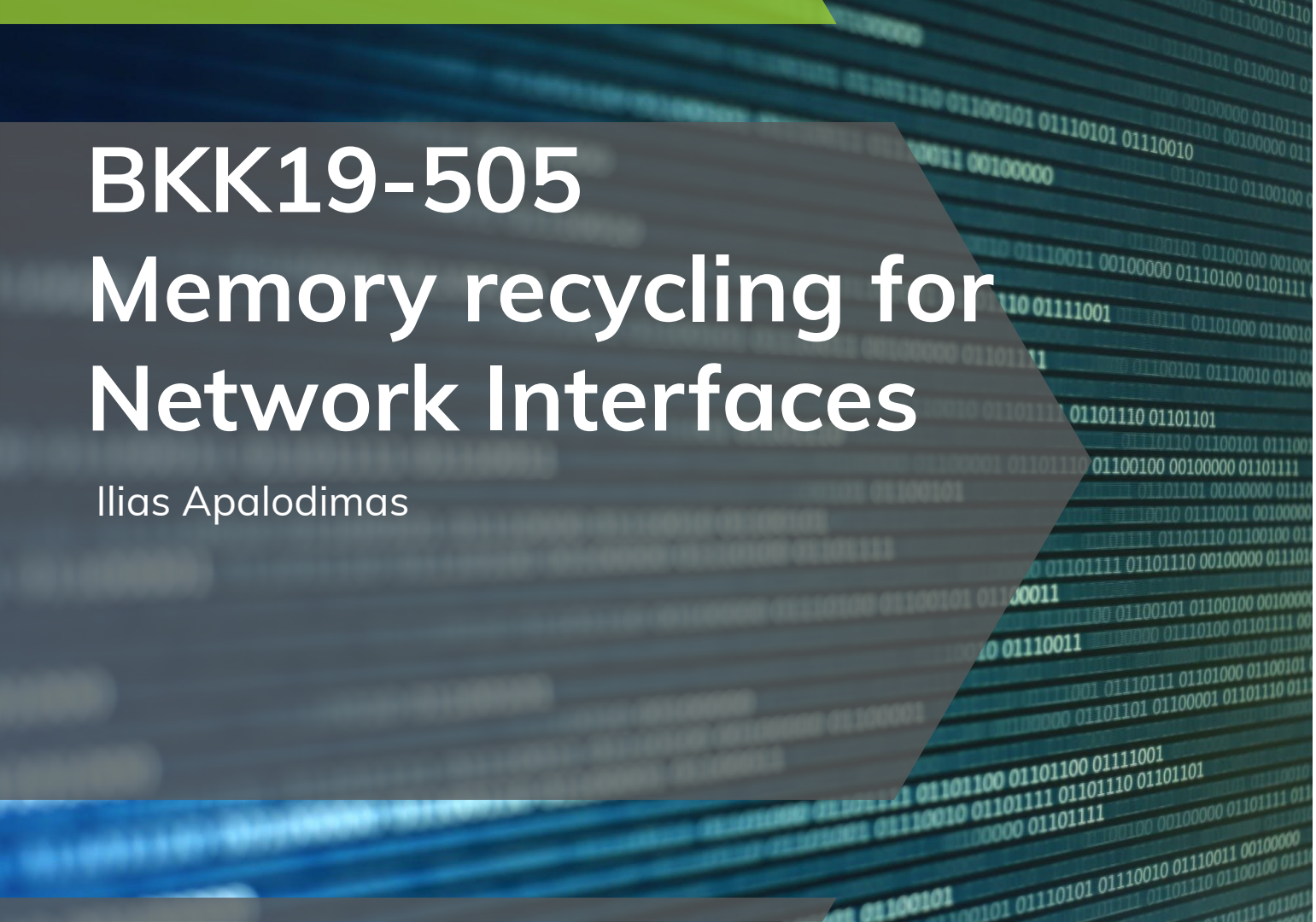
BKK19-505 Memory recycling for Network Interfaces

Ilias Apalodimas



Linaro

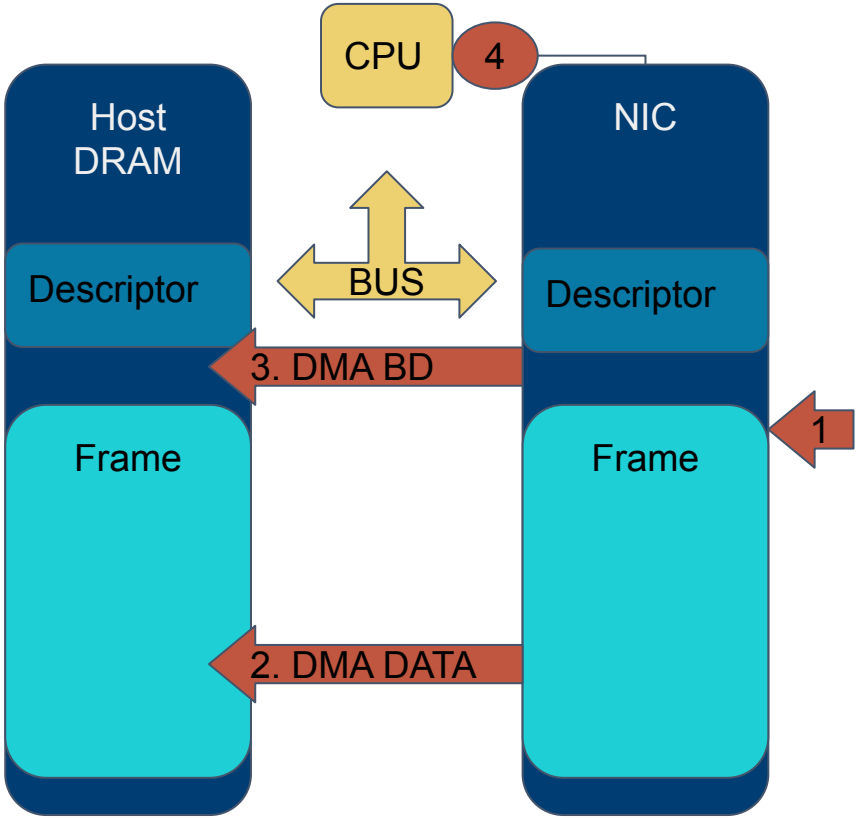
Why change it

- SKB recycling removed in the past, attracted little interest over time
- Many drivers are doing similar page recycling tricks themselves and there's no common code, everyone is doing something similar
- Right now driver authors end up with ‘skb-based’ drivers and shoehorn XDP support
  - This can vary from a simple change up to a full DMA/memory management rewrite
- The model should be the other way around, drivers should be targeting some raw data buffer. If the buffers get wrapped into an SKB or an XDP frame the driver shouldn't know or care
- Try to provide an API for NIC drivers with DMA mapping capabilities

Recycling reasons

- High speed (> 40gbit) are challenging the page allocator
- Constantly allocating and freeing buffers, which we know we'll need, is expensive and can be avoided especially at high speeds
- With the increasing use of SMMU/IOMMU, the cost of mapping and unmapping buffers is going to go up

NIC Rx path



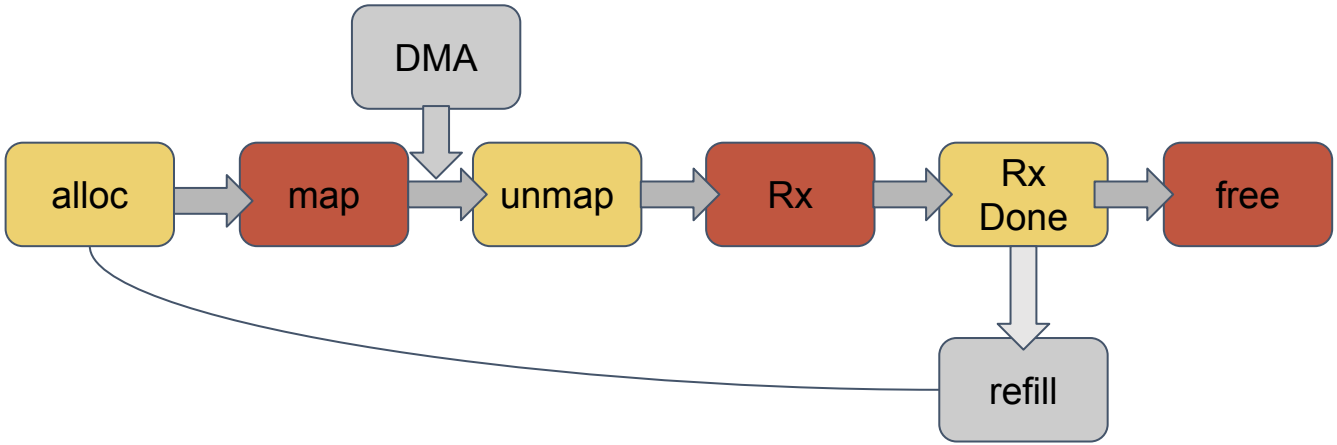
1. Receive Frame
2. Prefetch next free descriptor and DMA data frame
3. Modify the previously fetched buffer descriptor with updated length, checksum and writes it into memory
4. Depending on how the OS has configured the NIC, the NIC may interrupt the host to indicate that a frame has arrived.

Linaro

Current Rx path

1. Allocate buffers
2. Map the buffers to the NIC and populate/refill descriptors
3. Packet arrives
4. Sync for CPU or unmap
5. Allocate new buffer and map it
6. Unmap if (4) just synced
7. Pass the packet to the network stack
8. Refill the descriptors with new allocated buffers
9. kfree_skb () will free the memory when the packet is consumed

Current Rx path

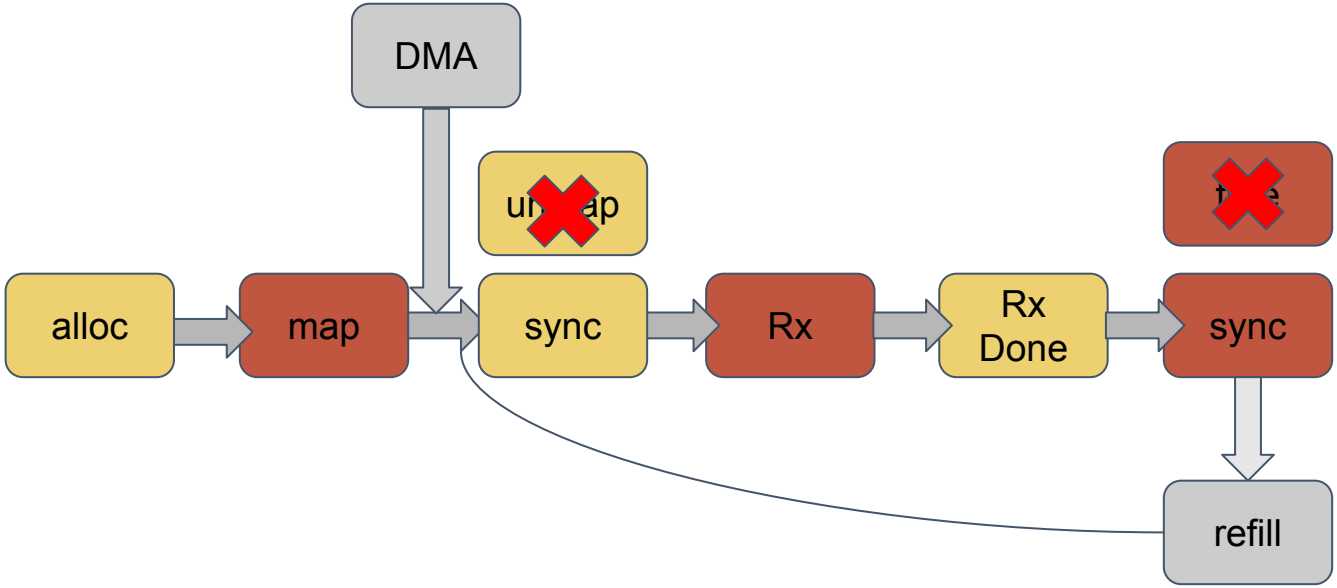


New Rx path

1. Allocate buffers*
2. Map the buffers to the NIC and populate/refill descriptors
3. Packet arrives
4. Sync for CPU
5. Pass the packet to the network stack
6. Refill descriptors*
7. Kfree_skb will recycle the buffer and sync the memory for the NIC

* Since we need the packet processing to finish and then recycle the buffers we have to allocate some extra buffers to compensate

New Rx path



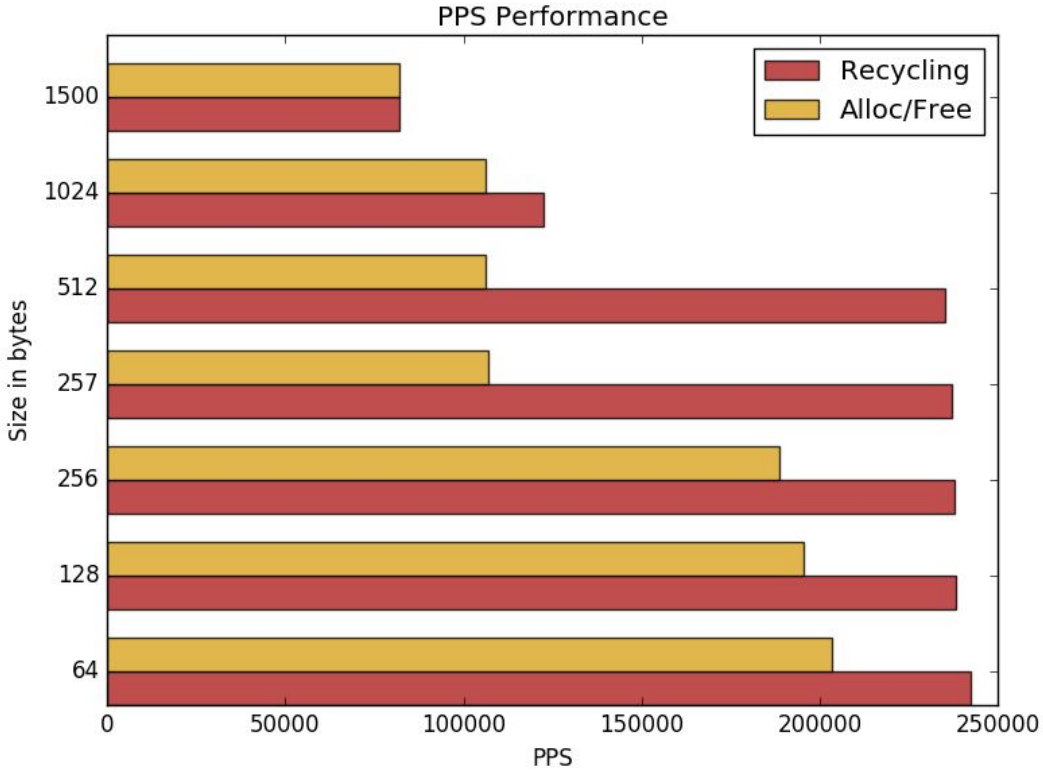
Concerns

- Since we need the packet processing to finish and then recycle the buffers we have to allocate some extra buffers to compensate. We can’t let the hardware starve
- Recycling is based on page_pool API, which allocates a page for a single packet
- Both of the above mean we need more memory
- The memory overhead might be significant on 64k pages and embedded devices but not that much on 4k pages

Tests

- Tested on a single core cortex-a53 using the mvneta driver
- More than 2x performance for specific workloads (256b < packet < 512b)
- No SMMU present on the test hardware. We expect the gains to be even bigger there

Recycling vs current performance



Linaro

Reasons for performance boost

- Driver is recycling buffers instead of allocating and freeing
- Since the old driver recycles buffers for < 256b packets we would expect similar performance
  - We would actually expect the driver to out-perform the new approach since less calls are needed for the descriptor recycling
- The new driver is using build_skb() instead of netdev_alloc_skb_align() which is optimizing the driver further

Next steps

- Fix comments on RFC and sent a v2
- Right now recycling information lives in SKB
- This makes recycling fragments almost impossible and creates lots of corner cases we got to handle.
- By moving recycling information to ‘struct page’ and recycle from there every corner case goes away

All sources can be found on https://github.com/xdp-project/xdp-project/

Linaro

Thank you

Join Linaro to accelerate deployment of your Arm-based solutions through collaboration

contact@linaro.org