l2)
    - **Options:**
      - In BPF program copy dst MAC to MAC of host device (retains PACKET_HOST)
      - Do eth_type_trans() before invoking BPF, needs skb push/pull dance
      - Detect that BPF program set pkt_type and override after eth_type_trans()
      - Remove eth_type_trans() and push responsibility into BPF program, **just do skb pull in driver if BPF otherwise eth_type_trans() if no BPF**

Cilium & netkit: What's ongoing

"netkit" mode:

- Next step after netkit-l2 is working
- Given L3 mode, BPF ARP responder needs to be compiled out
- Also all netkit-related MACs are zeroed (CNI records them, currently crashes with all-zero MAC - tbd)
- From datapath PoV ETH_HLEN still remains at 14
- BPF remains the same, except for local Pod-Pod: s/bpf_redirect_peer/bpf_redirect/
- Traffic to external must use bpf_redirect_neigh to populate L2

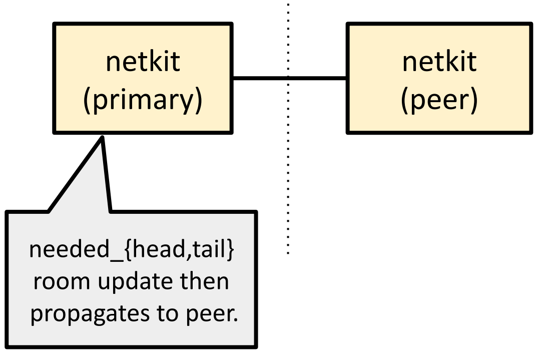
netkit: Future work

Experimenting with head/tailroom customization

- Goal: Being able to control dev->needed_headroom and dev->needed_tailroom

- Could benefit datapath performance under tunneling (vxlan, geneve) or encryption (wireguard)
    - Potentially avoids pskb_expand_head() reallocation costs

- Idea: Have actual IFLA_HEADROOM and IFLA_TAILROOM attributes to dump and set on a device
- needed_{head,tail}room is by default 0, vxlan/geneve adjusts needed_headroom, wireguard also needed_tailroom
- Performance benefit: to be measured, references from old patches mention costs around 5% on realloc



WIP: head/tailroom getter/setter in rtnetlink

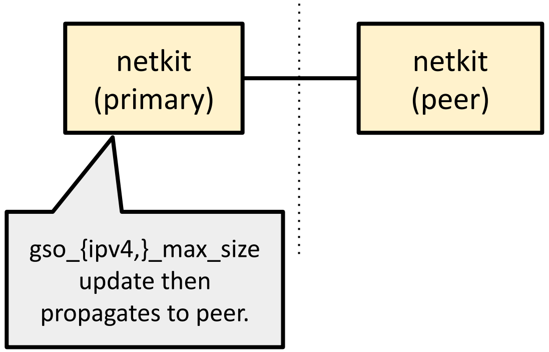
netkit: Future work

Adding new ndo for setting dev->gso_{ipv4,}_max_size

- Goal: Enabling BIG TCP for Pods without having to restart Pods

- Cilium agent is not able to exec into the Pod’s netns at runtime and mounting host procfs into Cilium container is not desired (security reasons). Only the Cilium CNI plugin has access when setting up devices.

- Downside: Enabling BIG TCP on an existing cluster requires restart of application Pod
- New ndo for updating dev->gso_{ipv4,}_max_size in similar style as dev->needed_{head,tail}room would be desirable.. e.g. picks max of primary/peer and applies it to both

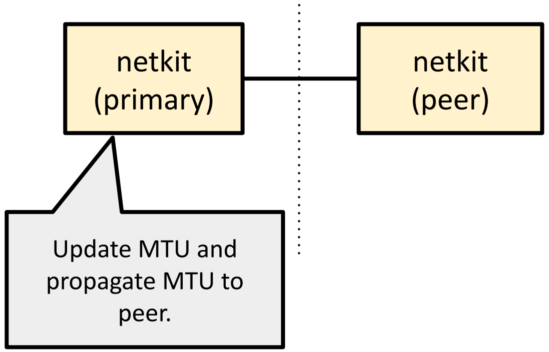


netkit: Future work

Implement ndo_change_mtu for netkit

- Goal: Changing MTU on primary without needing to change on peer

- Cilium agent is not able to exec into the Pod's netns at runtime and mounting host procfs into Cilium container is not desired (security reasons). Only the Cilium CNI plugin has access when setting up devices.



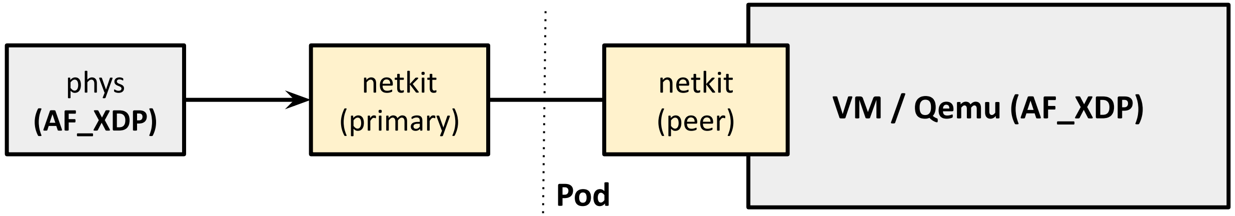
netkit: Future work

netkit and AF_XDP support

- Goal: 100G+ tput via AF_XDP with netkit without pulling in all the XDP infra into the driver. If these speeds can be achieved, then it would be more advantageous than SRIOV given there is still possibility of visibility / policy enforcement via BPF

Qemu now has native AF_XDP support:

```
"-netdev
af-xdp,id=str,ifname=name[,mode=native|skb][,force-copy=on|off][,queues=n][,start-queue=m][,inhibit=on|off][,sock-fds=x:y:...:z]"
```



“Global socket iterator”

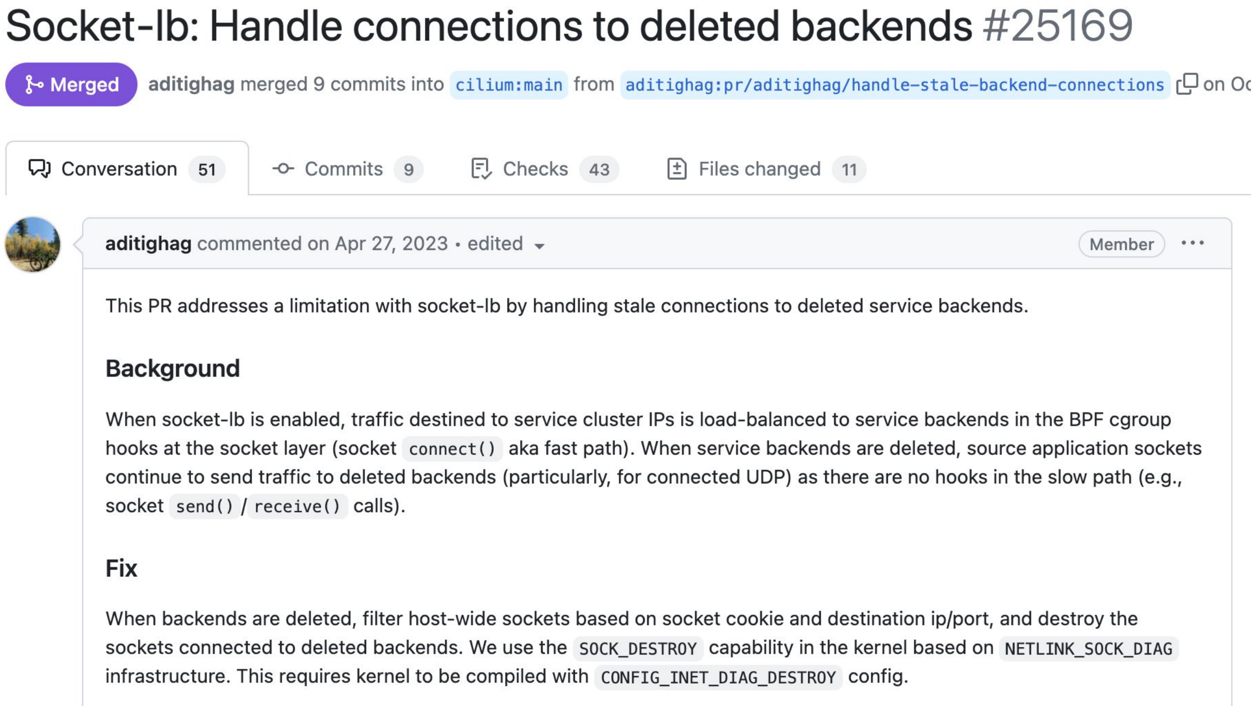
Problem: TCP/UDP connect binds VIP to backend, backend terminates, but application does not receive feedback for it. Stays connected, worst case: backend IP reuse.

Last attempt presented in LSF/MM/BPF 2023:

- Part 1: socket destroy kfunc
- Part 2: Iterator over netns’es given Cilium agent does not have access to Pod netns’es

“Global socket iterator”

In Cilium: Only solved in hostns today (via SOCK_DESTROY through DIAG infra)



Socket-lb: Handle connections to deleted backends #25169

Merged aditighag merged 9 commits into `cilium:main` from `aditighag:pr/aditighag/handle-stale-backend-connections` on Oc

Conversation 51 | Commits 9 | Checks 43 | Files changed 11

aditighag commented on Apr 27, 2023 • edited — Member

This PR addresses a limitation with socket-lb by handling stale connections to deleted service backends.

Background

When socket-lb is enabled, traffic destined to service cluster IPs is load-balanced to service backends in the BPF cgroup hooks at the socket layer (socket `connect()` aka fast path). When service backends are deleted, source application sockets continue to send traffic to deleted backends (particularly, for connected UDP) as there are no hooks in the slow path (e.g., socket `send()` / `receive()` calls).

Fix

When backends are deleted, filter host-wide sockets based on socket cookie and destination ip/port, and destroy the sockets connected to deleted backends. We use the `SOCK_DESTROY` capability in the kernel based on `NETLINK_SOCK_DIAG` infrastructure. This requires kernel to be compiled with `CONFIG_INET_DIAG_DESTROY` config.

“Global socket iterator”

Problem: TCP/UDP connect binds VIP to backend, backend terminates, but application does not receive feedback for it. Stays connected, worst case: backend IP reuse.

Last attempt presented in LSF/MM/BPF 2023:

- Part 1: socket destroy kfunc (Cilium upgraded to LLVM 17 few weeks ago, last blocker for kfuncs gone)
- Part 2: Iterator over netns’es given Cilium agent does not have access to Pod netns’es

Possible options:

- Plumb global flag for bpf_iter_attach_opts (LSF/MM/BPF 2023): not flexible enough
- Socket connect() call records {backendIP/port + socket address or cookie} -> {socket kptr} in hash map and upon destruction we iterate hash map, fetch kptr and destroy socket
  - Downside: needs to hold reference on socket
- Sockmap as storage which does not need to hold reference, but installs psock and might have other bumps
- Open-coded netns iterator and then we feed netns pointer into open-coded socket iterator

netns iterator

Slow-path example upon backend termination event:

```
SEC("fentry/" SYS_PREFIX "sys_getpgid")
int foo_nested(void *ctx)
{
	struct task_struct *cur_task = bpf_get_current_task_btf();
	struct sock_common *skc;
	struct net *net;

	if (cur_task->pid == target_pid) {
		bpf_for_each(net, net) {
			bpf_for_each(tcp, skc, net) {
				if (bpf_get_socket_cookie(skc) == cookie) {
					bpf_sock_destroy(skc);
				}
			}
		}
	}
	return 0;
}
```

netns iterator

```
struct bpf_iter_net {
        __u64 __opaque[3];
} __attribute__((aligned(8)));

struct bpf_iter_net_kern {
        struct net **net_array;
        netns_tracker ns_tracker;
        u32 len;
        u32 pos;
} __attribute__((aligned(8)));
```

Can be used in sleepable / non-sleepable programs.

```
__bpf_kfunc int bpf_iter_net_new(struct bpf_iter_net *it)
{
        struct bpf_iter_net_kern *kit = (void *)it;
        struct net *net, **tmp;
        u32 i;

        BUILD_BUG_ON(sizeof(struct bpf_iter_net_kern) > sizeof(struct bpf_iter_net));
        BUILD_BUG_ON(__alignof__(struct bpf_iter_net_kern) !=
                                        __alignof__(struct bpf_iter_net));
        kit->len = kit->pos = 0;
        kit->net_array = NULL;
        rcu_read_lock();
        for_each_net_rcu(net) {
                tmp = realloc_array(kit->net_array, kit->len, kit->len + 1,
                                    sizeof(*kit->net_array), GFP_ATOMIC, false);
                if (!tmp) {
                        rcu_read_unlock();
                        goto unwind;
                }
                kit->net_array = tmp;
                kit->net_array[kit->len++] = get_net_track(net, &kit->ns_tracker,
                                                           GFP_ATOMIC);

        }
        rcu_read_unlock();
        return 0;
unwind:
        for (i = 0; i < kit->len; i++)
                put_net_track(kit->net_array[i], &kit->ns_tracker);
        kfree(kit->net_array);
        return -ENOMEM;
}
```

netns iterator

```
__bpf_kfunc struct net *bpf_iter_net_next(struct bpf_iter_net *it)
{
        struct bpf_iter_net_kern *kit = (void *)it;
        struct net *pos = NULL;

        if (kit->pos < kit->len) {
                pos = kit->net_array[kit->pos];
                kit->pos++;
        }
        return pos;
}

__bpf_kfunc void bpf_iter_net_destroy(struct bpf_iter_net *it)
{
        struct bpf_iter_net_kern *kit = (void *)it;
        u32 i;

        for (i = 0; i < kit->len; i++)
                put_net_track(kit->net_array[i], &kit->ns_tracker);
        kfree(kit->net_array);
}
```

netns iterator

Open TODOs:

net argument required to be trusted input argument:

```
bpf_for_each(net, net)
    bpf_for_each(tcp, skc, net)
```

- Refcount handling is part of bpf_iter_net_new() / bpf_iter_net_destroy()
- Either bpf_iter_net_next() described as KF_RET_TRUSTED or we assign obj id as if it was refcounted (tbd)
- Still needs TCP/UDP socket iterator conversion to open-coded iterator as next step

ISOVALENT

Thank you! Questions?

github.com/cilium/cilium

tcx BPF datapath

netkit devices

Open coded iterators for netns