Optimizing `sk_msg` for Socket Map

Linux Kernel Maintainer

Cong Wang, xiyou.wang@gmail.com

https://wangcong.org/about/

LSF/MM/BPF 2025, March 26, 2025

What is `sk_msg`?

- `sk_msg` is a kernel data structure for socket layer messaging
- Similar to the traditional `sk_buff` but much simpler
- Core component of the socket map (sockmap) infrastructure
- Exposed directly to (some) eBPF socket map programs

Socket Map and `sk_msg`

- Socket map: eBPF map type that stores socket references
- `sk_msg` works with sockmap to:
  - Intercept messages at socket layer
  - Apply eBPF programs for verdict decisions
  - Redirect messages between sockets
- Used extensively in socket-level policying and redirection

What is Cilium enable-sockops?

- `enable-sockops` is a Cilium feature that leverages sockops to bypass TCP stack
- It is removed in Cilium v1.13 and later
- It hijacks *all* the local TCP sockets and splices the data in between the sockets
- Including loopback TCP sockets, sockets between containers on the same host
- This is a very elegant optimization and independent of Cilium
    - Remember TCP friends?

Performance Issue #1

- Small messages perform worse than traditional TCP!
    - TCP has sophisticated batching, e.g. `release_sock`
    - TCP protocol itself supports batching (e.g. Nagle's algorithm)
    - `sk_data_ready()` is batched by `sk->sk_rcvlowat`
    - `sk_buff` is highly optimized by network developers for decades
    - `sk_msg` is a simple structure, with no batching mechanism (as of now)
    - `lock_sock` is really a performance killer

Performance Issue #2

- Data copying on the redirection path:
    - Ingress (relatively okay):
        - `tcp_bpf_recvmsg` takes `sk_msg`
        - `sk_psock_skb_ingress_enqueue()` moves data from `sk_buff` to `sk_msg`
    - Egress (relatively bad):
        - `->sendmsg()` moves data from `sk_buff` to `struct msghdr`
        - `->sendmsg()` copies data from `struct msghdr` to a TX `sk_buff` again
        - Essential, it is not easy to reuse `sk_buff` from RX for TX

Recap of Data Path

In order of complexity:

- TX -> RX: `sk_msg` -> `sk_msg`
- RX -> RX: `sk_buff` -> `sk_msg`
- TX -> TX: `sk_msg` -> `sk_buff`
- RX -> TX: `sk_buff` -> `sk_msg` -> `msghdr` -> `sk_buff`

Batching Ingress `sk_msg`

- Excellent work by Zijian Zhang
- Introduces kworker-based message corking mechanism
- Adds a backlog queue to accumulate messages before delivery
- Intelligent notification based on buffer fullness, message size, and TCP settings
- Significantly improves throughput by reducing wake-ups and lock contention

What About Egress?

- Egress is much more complex
  - `->sendmsg()` is invoked directly but serves `sendmsg()` syscall too
  - TCP `->sendmsg()` could coalesce packets
  - TCP implements Nagle's algorithm

RX and TX Contexts

- Networking RX is typically in softirq context (unless in `ksoftirqd`)
- Networking TX is typically in process context, nearly synchronous with `sendmsg()`
- `skb->data` is already past all headers at the point of `->sk_data_ready()`
- `struct sk_buff` has a layer-specific control buffer `skb->cb`
- Not to mention, `skb->dst`, `skb->sk`, `skb->sk_mark`, etc

Idea: Scrub `sk_buff` and Replace `->sendmsg()`

- Scrub RX `sk_buff` properly and place it directly to `sk->sk_write_queue`
    - We already have `skb_scrub_packet()`
    - Replace `->sendmsg()` with direct skb queueing
    - Eliminate the data copying

Idea: Lockless Socket Accounting

- Socket accounting functions on receive side:
  - `sk_mem_charge()` / `sk_mem_uncharge()`
  - `atomic_add()` / `atomic_sub()` on `sk->sk_rmem_alloc`
  - `__sk_rmem_schedule()` for memory reservation
- Those counters are atomic anyway
- This is possible in this simple case
- Need to satisfy `inet_sock_destruct()`

Idea: Getting rid of `psock->work`

- This is essentially hard due to atomic context
- `lock_sock` itself is blocking too (shrug)
- Potentially there are more blocking operations on the path

Idea: One `__sk_buff` to Rule Them All

- Too late to change some eBPF socket programs due to compatibility
- But we can always introduce new programs
- Apply "message" verdicts to `__sk_buff` directly
- Goal: No more `sk_msg` on all the data path

Thank You!

Questions?

Contact: Cong Wang xiyou.wangcong@gmail.com

https://wangcong.org/about/