1) netkit updates
2) BPF for user space live patching

Daniel Borkmann (Isovalent)
LSF/MM/BPF 2026

ISOVALENT
now part of cisco

Part 1) netkit updates

The idea back then...

- Accelerate KubeVirt (aka qemu inside netns) to *eventually* achieve full zero-copy for VMs

- Qemu got an AF_XDP backend driver merged not too long ago, which could act as a foundation

- AF_XDP & more general other zero-copy memory providers cannot be used out of Pods/containers, hence the idea of physical to virtual queue leasing so that apps can bind against a virtual device to still access physical device's HW capabilities

- Must work for:
  - io_uring zero copy Rx
  - TCP devmem
  - AF_XDP with native ZC

Status: RX queue leasing merged now

- main series, plus follow-ups for netkit, net selftests; ynl spec:

```
-
  name: queue-create
  doc: |
    Create a new queue for the given netdevice. Whether
    is supported depends on the device and the driver.
  attribute-set: queue
  flags: [admin-perm]
  do:
    request:
      attributes:
        - ifindex
        - type
        - lease
    reply: &queue-create-op
      attributes:
        - id
```

```
-
  name: lease
  attributes:
    -
      name: ifindex
      doc: The netdev ifindex to lease the queue from.
      type: u32
      checks:
        min: 1
    -
      name: queue
      doc: The netdev queue to lease from.
      type: nest
      nested-attributes: queue-id
    -
      name: netns-id
      doc: The network namespace id of the netdev.
      type: s32
```

Leftover TODOs on RX queue leasing?

Topics we were planning to address later:

- Combining netkit with io_uring zero copy Rx + large buffers
  - Pavel's series Add support for providers with large rx buffer
  - 30% CPU utilization improvement for 32k vs 4k buffers in io_uring ZC on 200G NIC
  - Works out of the box with netkit given we just proxy virt-to-phys queue before the actual call into __netif_mp_open_rxq

Leftover TODOs on RX queue leasing?

Topics we were planning to address later:

- Combining netkit with io_uring zero copy Rx + large buffers + BIG TCP
  - Old netconf 2023 idea to combine zero copy with BIG TCP to reduce stack traversal cost
  - Now becomes feasible, and also works out of the box since skbs traverse GRO engine normally
  - Bumping default CONFIG_MAX_SKB_FRAGS complements it but not necessary for io_uring

Mainly just showcasing both as selftests: https://github.com/cilium/linux/commits/pr/netkit-qcfg/

Leftover TODOs on RX queue leasing?

User space consumers:

- Integration into Cilium via native queue leasing DRA driver
  - Pod will be able to specify how many HW queues should be leased to netkit
  - Once K8s Service is set up with the Pod as backend, Cilium could install flow steering such that traffic lands on the right RX queues where io_uring TCP ZC will be used

PoC for Cilium planned until KubeCon US

TX queue leasing

- Container-native zero-copy networking is asymmetric today
- TX side is missing: container can't bind-tx dmabuf, can't register an AF_XDP TX pool against netkit

- Goal: Same lease pointer model, but for TX
  - ndo_queue_create get queue type argument
  - netkit would be able to create up to 1024 RX/TX queues
  - genl queue_create accepts NETDEV_QUEUE_TYPE_TX

TX queue leasing

- Forwarding becomes a bit trickier: dev->netmem_tx = {true,false} gates validate_xmit_unreadable_skb() and drops dmabuf-frag skbs inside the container/netns unless the next-hop driver advertises support
- netkit never DMAs and never touches skb frag contents - it just forwards

- Proposal from Bobby Eshleman (Meta) to replace with a 2-bit enum
  - NETMEM_TX_NONE: drop unreadable skbs (default)
  - NETMEM_TX_DMA: phys driver, sets up IOMMU, does DMA (bnxt, mlx5e, gve, fbnic converted)
  - NETMEM_TX_NO_DMA: pure pass-through (netkit)

- Validator now lets NO_DMA devices forward to eventual phys netdev

Combined PoC: https://github.com/cilium/linux/commits/pr/netkit-followups/

netkit & BPF TPROXY

Currently BPF TPROXY is limited to ingress-only, with netkit goal is to also enable on egress

- Today's skb_orphan guard is buggy, sits in act_bpf instead of skb_do_redirect (fix#1)
- Refcounted sockets are safe for egress path to open up (fix#2)
- On egress SOCK_RCU_FREE sockets need a refcount to survive qdisc or backlog queue
  - sock_pfree_ref vs sock_pfree destructor to distinguish that ref was taken
  - in skb_do_redirect instead of orphaning sock_pfree skbs, we migrate to sock_pfree_ref
  - Extra reference not optimal, but for L7 Envoy slow-path fine for now

PoC: https://github.com/cilium/linux/commits/pr/bpf-tproxy/

Part 2) BPF for user space live patching

BPF user space live patching

Use cases:

- Existing uprobes + BPF: Optimizing tracing without a kernel-trap cost
- Application live patching: Fixing or sanitizing vulnerabilities (think: Mythos/Glasswing) when there are constraints on how fast applications can be redeployed
- Applications could use it co-operatively to dynamically change app behavior at runtime

BPF user space live patching

Use cases:

- Existing uprobes + BPF: Optimizing tracing without a kernel-trap cost
- Application live patching: Fixing or sanitizing vulnerabilities (think: Mythos/Glasswing) when there are constraints on how fast applications can be redeployed
- Applications could use it co-operatively to dynamically change app behavior at runtime

What it could buy us:

- Argument sanitization at function entry (clamp, allowlist, normalize inputs, etc)
- Short-circuit return (error out to skip the vulnerable body)
- Reimplement the function body with the fixed logic (longer-term)
- Telemetry into applications (how often did the problematic code fire?)

BPF user space live patching

Why with BPF and not just binary blob?

- Injected code must not crash applications (vs. not crashing kernel)
- Composes & grows with existing BPF capabilities: insns, verifier, BTF, JIT (see Alexei's session)
- Information sharing with kernel BPF (e.g. BPF LSM, sched-ext, tcx, etc) for application policies
- Agent-less control-plane given kernel manages BPF
- Can be combined with KP's signed loader work

BPF user space live patching

Why with BPF and not just binary blob?

- Injected code must not crash applications (vs. not crashing kernel)
- Composes & grows with existing BPF capabilities: insns, verifier, BTF, JIT (see Alexei's session)
- Information sharing with kernel BPF (e.g. BPF LSM, sched-ext, tcx, etc) for application policies
- Agent-less control-plane given kernel manages BPF
- Can be combined with KP's signed loader work

Constraints:

- Restricting to only allow BPF arena as a communication mechanism, no BPF maps
- BPF kernel helpers/kfuncs are not available potentially could be user provided
  (call id -> relative func address + BTF signature for verifier)
  - Useful for things which cannot be implemented in BPF yet (same thinking as in kernel land)
  - Different mappings for different applications, no fixed "UAPI" needed

BPF user space live patching

```
userspace process
-----------------

app text:
 foo:    jmp <tramp>      ← patched
 foo+5:  <displaced…>

trampoline page (PROT_RX, kernel-installed)
 prologue:  save clobbered regs
 body:      read args, call user helper
            maybe write back / synthesize
 epilogue:  jmp foo+5      (fall through)
            or ret         (short-circuit)

user helpers (loader-supplied, app code)
 sanitize_path, clamp_iov, allowlist…
arena (rules, counters, telemetry rb)
```

```
kernel
------

BPF_PROG_LOAD:
  verifier (ABI-preservation pass)
  JIT → PIC code for user RX page
  BPF LSM as gate, signature check

BPF_LINK_CREATE:
  locate VMA (build-id, offset)
  map RX page into target pid
  process_text_poke() rewrite

arena: shared mmap, kernel + user BPF
kernel-side BPF (LSM, kprobe, tc) can
read/write same arena → cross-plane policy
```

Thank you! Questions?

ISOVALENT
now part of cisco