WireGuard and GRO?

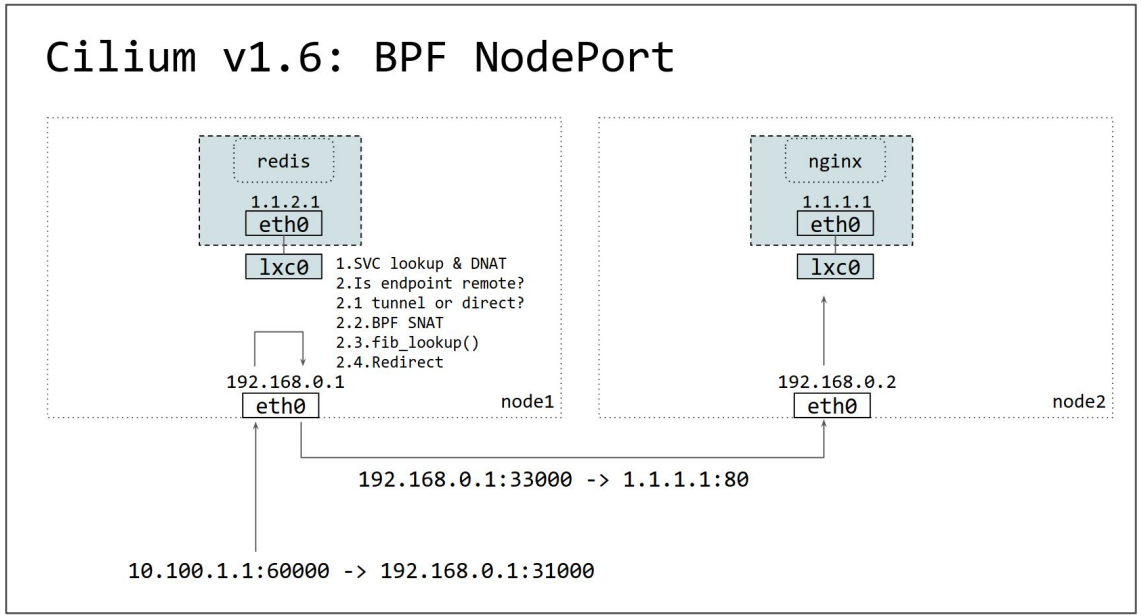
Improving WireGuard performance

Daniel Borkmann
Anton Protopopov
Martynas Pumputis

ISOVALENT
now part of CISCO

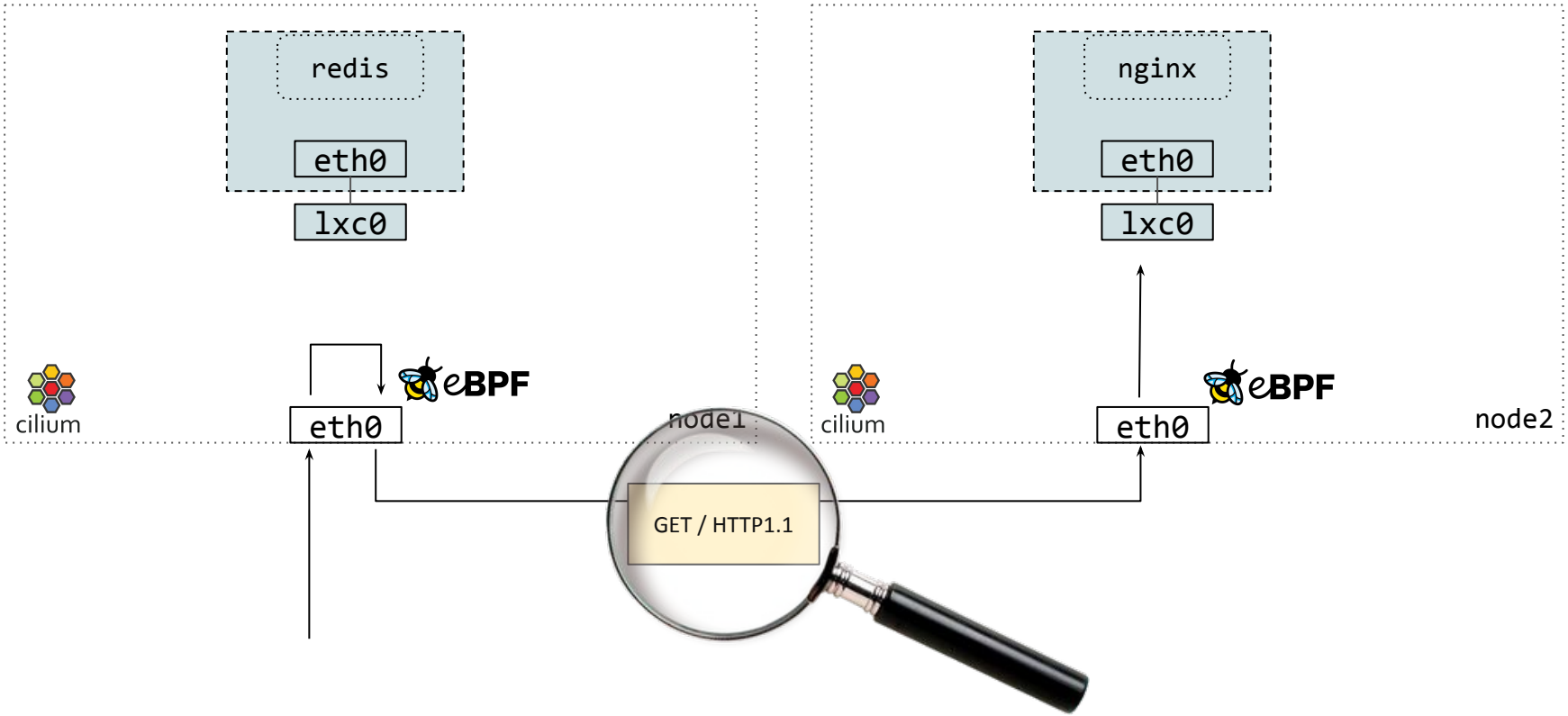
LINUX
PLUMBERS
CONFERENCE Vienna, Austria / Sept. 18-20, 2024

History (Cilium LB)



"Making the Kubernetes Service Abstraction Scale using BPF", LPC 2019

History (Cilium LB)

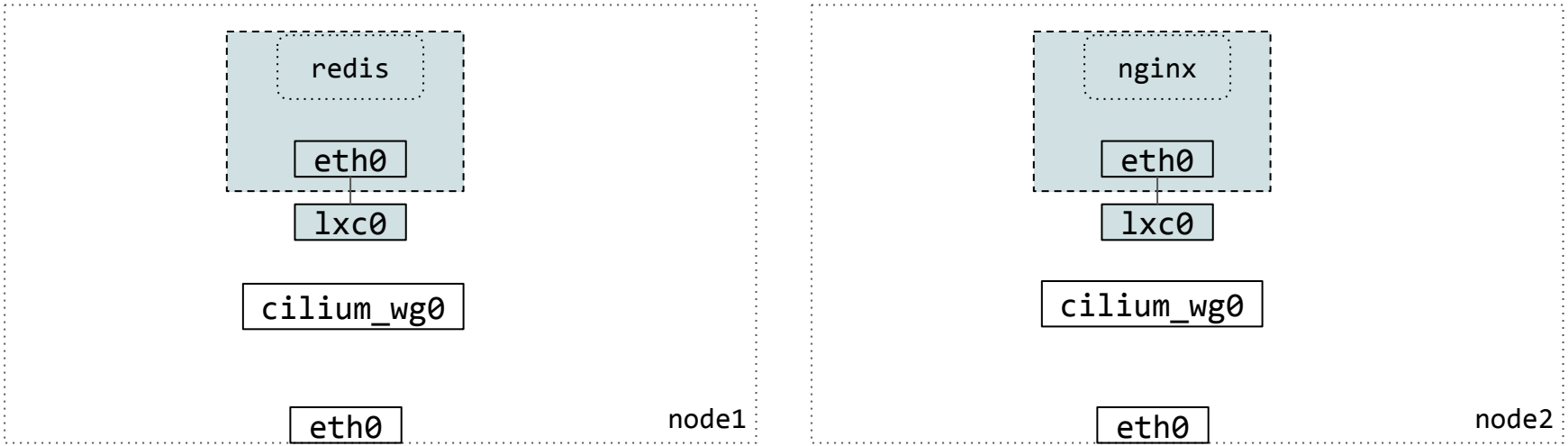


History (Cilium Encryption)

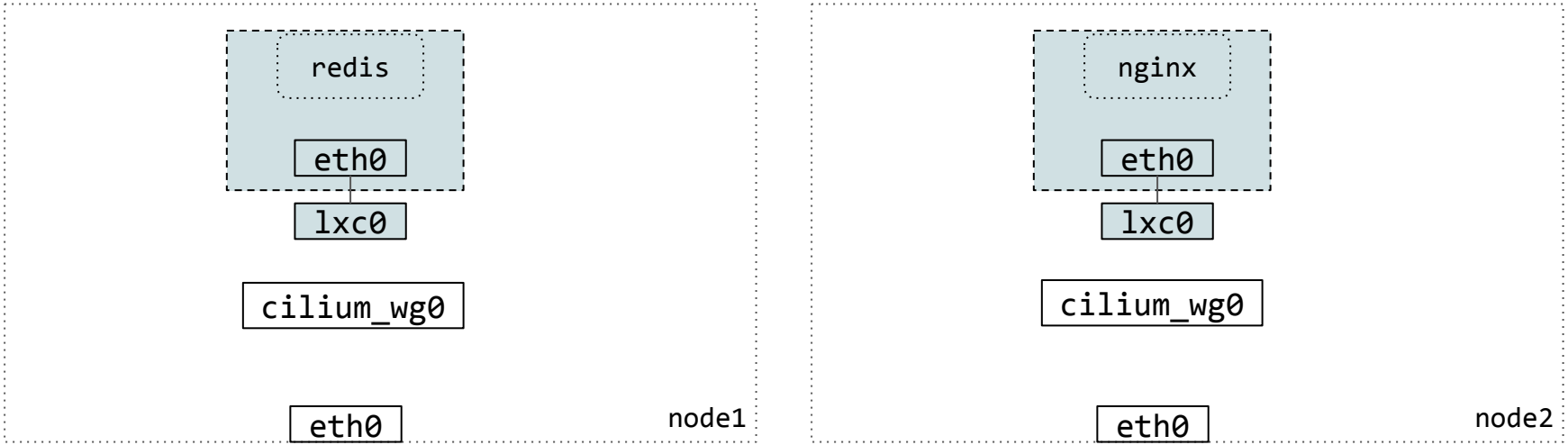
- IPSec integration since Cilium v1.14 for inter-container traffic
  - Host stack does encryption via kernel XFRM framework
  - Cannot just `bpf_redirect()` to encrypt
  - Tricky integration due to reliance on `skb->tc_index` and `skb->mark`
  - No automated key rotation (no IKE)

- WireGuard in CI to test Cilium LB L3 to L2 netdev redirection
  - Dedicated netdev for encryption (`cilium_wg0`)
  - Simple setup and auto key rotation (just exchange pub keys)

Cilium WireGuard integration



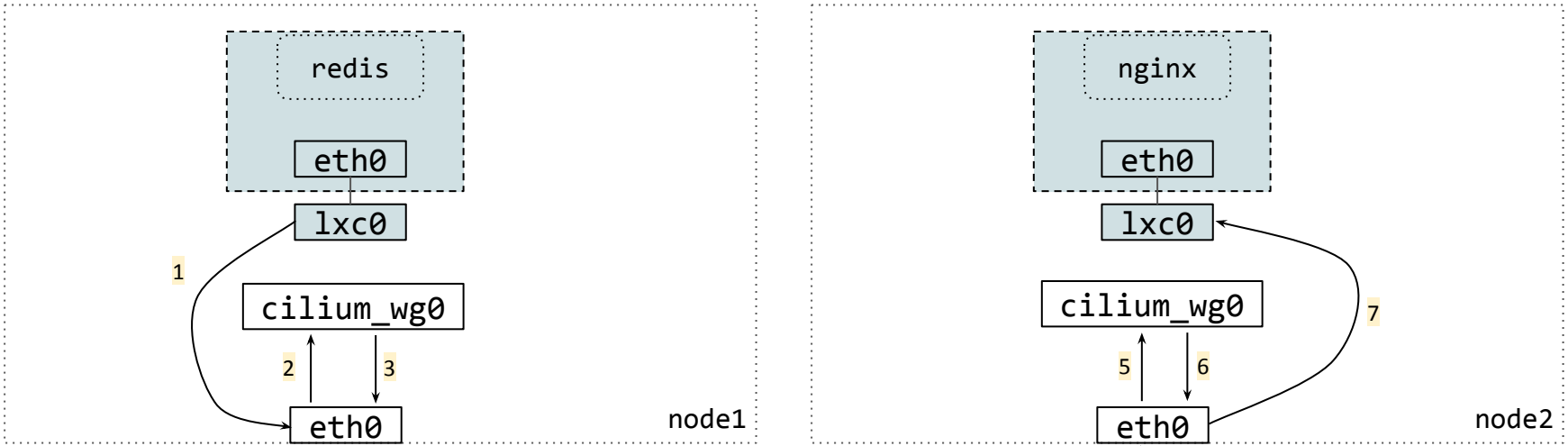
Cilium WireGuard integration



1. wg genkey # 1-{pub,priv}.keys
2. wg set cilium_wg0
   listen-port 51871
   private-key 1-priv.key
   peer 2-pub.key
   allowed-ips 1.1.1.1,<…>
   endpoint 192.168.0.2:51871

1. wg genkey # 2-{pub,priv}.keys
2. wg set cilium_wg0
   listen-port 51871
   private-key 2-priv.key
   peer 1-pub.key
   allowed-ips 1.1.2.1,<…>
   endpoint 192.168.0.1:51871

Cilium WireGuard integration



1. wg genkey # 1-{pub,priv}.keys
2. wg set cilium_wg0
   listen-port 51871
   private-key 1-priv.key
   peer 2-pub.key
   allowed-ips 1.1.1.1,<…>
   endpoint 192.168.0.2:51871

1. wg genkey # 2-{pub,priv}.keys
2. wg set cilium_wg0
   listen-port 51871
   private-key 2-priv.key
   peer 1-pub.key
   allowed-ips 1.1.2.1,<…>
   endpoint 192.168.0.1:51871

Cilium WireGuard (userspace) integration

- User-space mode to support WireGuard on < 5.6 kernels (now deprecated)
  - Relies on TUN device
  - Not intended for production use (cannot withstand cilium-agent restarts)
  - Probably not performant enough (?)

WireGuard driver vs WireGuard-go

- WireGuard-go got support for UDP GRO/GSO
  - Blog: “Userspace isn't slow, some kernel interfaces are!”
  - Blog: “Surpassing 10Gb/s over Tailscale”

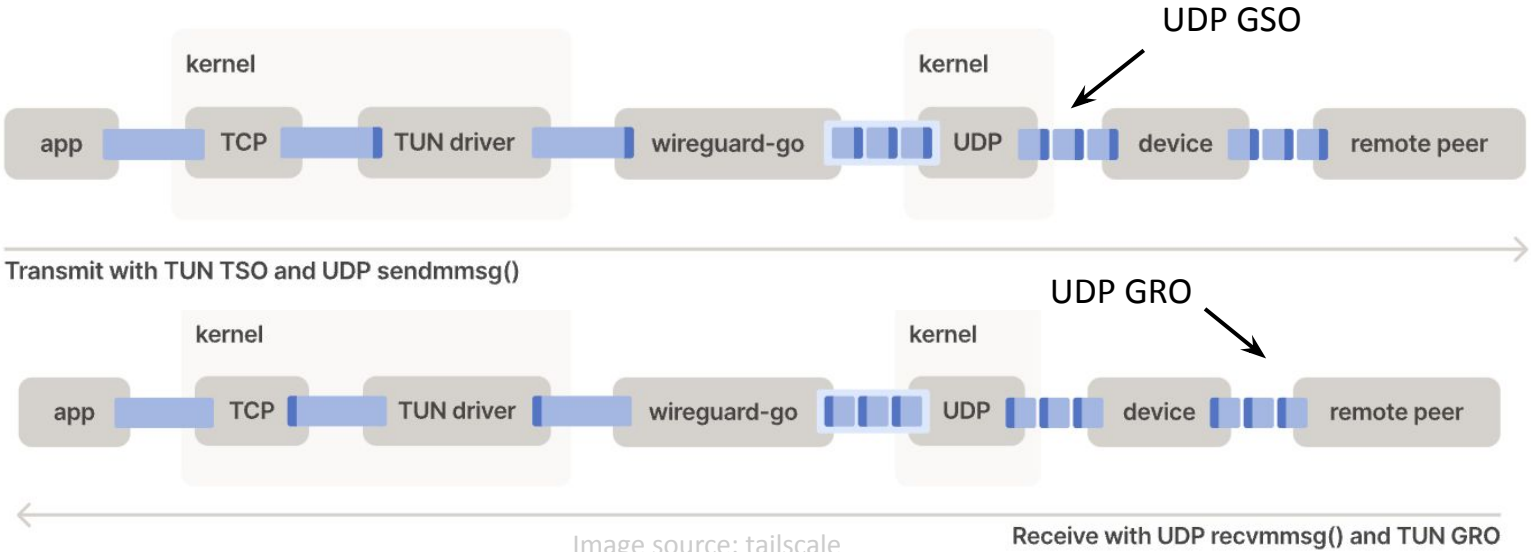
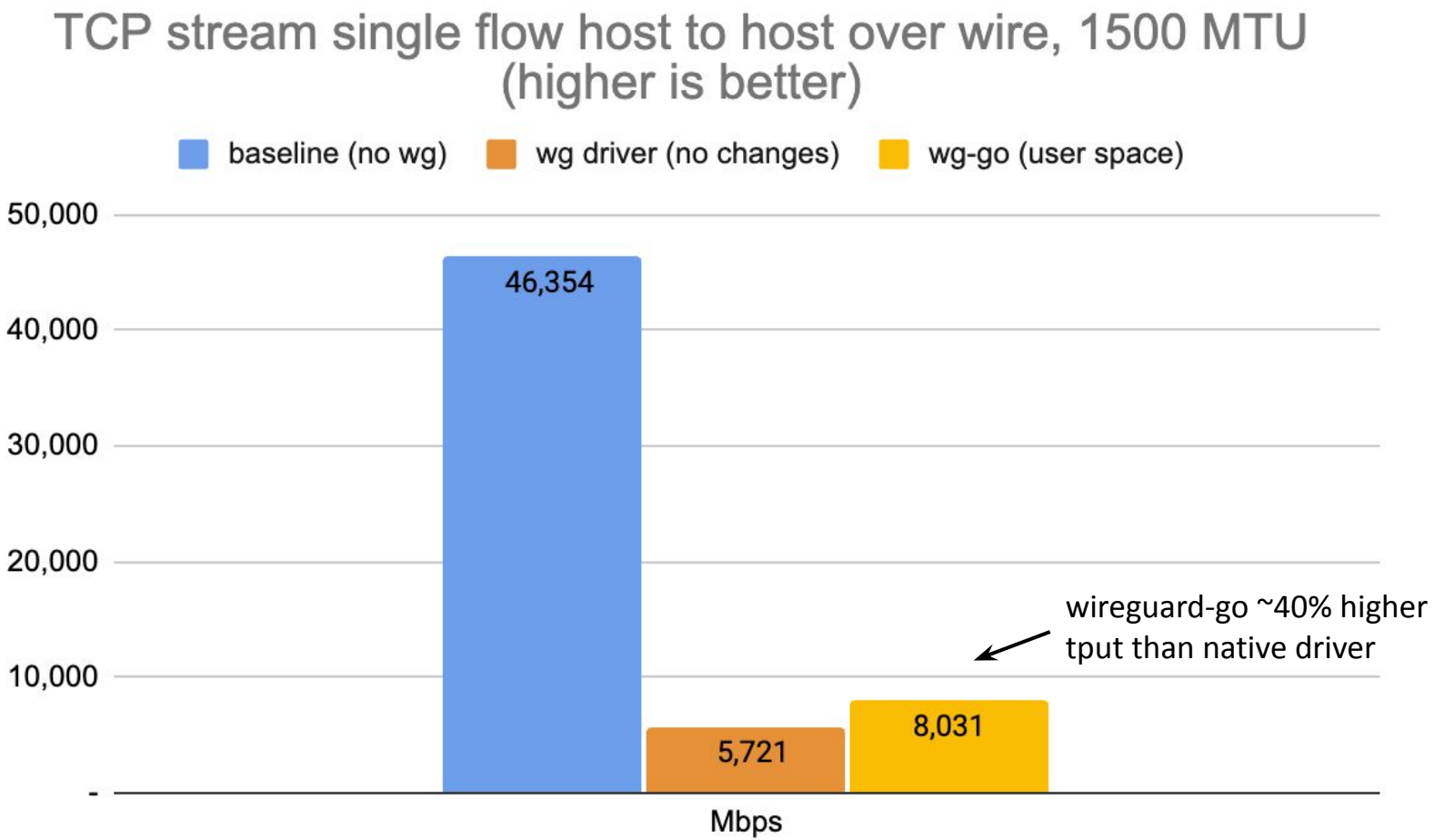


Image source: tailscale

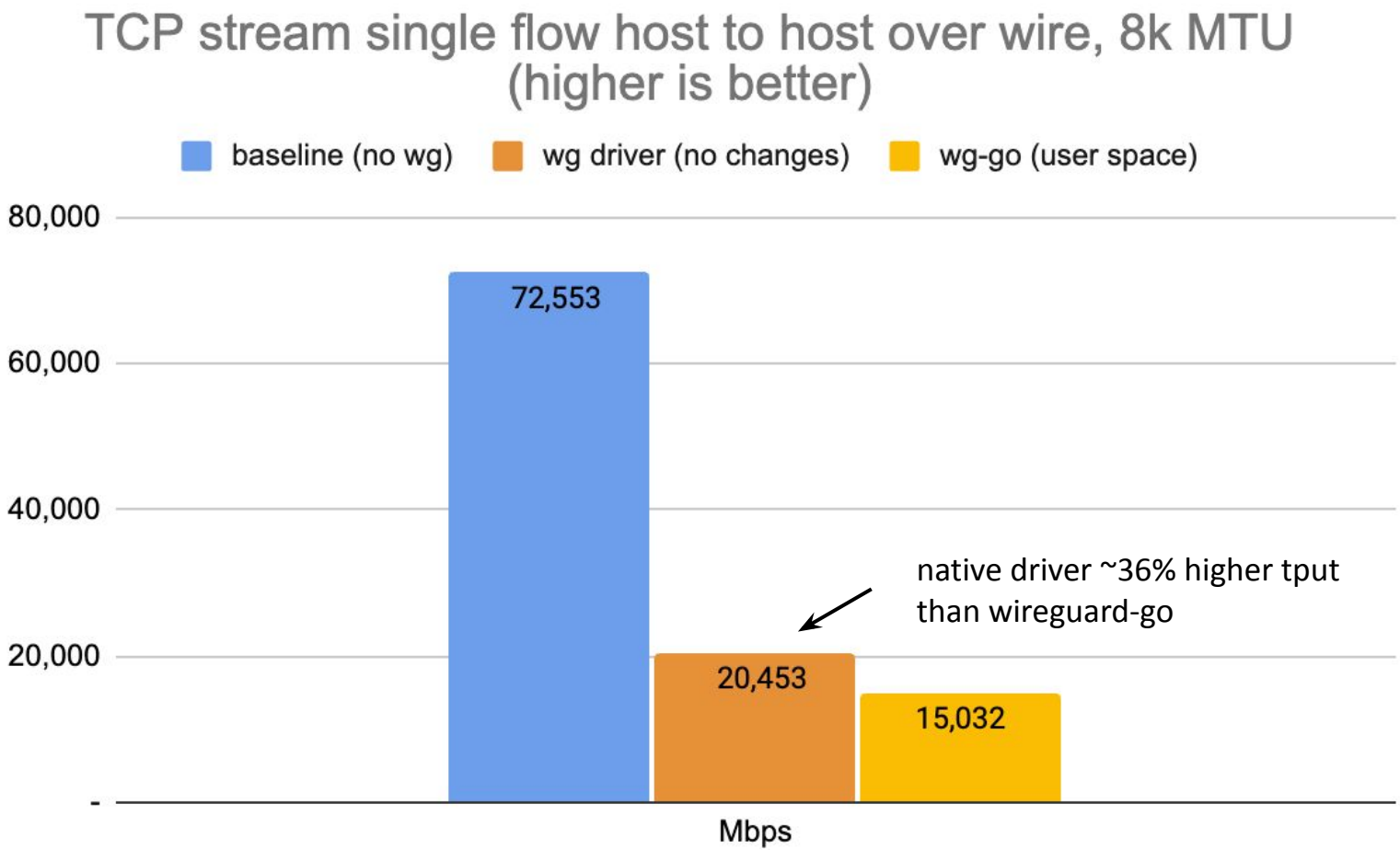
WireGuard benchmark setup

- AMD Ryzen 9 3950X @ 3.5 GHz, 128G RAM @ 3.2 GHz, PCIe 4.0
- 100Gb/s dual port ConnectX-6 Dx (mlx5), LRO enabled
- PREEMPT_NONE, IRQs pinned, no SMT, CPU gov: performance
- CPU mitigations compiled out
- BIG TCP enabled
- Git trees: net tree, wireguard-go (12269c27617)



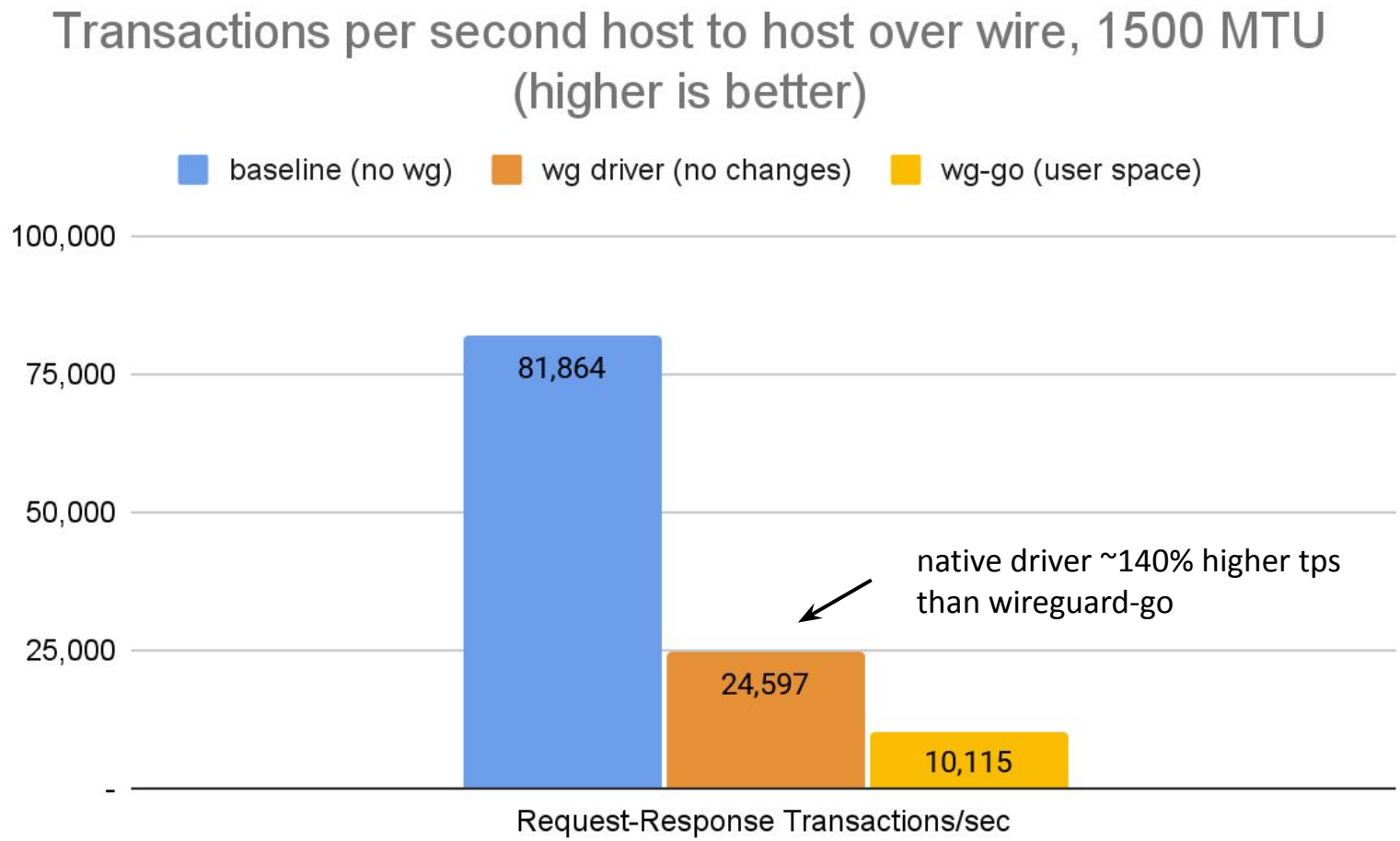
DB

Back to back: AMD Ryzen 9 3950X @ 3.5 GHz, 128G RAM @ 3.2 GHz, PCIe 4.0, ConnectX-6 Dx, mlx5 driver, striding mode, LRO, 1500 MTU



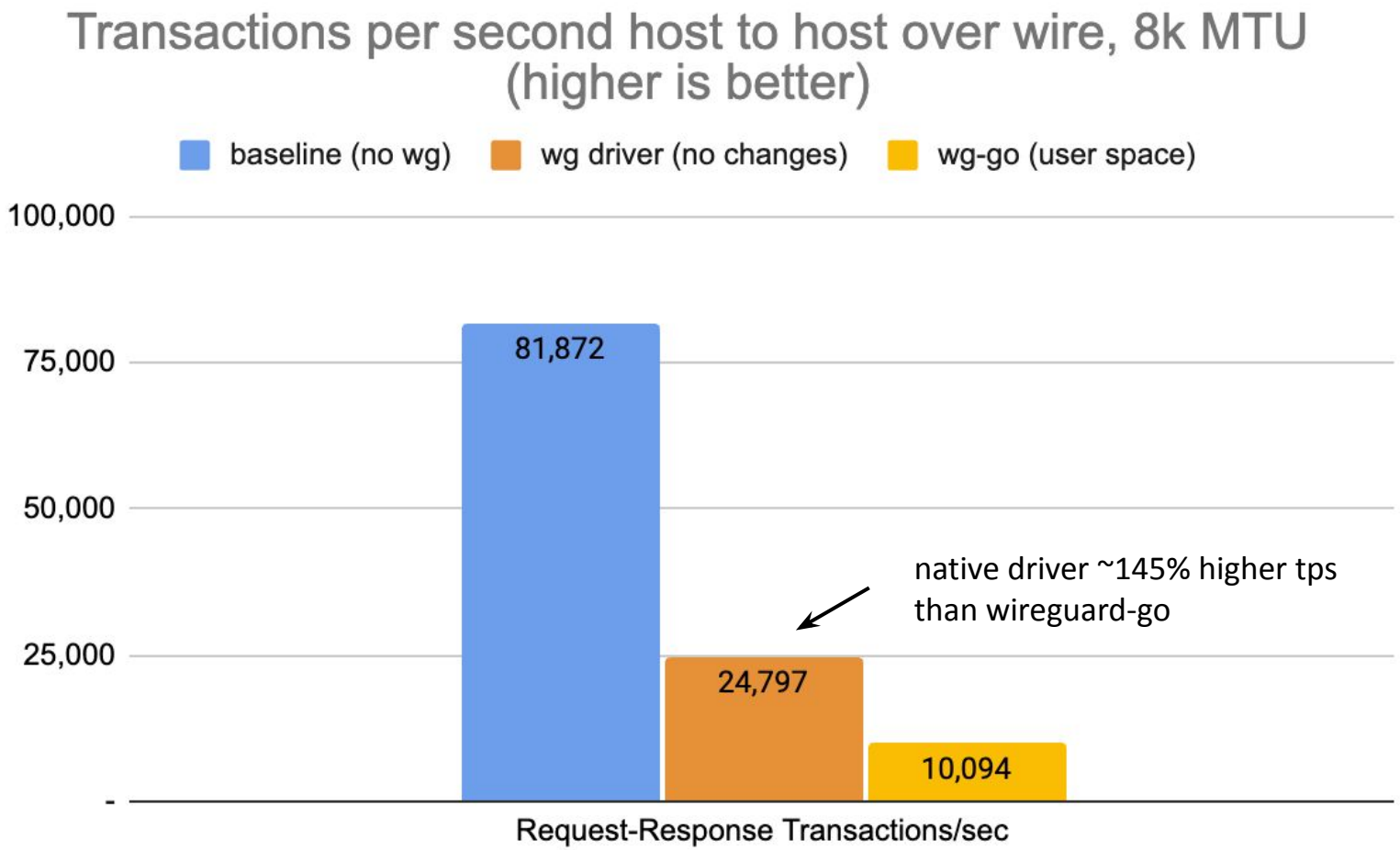
DB

Back to back: AMD Ryzen 9 3950X @ 3.5 GHz, 128G RAM @ 3.2 GHz, PCIe 4.0, ConnectX-6 Dx, mlx5 driver, striding mode, LRO, 8k MTU



DB

Back to back: AMD Ryzen 9 3950X @ 3.5 GHz, 128G RAM @ 3.2 GHz, PCIe 4.0, ConnectX-6 Dx, mlx5 driver, striding mode, LRO, 1500 MTU

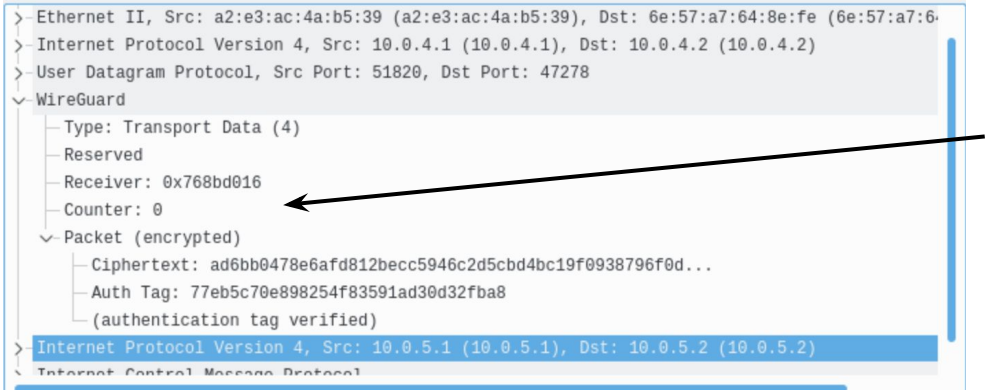


DB

Back to back: AMD Ryzen 9 3950X @ 3.5 GHz, 128G RAM @ 3.2 GHz, PCIe 4.0, ConnectX-6 Dx, mlx5 driver, striding mode, LRO, 8k MTU

Can we still do better for the native driver?

- How does GRO/GSO currently work in the native WireGuard driver?
- GRO:
  - Individual UDP packets (no GRO) go up the stack into WireGuard socket
  - WireGuard decrypts, then aggregates via napi_gro_receive(&peer->napi, skb)
- GSO:
  - Stack can send up to 64k GSO packets down into wg device
  - WireGuard segments via skb_gso_segment(skb, 0), then encrypts



nonce/counter for replay protection part of header
Means: skb_gso_segment() is here to stay..

Can we still do better for the native driver?

- Low hanging fruit? Two ideas:
- GRO:
  - Instead of sending individual packets up the stack into the UDP socket, why not take a similar approach as xfrm's ESP offload?
  - GRO handler enqueues the skb internally for decryption, returns ERR_PTR(-EINPROGRESS) back to GRO engine to tell skb has been GRO_CONSUMED
  - Details: see Steffen's IPsec GRO layer decapsulation
- GSO:
  - Enable BIG TCP support for the driver to allow even bigger packets to reach the device: netif_set_tso_max_size(dev, GSO_MAX_SIZE) during dev setup

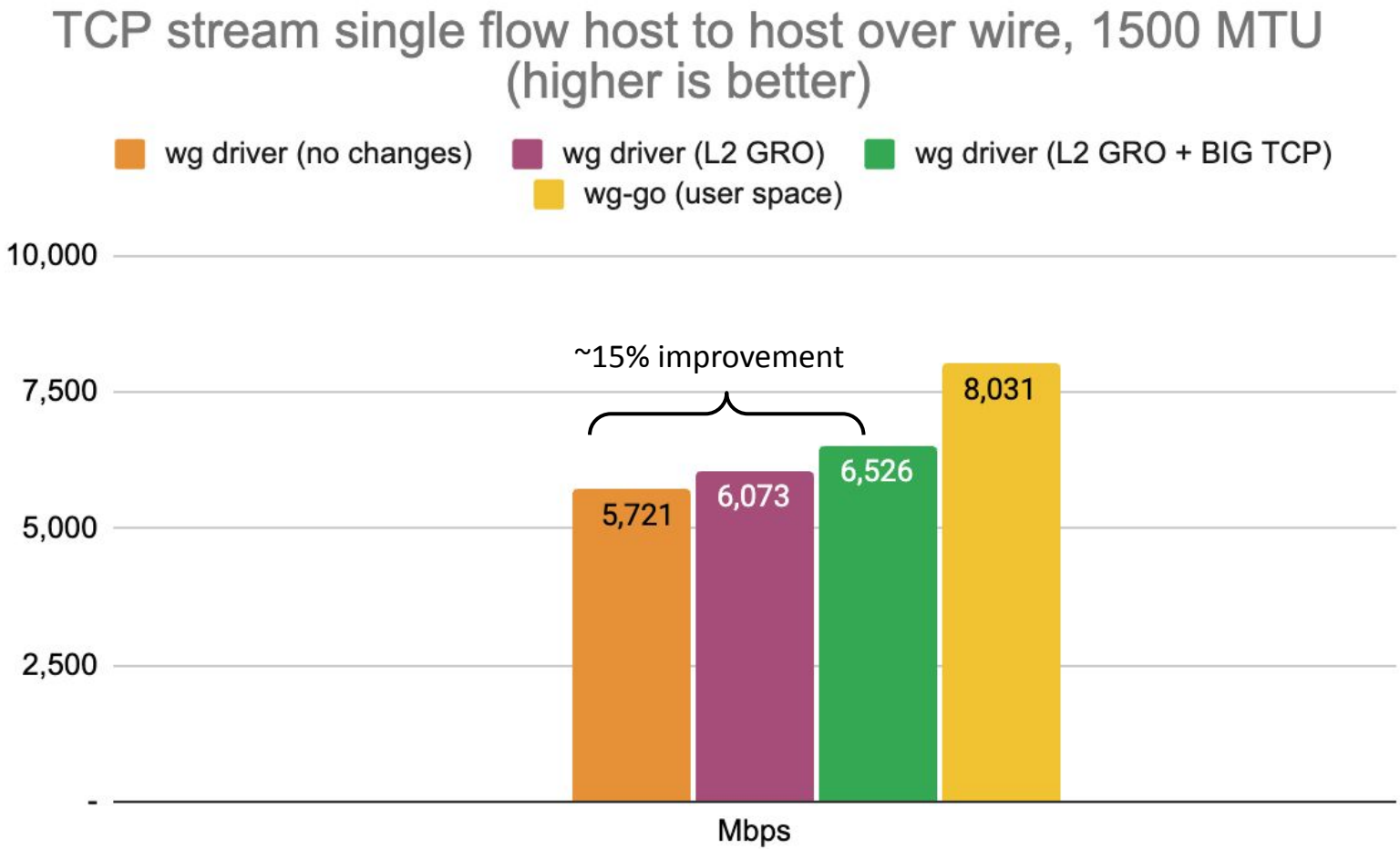
```c
static size_t wg_gro_candidate(struct sk_buff *skb)
{
        if (unlikely(skb->len < sizeof(struct message_header)))
                return false;
        if (SKB_TYPE_LE32(skb) == cpu_to_le32(MESSAGE_DATA) &&
            skb->len >= MESSAGE_MINIMUM_LENGTH)
                return true;
        return false;
}

struct sk_buff *wg_gro_receive(struct sock *sk,
                               struct list_head *head,
                               struct sk_buff *skb)
{
        struct wg_device *wg = sk->sk_user_data;
        int offset = skb_gro_offset(skb);

        if (!pskb_pull(skb, offset))
                return NULL;
        if (!wg_gro_candidate(skb))
                goto out;
        skb_mark_not_on_list(skb);
        PACKET_CB(skb)->ds = ip_tunnel_get_dsfield(ip_hdr(skb), skb);
        wg_packet_consume_data(wg, skb);
        return ERR_PTR(-EINPROGRESS);
out:
        skb_push(skb, offset);
        NAPI_GRO_CB(skb)->same_flow = 0;
        NAPI_GRO_CB(skb)->flush = 1;
        return NULL;
}
```

```c
int wg_socket_init(struct wg_device *wg, u16 port)
{
        struct net *net;
        int ret;
        struct udp_tunnel_sock_cfg cfg = {
                .sk_user_data   = wg,
                .encap_type     = 1,
                .encap_rcv      = wg_receive,
                .gro_receive    = wg_gro_receive,
        };
...
```

UDP socket registers GRO handler

GRO handler pushes data packets directly for decryption when GRO engine is invoked from phys dev

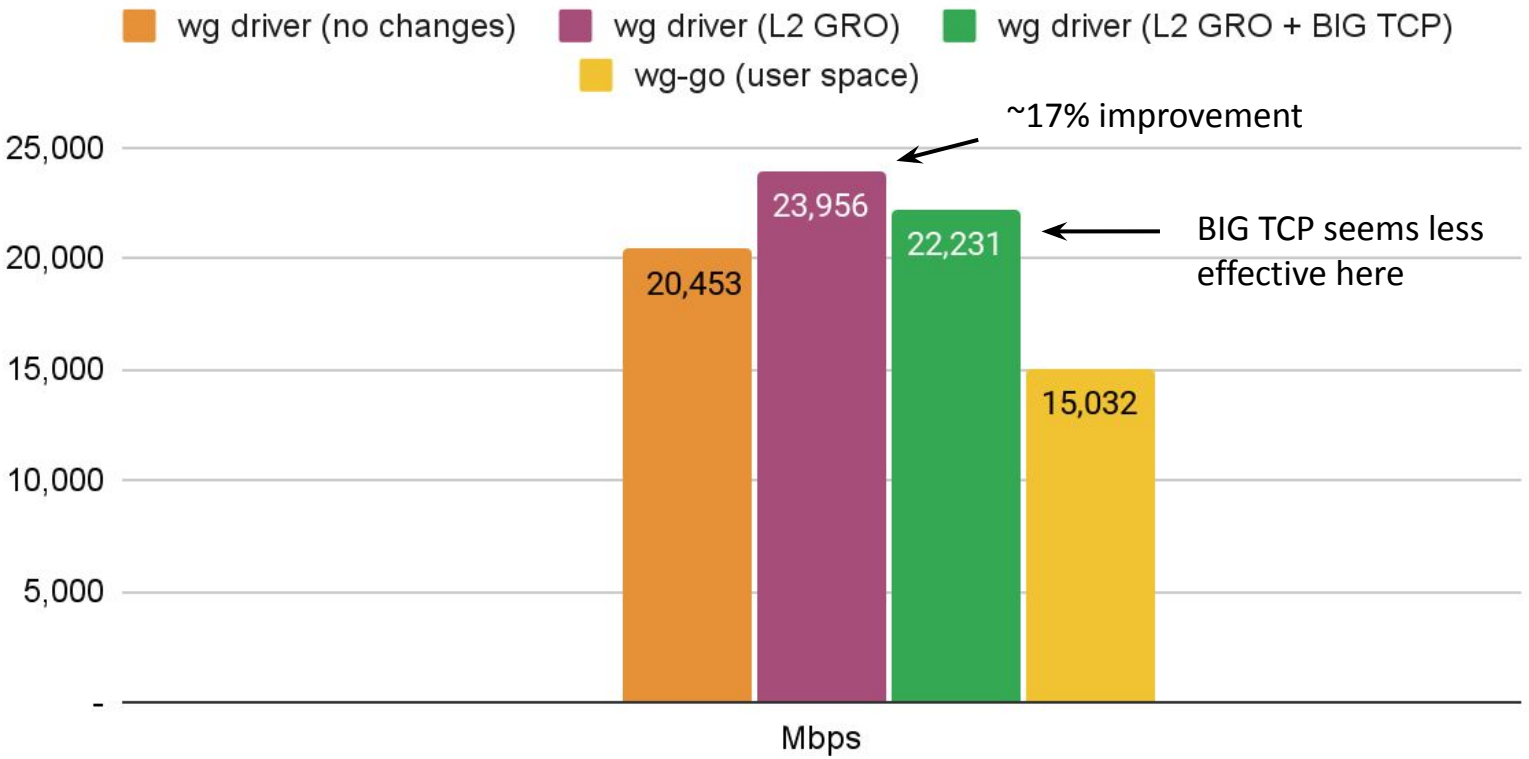
ESP GRO added INET_ESP_OFFLOAD Kconfig knob, do we need a similar Kconfig knob for WireGuard, or an attribute during device creation?



DB

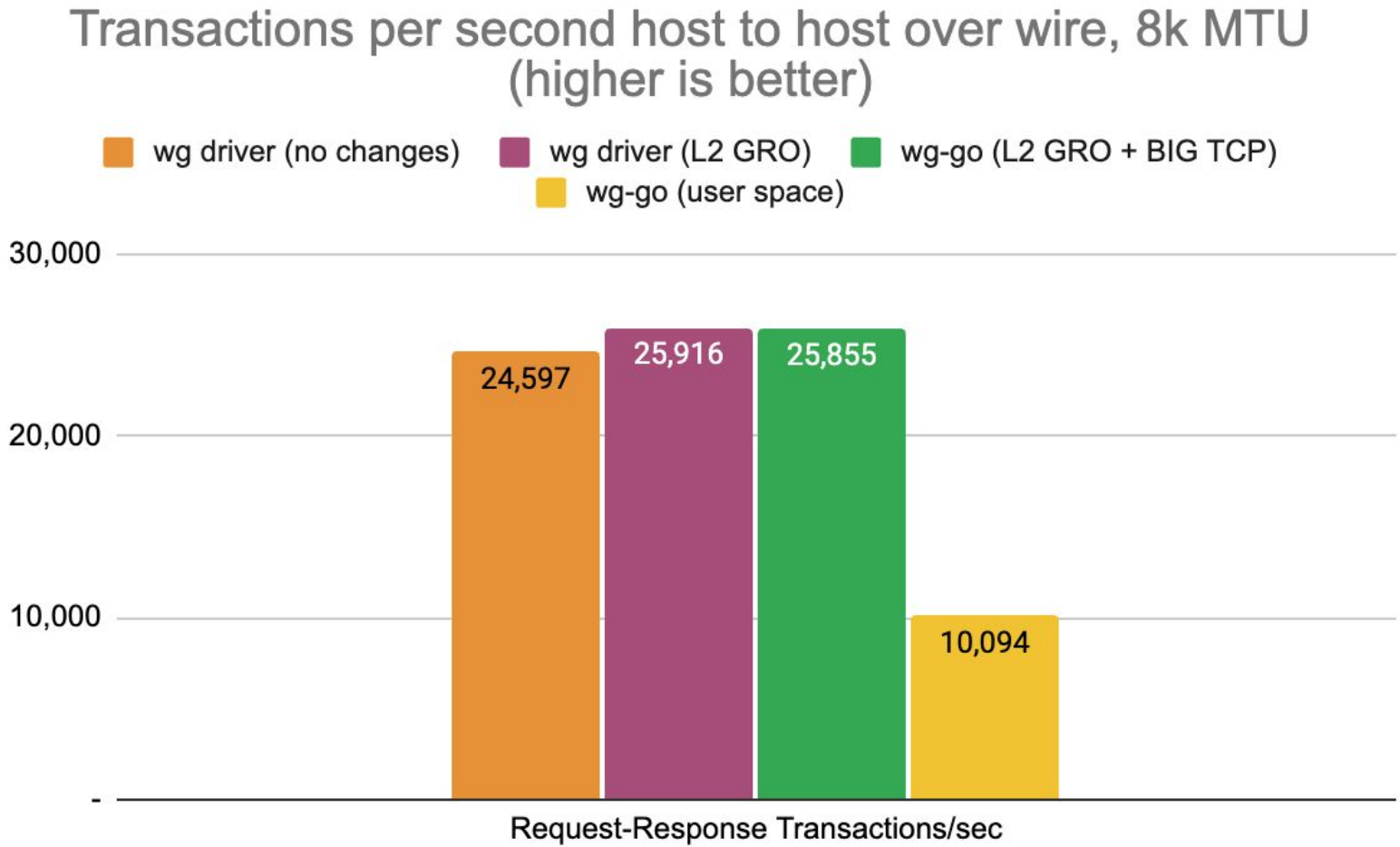
Back to back: AMD Ryzen 9 3950X @ 3.5 GHz, 128G RAM @ 3.2 GHz, PCIe 4.0, ConnectX-6 Dx, mlx5 driver, striding mode, LRO, 1500 MTU

TCP stream single flow host to host over wire, 8k MTU (higher is better)



DB

Back to back: AMD Ryzen 9 3950X @ 3.5 GHz, 128G RAM @ 3.2 GHz, PCIe 4.0, ConnectX-6 Dx, mlx5 driver, striding mode, LRO, 8k MTU



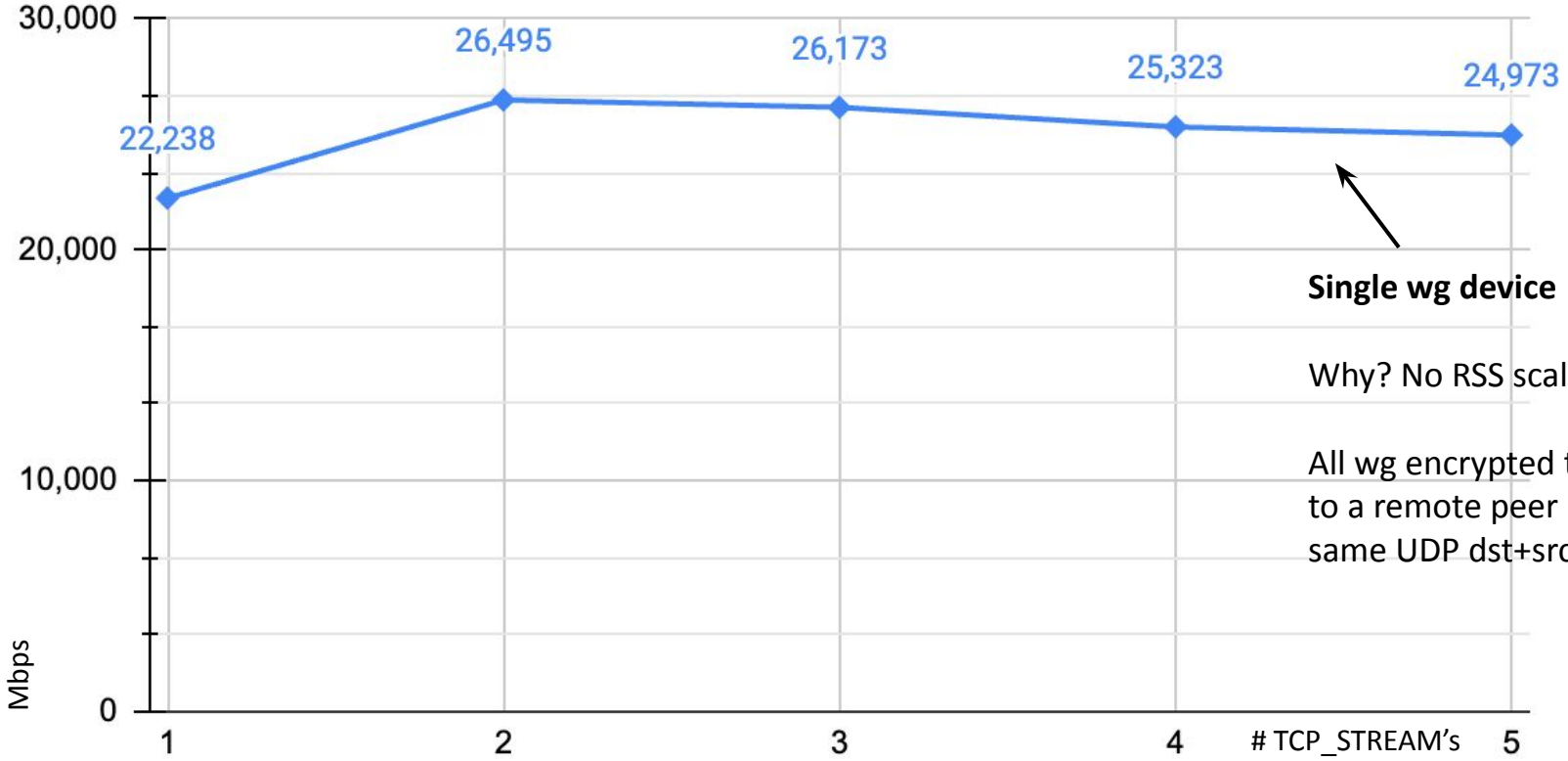
DB

Back to back: AMD Ryzen 9 3950X @ 3.5 GHz, 128G RAM @ 3.2 GHz, PCIe 4.0, ConnectX-6 Dx, mlx5 driver, striding mode, LRO, 8k MTU

What about multiple flows?

- Rationale: Cilium creates a single cilium_wg0 device for all east-west Pod/Pod traffic
  - BPF datapath basically bpf_redirect()’s to cilium_wg0
- Question: How well does it scale when multiple parallel flows hit cilium_wg0?

TCP stream multi flow host to host over wire, 8k MTU (higher is better)



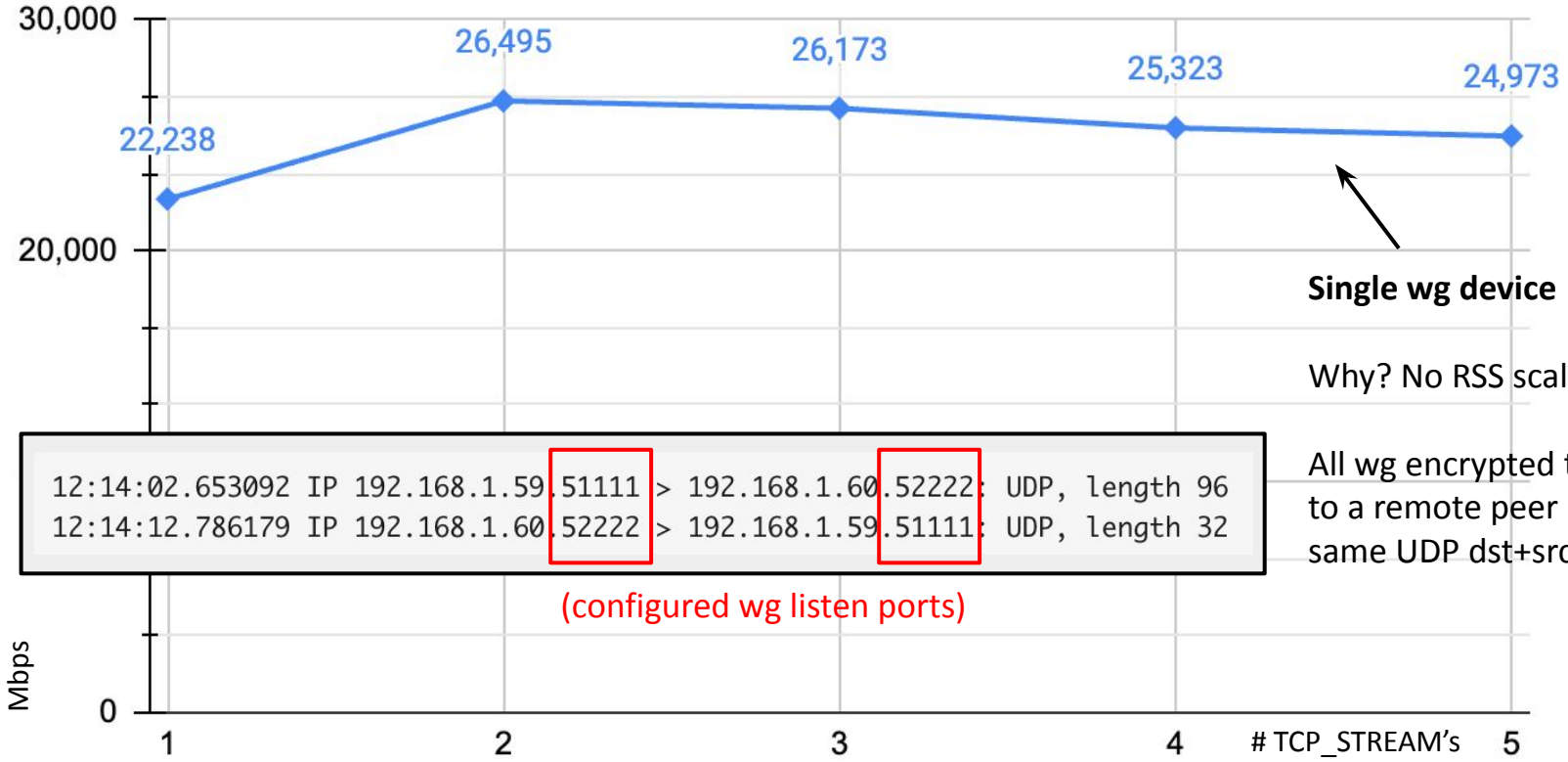
Single wg device

Why? No RSS scaling!

All wg encrypted traffic to a remote peer has the same UDP dst+src IP/port

Back to back: AMD Ryzen 9 3950X @ 3.5 GHz, 128G RAM @ 3.2 GHz, PCIe 4.0, ConnectX-6 Dx, mlx5 driver, striding mode, LRO, 8k MTU

TCP stream multi flow host to host over wire, 8k MTU (higher is better)



Back to back: AMD Ryzen 9 3950X @ 3.5 GHz, 128G RAM @ 3.2 GHz, PCIe 4.0, ConnectX-6 Dx, mlx5 driver, striding mode, LRO, 8k MTU

What about multiple flows?

- Potential improvements?
  - Creating **multiple WireGuard devices under a bond** and then load-balance based on hash
    - Currently not possible due to bond being an L2 device and WireGuard L3
    - Missing .ndo_set_mac_address but also refuses after dummy implementation
    - Probably new bond mode needed (?) or fixups when bond/slave device are both in NOARP mode. Would be nice for bpf_redirect() in datapath.
  - Creating **multiple WireGuard devices and load-balance via multipath next hops**
    - Works in terms of routing, but WireGuard reveals unexpected behavior

```
# ip r
default via 192.168.1.1 dev enp5s0 proto dhcp src 192.168.1.119 metric 100
10.0.0.0/24 dev enp10s0f0np0 proto kernel scope link src 10.0.0.2
10.1.0.1
        nexthop dev wg0 weight 1
        nexthop dev wg1 weight 1
```

What about multiple flows?

- Several WireGuard devices on same node, options tried:
  - Different listen-port but otherwise same peer key/endpoint/allowed-ip settings?
    - Currently buggy: allowed-ips overridden/removed to “none”

```
peer: xvYlNOXRTf3OcaylpH5EFgEYluKY0Zp1bZkFEDIfE1I=
  endpoint: 10.0.0.1:9001
  allowed ips: (none)
```

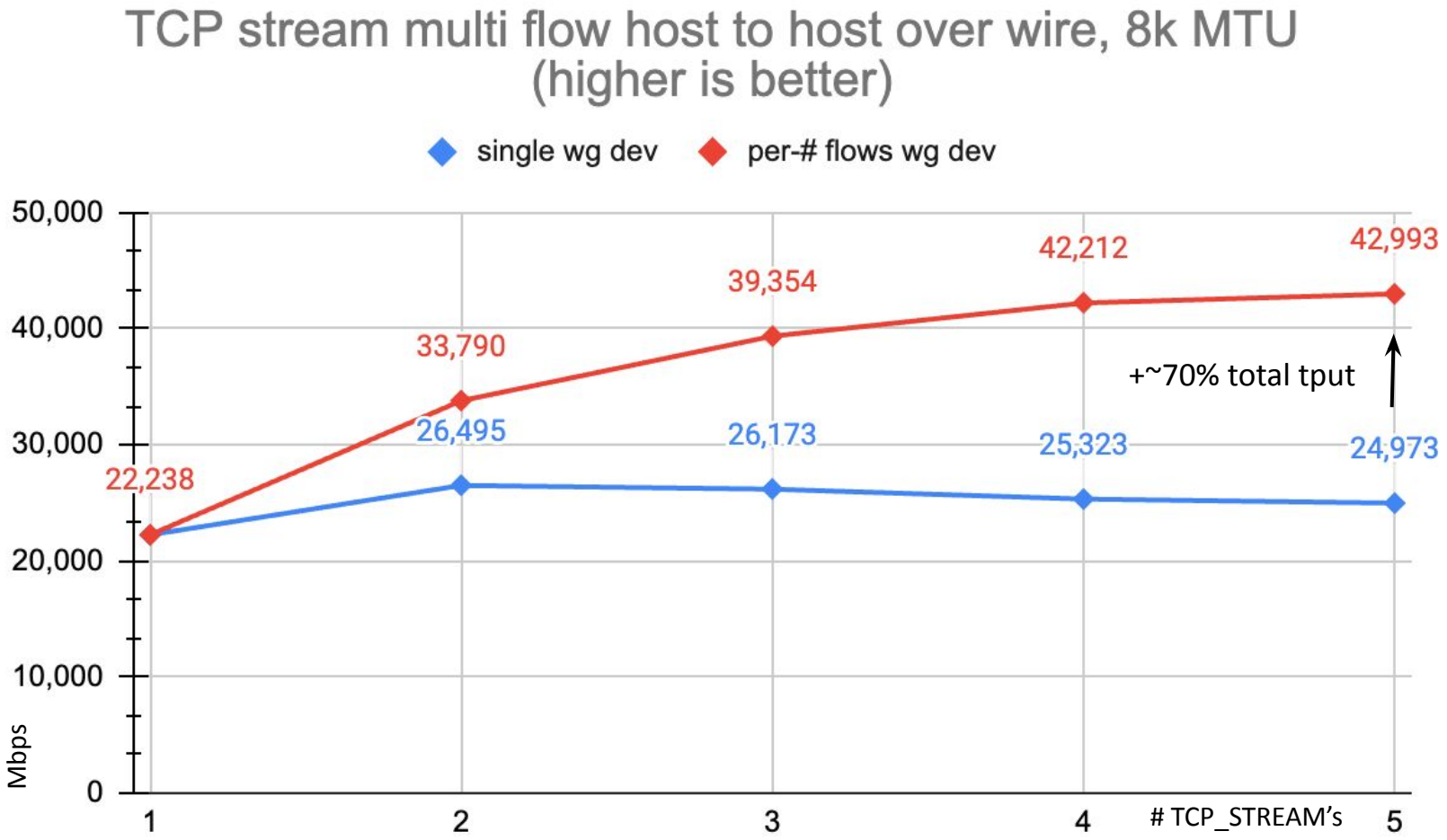
  - Different listen-port and different key-pairs, but same endpoint/allowed-ip settings?
    - Same behavior as above (needs fixing)

What about multiple flows?

- Several WireGuard devices on same node, options tried:
  - Different listen-port but otherwise same peer key/endpoint/allowed-ip settings?
    - Currently buggy: allowed-ips overridden/removed to “none”

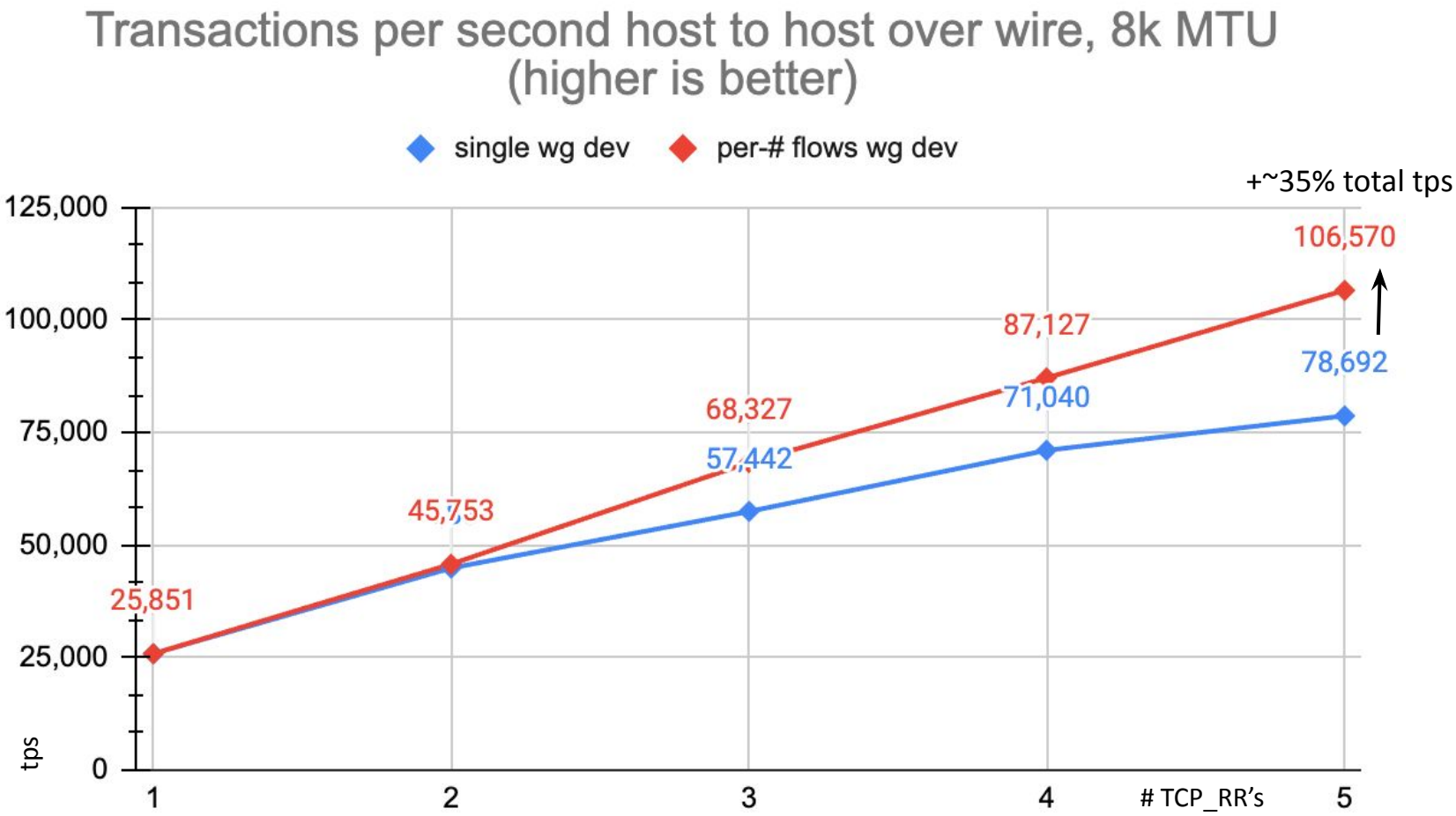
```
peer: xvYlNOXRTf3OcaylpH5EFgEYluKY0Zp1bZkFEDIfE1I=
  endpoint: 10.0.0.1:9001
  allowed ips: (none)
```

  - Different listen-port and different key-pairs, but same endpoint/allowed-ip settings?
    - Same behavior as above (needs fixing)
- What about a WireGuard mode to have inner hash part of outer src port?
  - Downside: Exposes information of different flows, assumes single wg dev per host
- Workaround for test: all properties different (key-pairs/endpoint/allowed-ip)
  - This works for testing the idea, but is not practical for production



DB

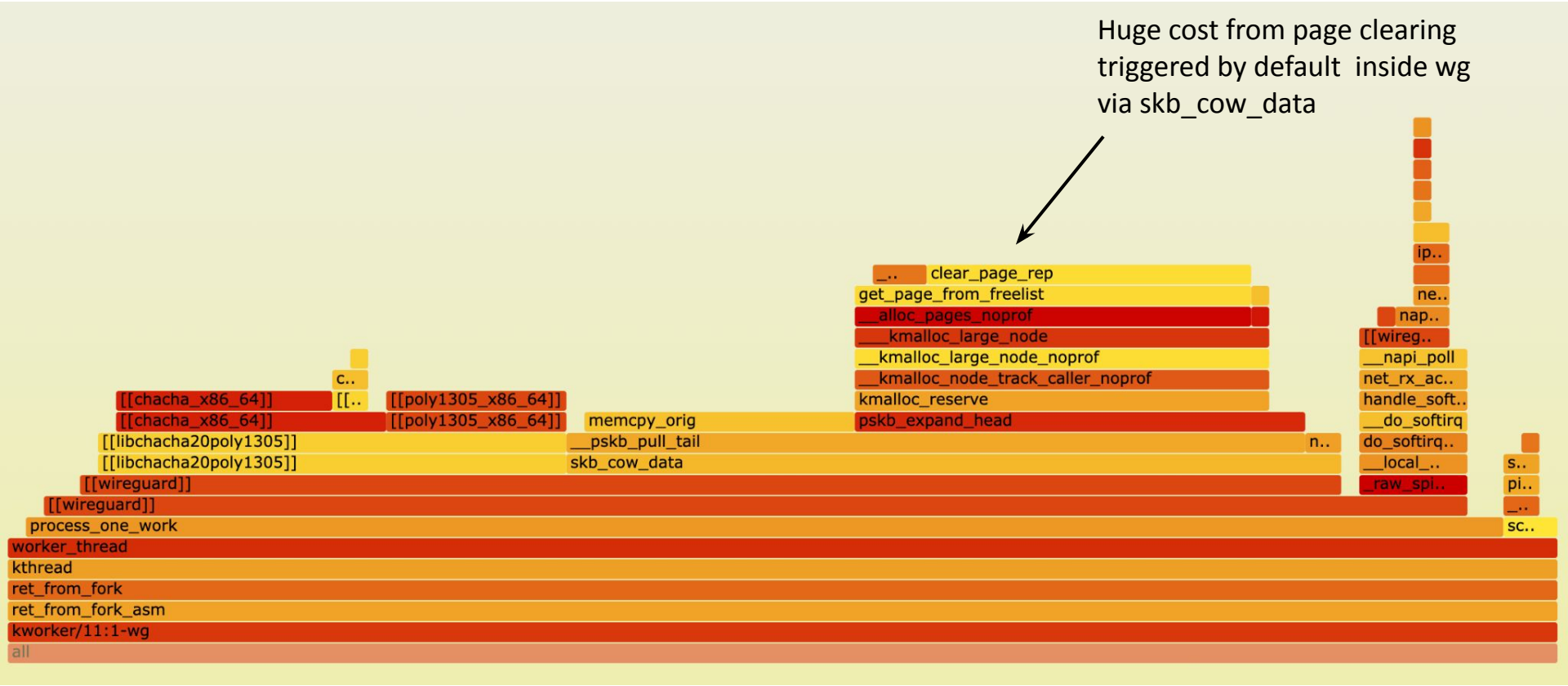
Back to back: AMD Ryzen 9 3950X @ 3.5 GHz, 128G RAM @ 3.2 GHz, PCIe 4.0, ConnectX-6 Dx, mlx5 driver, striding mode, LRO, 8k MTU



DB

Back to back: AMD Ryzen 9 3950X @ 3.5 GHz, 128G RAM @ 3.2 GHz, PCIe 4.0, ConnectX-6 Dx, mlx5 driver, striding mode, LRO, 8k MTU

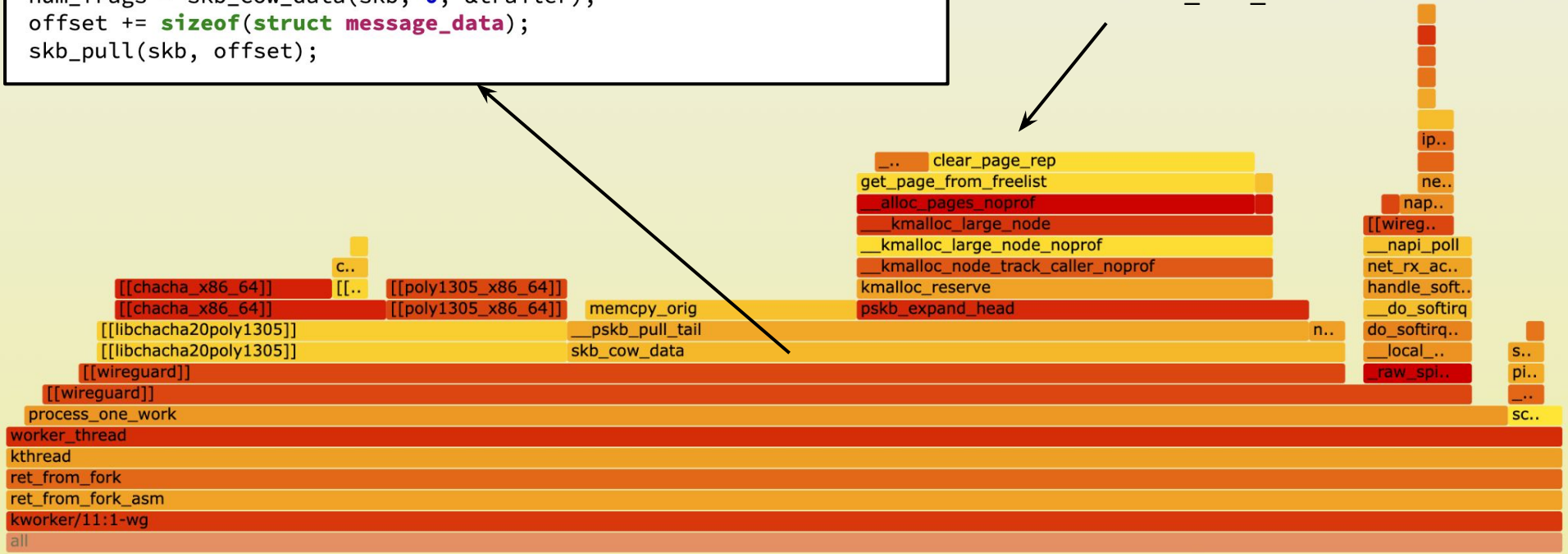
Other findings from testing:



wg: decrypt_packet() :

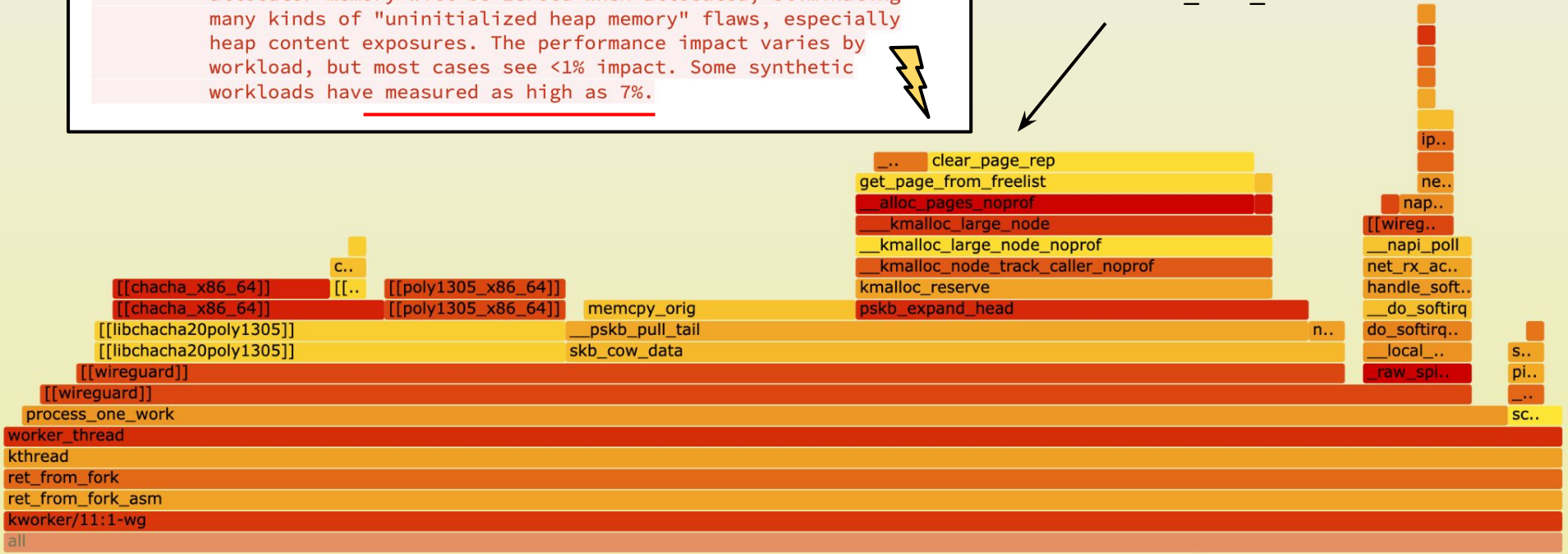
```
/* We ensure that the network header is part of the packet before we
 * call skb_cow_data, so that there's no chance that data is removed
 * from the skb, so that later we can extract the original endpoint.
 */
offset = -skb_network_offset(skb);
skb_push(skb, offset);
num_frags = skb_cow_data(skb, 0, &trailer);
offset += sizeof(struct message_data);
skb_pull(skb, offset);
```

Huge cost from page clearing triggered by default inside wg via skb_cow_data



```
config INIT_ON_ALLOC_DEFAULT_ON
        bool "Enable heap memory zeroing on allocation by default"
        depends on !KMSAN
        help
          This has the effect of setting "init_on_alloc=1" on the kernel
          command line. This can be disabled with "init_on_alloc=0".
          When "init_on_alloc" is enabled, all page allocator and slab
          allocator memory will be zeroed when allocated, eliminating
          many kinds of "uninitialized heap memory" flaws, especially
          heap content exposures. The performance impact varies by
          workload, but most cases see <1% impact. Some synthetic
          workloads have measured as high as 7%.
```

Huge cost from page clearing triggered by default inside wg via skb_cow_data



Other TODO items

- Once RSS is solved, experiment with CPU locality in terms of encryption/decryption
- __cacheline_group_begin/end for RX/TX mostly data in hot path
- Atomic queue counter shared across CPUs
- Complete removal of wg driver segmenting skbs?
  - Probably not possible due to nonce as part of wg header

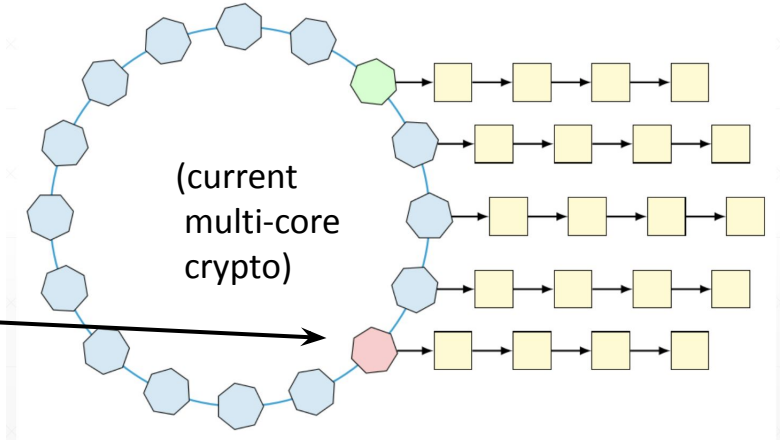
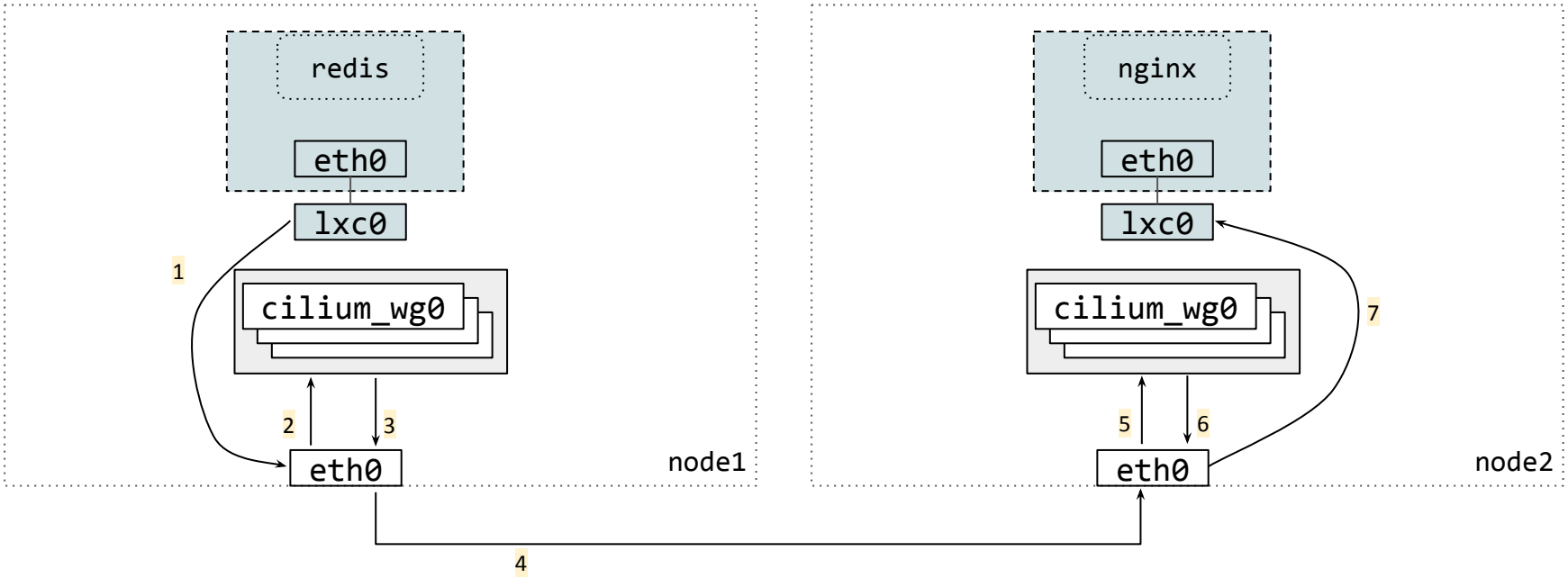


Image source: https://www.wireguard.com/talks/lpc2018-wireguard-slides.pdf

Cilium WireGuard integration: future? (~KubeCon'24)



Acknowledgements

Jason A. Donenfeld (WireGuard)
Jordan Whited & James Tucker (WireGuard-go improvements)
Sebastian Wicki (initial Cilium integration co-author)
Cilium, netdev & BPF communities

Thanks! Questions?

Cilium + WireGuard: https://docs.cilium.io/en/stable/security/network/encryption-wireguard/
PoC code: https://github.com/cilium/linux/commits/pr/wg