Spectres never die - they turn into zombie

MDS performance impact, repolines and sparse performance improvements

NetConf, Boston 2019

Outline

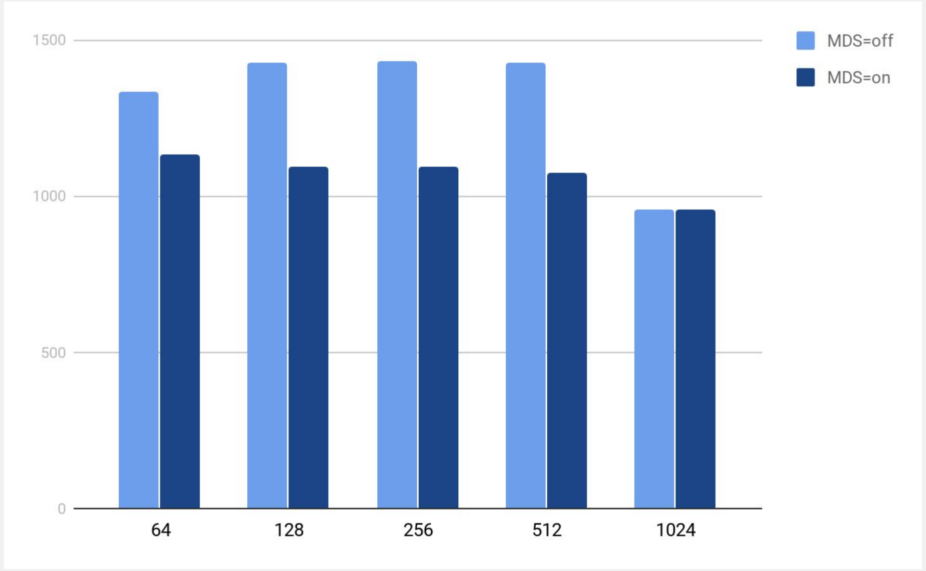
- ZombieLoad/MDS performance impact…
- … but we are still catching-up with retpoline
- Other future possible misc performance improvements

What is MDS (aka zombieLoad) after all?

- Just another bunch of x86 microarchitecture issues
- User space can speculatively access data into the store buffer, load buffer, load port breaking the process or even guest boundaries
- Cross hyper-thread: with SMT-on complete mitigation is not possible
- Mitigation: flush the affected buffer on context switch

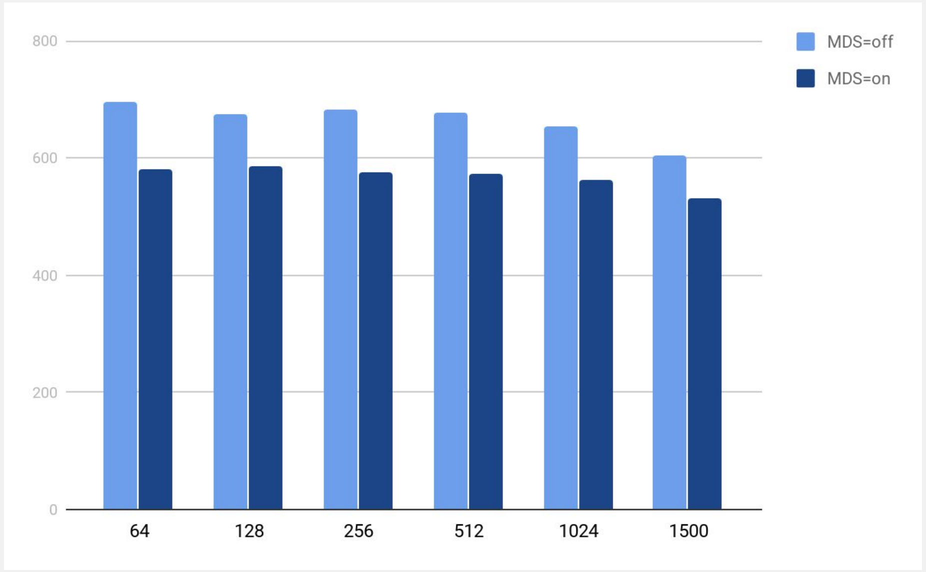
UDP RX performances

64 bytes ipv4 packets, single RX queue, different packet sizes



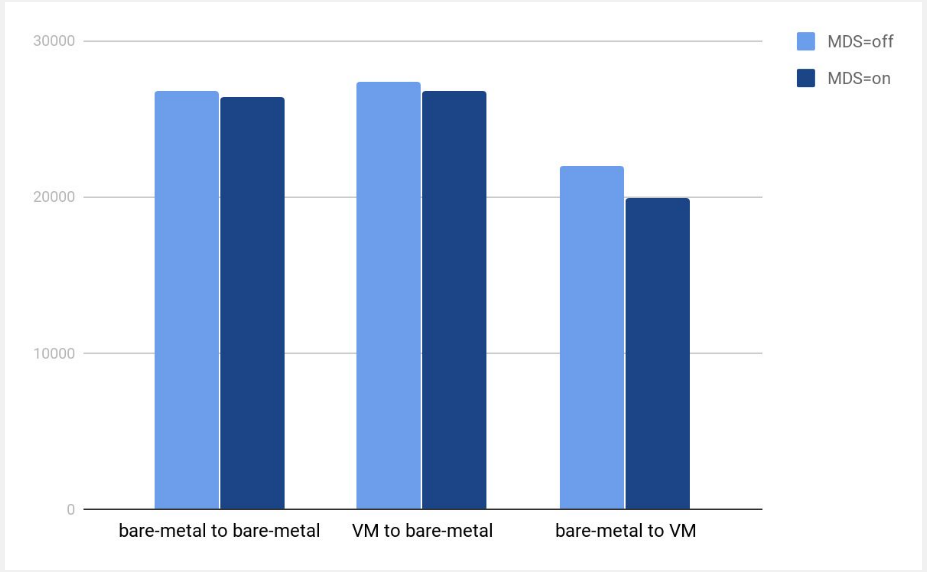
UDP TX performances

64 bytes ipv4 packets, single RX queue, different packet sizes



TCP performances

Single flow stream workload



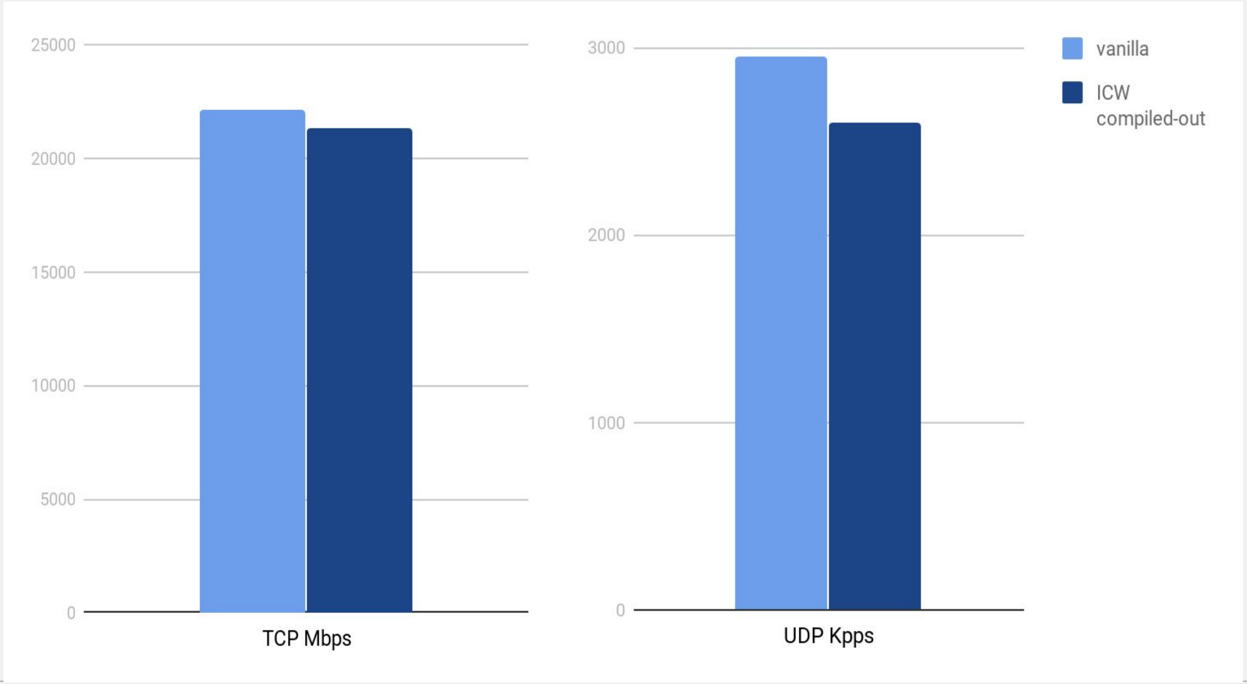
Still dealing with retpolines [1]

We avoid some indirect calls via the indirect call wrappers:

- GRO
- Part of the RX path
- Part of the TX path
- Other users: csum, ipvs

Indirect call wrapper effects

Kernel 5.2.0-rc1



Still dealing with retpolines [2]

Other possible INDIRECT_CALL_WRAPPER targets:

- skb->destructor
- sk_proto->{send,recv}msg
    - sock->ops->{send,recv}msg duplication (ipv4 vs ipv6) needed?

Indirect call pain-points

- Code uglification
- Un-addressable call-sites:
  - ndo_start_xmit, ndo_select_queue, … ?
- Device drivers:
  - mlx5(!!!) had 2 indirect calls per packet in fast path (even XDP!!!)
  - What about others NICs/vendors?

Skb header recycle

- napi_gro_frags() allows recycle the whole skb on merge
- Most [hi-perf] drivers use napi_gro_receive()

We could do something similar there:

- recycle [merged] the sk_buff into the napi struct
- A new napi allocator tries to fetch the recycled sk_buff first
    - Still need per driver patch
    - can further optimize the recycling, clearing only the fields touched by the GRO code and the driver

Bulk skb header allocation/free

Once we have a napi skb hdr allocator in place we can additionally

- Bulk alloc skb headers when the cache is empty
- Allow dev_kfree_skb() to recycle skbs to the cache
- Bulk free skb headers when recycling exceeds the cache limit

Still messing with costly skbs

zeroing (almost) all skb after the allocation is costly, we can:

- Shrink the struct
- Move fields at the end, control validity with bits, ala ‘extensions’, for less commonly used fields, e.g.
    - _nfct (the ptr part), secmark, priority, vlan_{proto,tci}, inner_{}_header, inner_{protocol,ipproto}
    - Will clear ‘only’ 3 CL per skb, instead of current 4

Shrink skb again

Some possible candidates:

- skb->truesize: 16 bits counter of 256b units, open points:
  - loss of precision
  - How to deal with ‘magic’ truesize values? (e.g. tcp pure ack)
- skb->data: 32 bits offset vs head, with set and get helpers
  - It’s a very invasive change

Shrink skb again [II]

- skb->cb[]: resize it to 40 bytes, add cb2[] at end
  - Requires some refactoring of current uses
  - WiFi drivers can be a problem
- Limiting the max value of some common fields (skb_iif, mac_len, ...)

THANK YOU