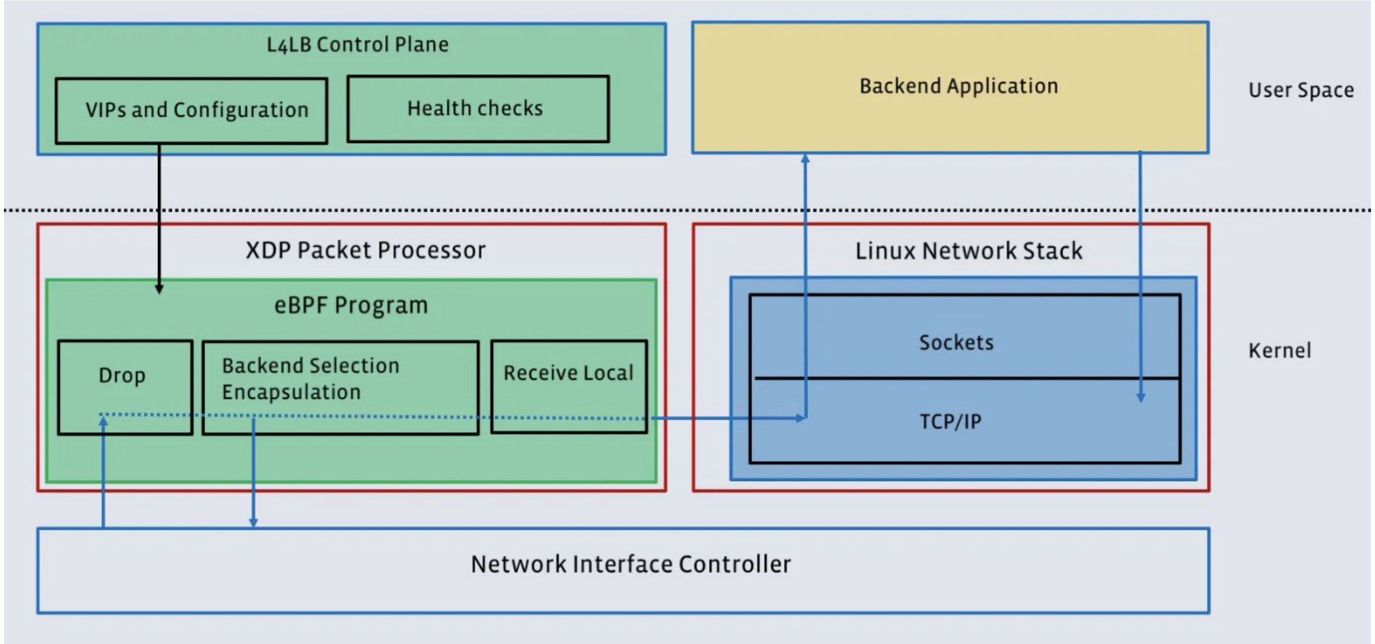
Thrift RPC parsing using BPF

David Wei
Meta

Seems like a good idea?

- Meta DC traffic is mostly RPC
- In-kernel consumers
  - Offload hot work
  - Drop work early
  - Reduce overheads
- Produce/consume *object* streams instead of *byte* streams

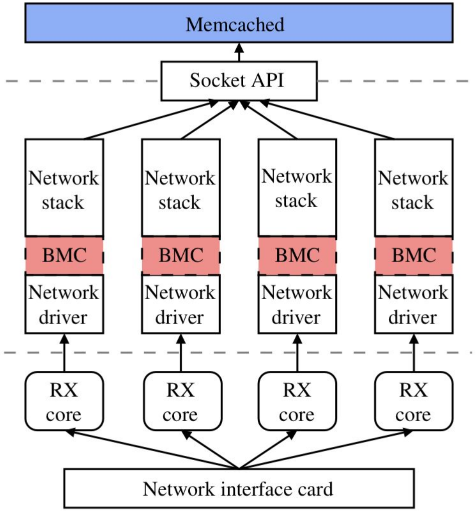
Katran: BPF L4 LB

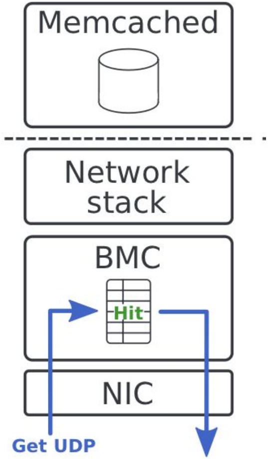


Katran: BPF L4 LB

- Stateless
- Simple
  - Hashing
  - Read configuration
  - Forwarding
- Complexity in userspace control plane
  - Configuration
  - Observability
  - Health Monitoring
- Skips TCP stack, no unneeded copies

BMC: Memcache in-kernel cache





(a) Lookup

BMC: Memcache in-kernel cache

- Open source Memcache
- Single server
- GET requests use UDP
- Bounded key + value lengths
- Unknown configuration
  - Is mitigations=on?
- https://www.usenix.org/conference/nsdi21/presentation/ghigoff

BMC: Memcache in-kernel cache

- There is no magic 😔
- Work needs to be done *somewhere*
- Where do efficiency wins come from?

Where does performance come from?

- Avoid syscalls
- Fewer context switches
- Avoid copies
- Skip networking stack
  - Especially if request ends up being dropped in userspace
- Reduce locking
- Increase locality
- Specialisation - HW offloads

Why Meta Memcache might not work?

- Ship of Theseus
- Distributed service
- Requests are RPC (Thrift) over TCP
- Lots of userspace code
  - ACL
  - Logging
  - Overload protection
  - Just usual userspace spaghetti...
- High rate of change

Why Meta Memcache might not work?

- What might a hot in-kernel cache save?
  - Two copies
  - Key sharding, maybe
  - Syscalls don't matter (for us)
- But only for GET
- Now need to de/serialise Thrift
- And still have to do logging/etc
- Have to always trade off complexity/effort vs efficiency gains
  - Can this be solved with more $

Zero copy spoils the party?

- Various mechanisms of doing ZC Tx
- New, non-page flipping mechanism for doing ZC Rx incoming
- Removes copies across kernel/user boundary
- Lets userspace do the things that need doing anyway...

Header/data split

- New zero copy features built on top of NIC HW header/data splitting
- Headers go into kernel memory
- Payload go into user/device memory
- Set up HW Rx queues and fill them with DMA mapped user/device memory
- NIC doesn't care what's in the Rx queue descriptors

What about large Thrift requests?

- Most of the request is an opaque data payload
- Destined for e.g. NVMe flash, GPU memory
- Can parsing and handling Thrift in-kernel enable us to intelligently split the payload out?

Thrift is inline

- As opposed to split control/data plane
- *Something* needs to read L7 headers to decide *what* to do
- Kernel could do this:
  - NIC -- DMA --> kernel memory
  - Parse and handle Thrift protocol
  - Kernel memory -- DMA --> PCIe

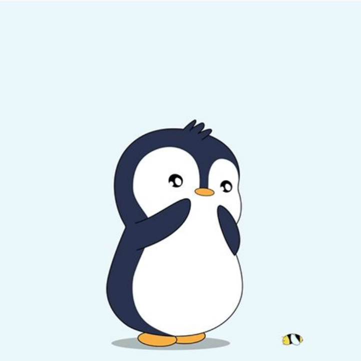
BPF in io_uring

- Patches from Pavel Begunkov
- BPF_PROG_TYPE_IOURING
- Register BPF programs w/ io_uring instance
- Issue IORING_OP_BPF requests
- Do (almost) anything io_uring can
- https://github.com/isilence/linux/commits/bpf_v3/

Zero copy spoils the party, again?

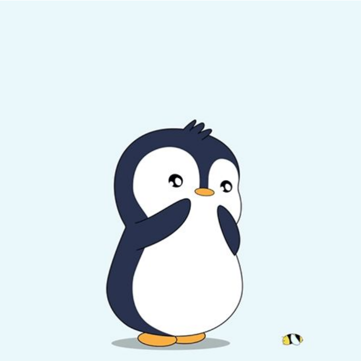
- NIC -- DMA --> user memory
- Parse and handle Thrift
- User memory -- DMA --> PCIe
- Maybe save syscalls?
  - But, they don't matter for us

(Bad) Idea coming?



- What if we can split L7 headers?

(Bad) Idea coming?

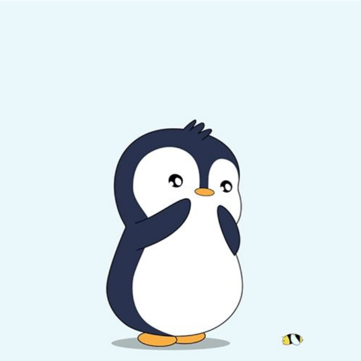


- What if we can split L7 headers?
- Using BPF?

(Bad) Idea coming?

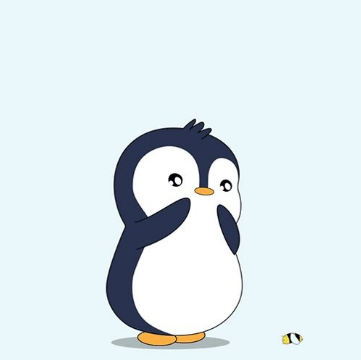
- What if we can split L7 headers?
- Using BPF?
- Really *really* early on?

(Bad) Idea coming?



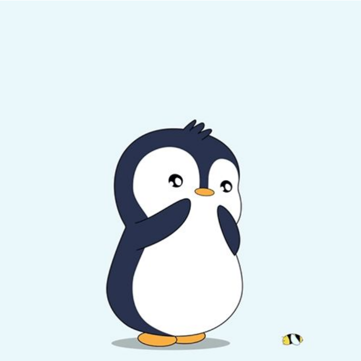
- What if we can split L7 headers?
- Using BPF?
- Really *really* early on?
- While still in NIC buffers, before DMAing into Rx queue descriptors?

(Bad) Idea coming?



- What if we can split L7 headers?
- Using BPF?
- Really *really* early on?
- While still in NIC buffers, before DMAing into Rx queue descriptors?
- And then handle L7 headers, using BPF?

(Bad) Idea coming?

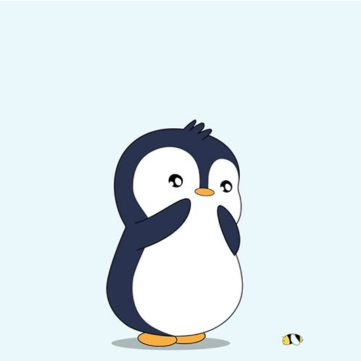


- NIC -- DMA --> user memory
- Parse and handle Thrift
- User memory -- DMA --> PCIe

(Bad) Idea coming?

- NIC -- DMA --> user memory
- Parse and handle Thrift
- User memory -- DMA --> PCIe

(Bad) Idea coming?



- Parse and handle Thrift
- L7 payload: NIC -- DMA --> user memory
- ~~User memory -- DMA --> PCIe~~

Can it be done?

- Different transport than TCP?
- Able to reorder packets at the NIC buffer level
- RPC protocol must support streaming deserialisation
- The right IPUs?
- BPF CPUs?
- Needs PSP?

More importantly: should it be done?

- We're never going to not use Thrift RPC
- We're also (probably) never going to rewrite services
  - We will run ~~PHP~~Hack and Python until the very end
- Needs new SKU w/ IPU and widescale deployment
- Hardware projects take *years* and the world changes in the meantime
- Observability + security

More importantly: should it be done?

- Are there easier things to do?
- Can we increase utilisation of very expensive GPUs?
- Is this easier than rewriting software?
- Do I live in a big tech bubble?