



RX and TX Bulking/Batching

Amir, Tariq and Saeed.

Netdev2.1 2017

Agenda



- RX Bulking in driver level
- RX Byte Streaming (Striding RQ) - HW level RX bulking
- TX Doorbell batching (Auto doorbell) – Driver level
- Multi-packet TX descriptor



RX Bulking in driver level

RX Bulking/Batching in driver level - Motivation



- **Data & RX descriptor early prefetch**
 - Prefetch next cacheline while handling current one
 - Data processing pipelining
- **Data cache utilization**
 - Recording the code into stages allows some nice tricks and code tuning
 - Early extraction of needed data from RX descriptors/completions and into temporary array pass them along the pipeline to next stages
- **Instruction cache utilization**
 - Spend more time in the same icache and process more data
- **XDP Intermixed decisions**
 - Bulk handle XDP RX/TX
 - Bulk handle Stack delivery



RX Bulking/Batching in driver level



- How RX is done now
- NAPI Loop:
 - Fetch next completion element
 - Fetch next RX descriptor
 - Fetch data head and tail pointers
 - dma sync
 - Run XDP: if (XDP_TX || XDP_DROP) continue;
 - Alloc/Buidl_skb
 - Populate SKB fields
 - Napi_gro_receive
- napi_complete:
 - Re-populate free RX descriptors
- Problems:
 - Descriptor and data pointers cache miss on every packet
 - Instruction cache invalidation assuming large XDP program with intermixed XDP decisions



RX Bulking/Batching in driver level – Implementation (WIP)



- **NAPI Loop (Broken into 3-4 loops/stages):**
 - **Prefech Loop N RX elements**
 - Fetch next completion element
 - Fetch next RX descriptor
 - Fetch N data head and tail pointers
 - Can be optimized with L2 prefetching
 - dma sync
 - **Run XDP program on N packets: if (XDP_TX || XDP_DROP) continue;**
 - **Stack delivery (N SKBs):**
 - Alloc/Build SKB
 - Populate SKB fields
 - Napi_gro_receive
- **napi_complete:**
 - Re-populate free RX descriptors



RX Bulking/Batching in driver level – Risks & TODOs



- In most modern systems with DDIO, data prefetching has little gain.
- Breaking code into stages adds more instruction caches and can reduce the common case performance
- How much to prefetch ?
 - “Too much” prefetching can have a bad effect on the staging mechanism and invalidate the cache too soon
 - Cache optimizations are Arch depended.
- Common stack delivery case (No XDP program)
 - The icache optimization can be seen only with large XDP programs with intermixed XDP descision.
 - Driver code relative to that stack code is so small, icache optimization is almost un-noticed.
 - New CPU instructions added to the common data path (7-10 more CPU cycle per packet)
 - **Separate RX staging/bulking data path for when XDP is loaded ?**
- Wide system/Arch testing and code tuning is required and lots of measurements
 - X86, ARM, PPC, etc..
 - All types of work loads with different types of XDP programs



RX Bulking/Batching in driver level – “Some Results”



- Early patches with only descriptor bulking and Data prefetching show some promising results even for the common case
 - Process RX descriptor into temporary array seem to have some positive effect (reduces cache misses)
 - Reduced cache miss on first packet.
 - 10% improvement in XDP_TX (Data touch)
 - 1-2% improvement in XDP_DROP (No data touch)
 - 1-2% improvement in Stack delivery.
- For Stack delivery and XDP bulking we still don't have clear results.



HW level RX Bulking

Byte streaming
Striding receive queue

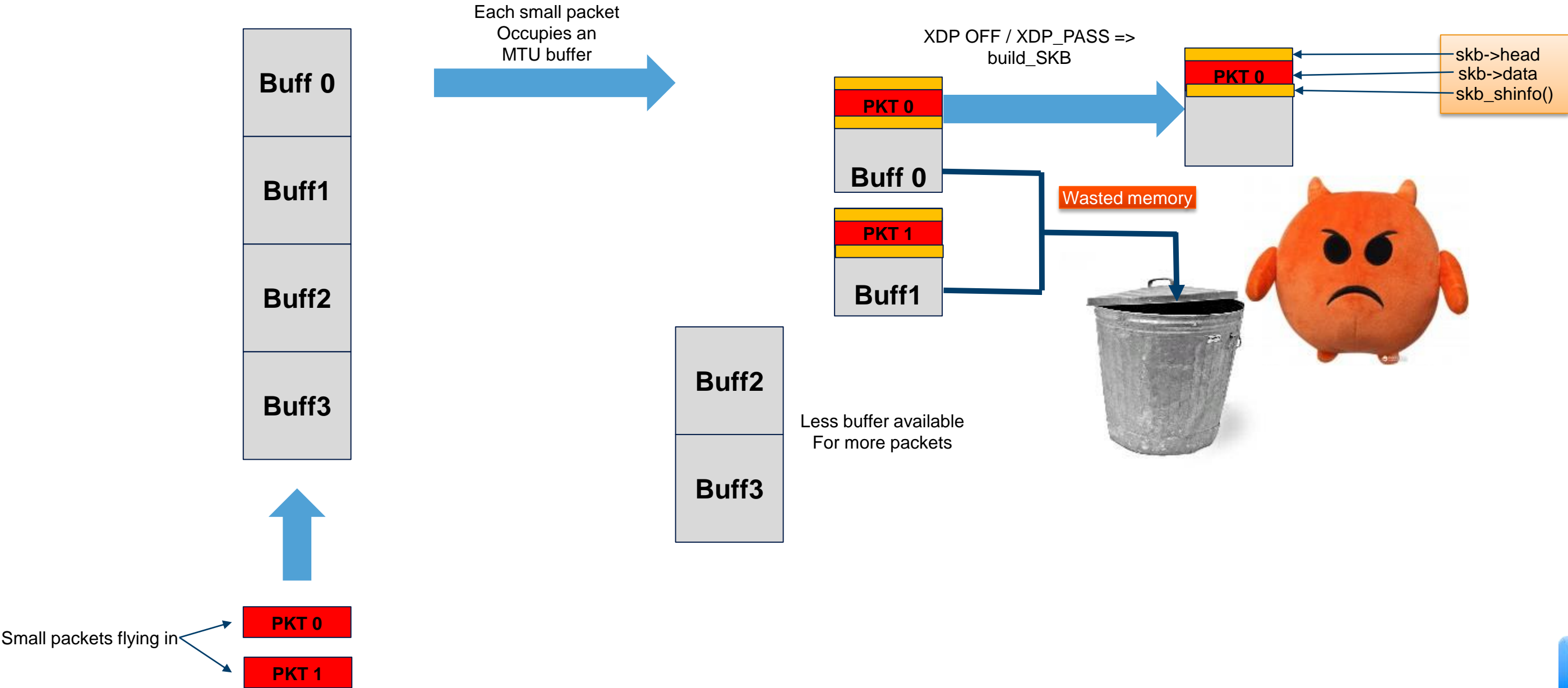
Motivation - Conventional RX Ring disadvantages



- **Per packet HW descriptor (WQE, Work Queue Element).**
 - Per packet, two HW PCI transactions:
 - read RX descriptor,
 - then write packet to data pointers.
- **Buffer size: largest possible packet, MTU.**
 - Memory wasteful, when actual occupying packets are small.

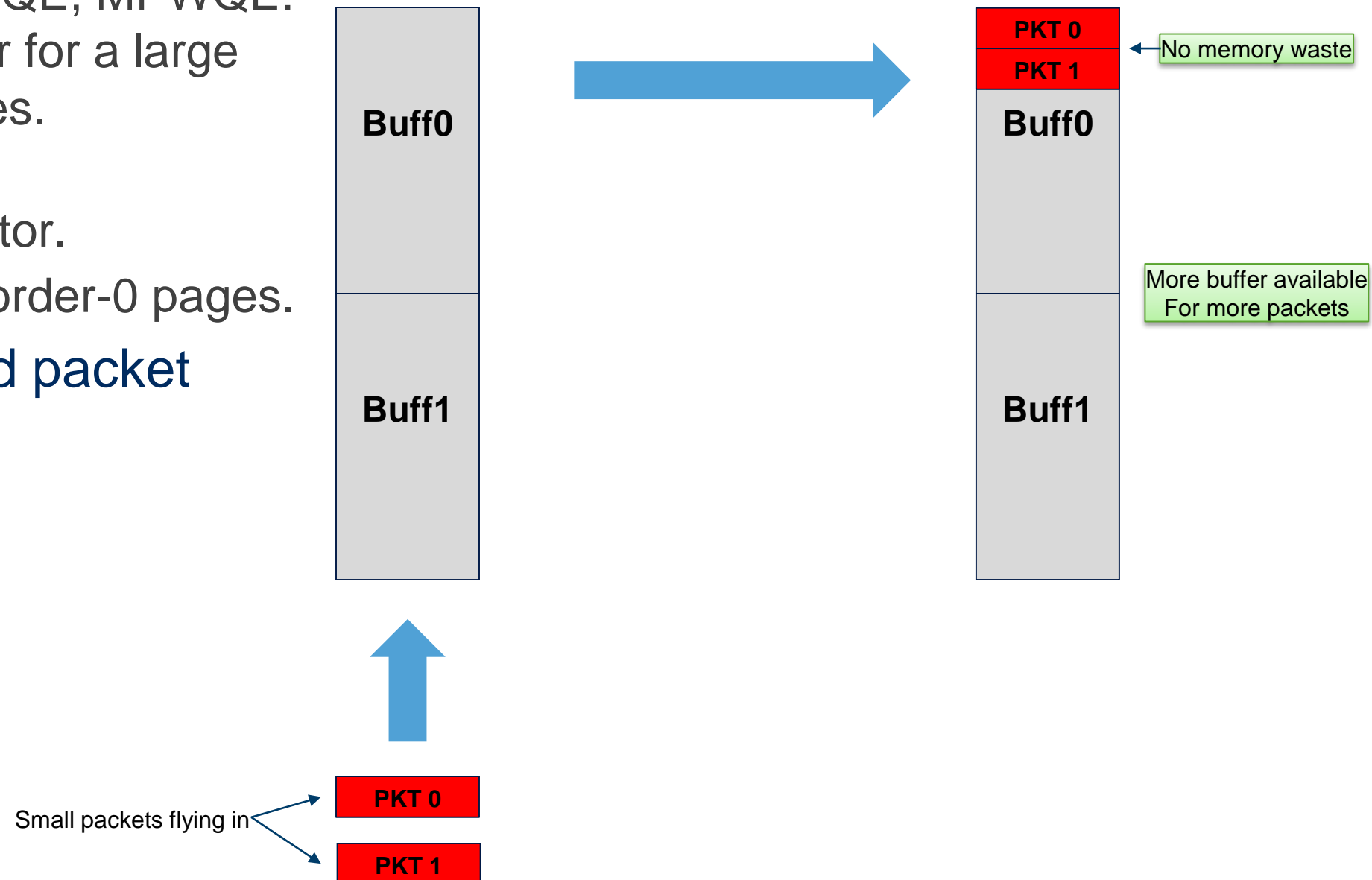


Conventional RX memory model: MTU buffer per packet + build_skb



Striding RQ: Byte Stream buffering

- **Batching of HW descriptors:**
 - RX Multi-packet WQE, MPWQE:
One HW descriptor for a large number of RX Bytes.
 - Typical buffer size:
256KB per descriptor.
 - Buffer consists of order-0 pages.
- **HW writes received packet continuously.**



Solution: Striding RQ (Byte Stream buffering)



■ Advantages:

- HW PCI transaction to read RX descriptor is done only once per RX MPWQE.
- No memory waste.
- NIC DMA writes memory locality.
- MTU agnostic: Not directly affected by MTU size.

■ HW packet rate limit for queue: 50 Mpps (vs. 30 Mpps)

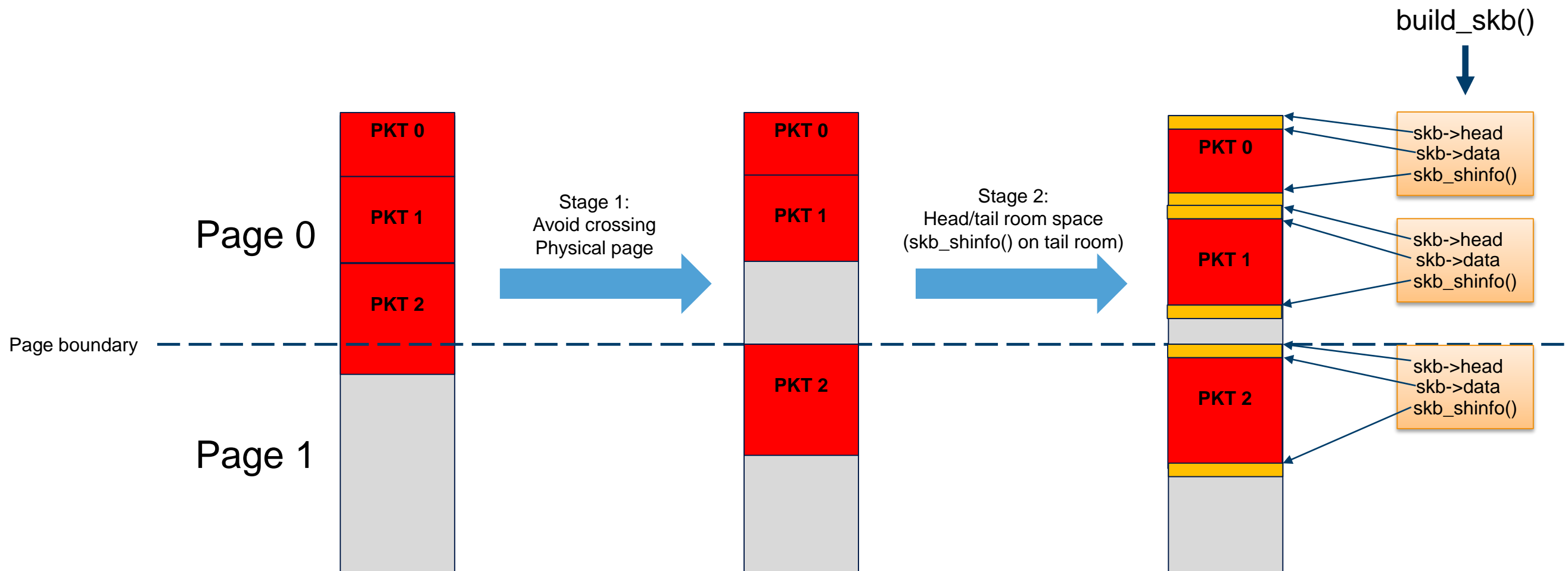
■ Issues: Currently does not fit with XDP requirements:

- page-per-packet.
- linear SKB.

■ Future work: XDP with Striding RQ.



Future HW enhancements: SKB Fragments avoidance



Notes:

- Linear packets should yield better performance
- XDP (eXpress Data Path) requires linear packets

- Netperf single TCP stream:
 - BW raised by 10-15% for representative packet sizes:
 - default, 64B, 1024B, 1478B, 65536B.
- Netperf multi TCP stream:
 - No degradation, line rate reached.
- Pktgen: packet rate raised by 2-10% for traffic of different message sizes:
 - 64B, 128B, 256B, 1024B, and 1500B.
- Pktgen: packet loss in short bursts of small messages (64byte), single stream:

# packets	Conventional RQ	Striding RQ
2K	~1K	0
8K	~6K	0
16K	~13K	0
32K	~28K	0
64K	~57K	~24K

Driver level TX bulking

■ Motivation:

- In TX packet production, HW is notified when new packets are ready for transmission.
- These notify HW (doorbells) operations are costly.
- Save doorbells by doing it once per a batch of TX packets.

■ Existing solutions:

- Xmit_more: a boolean field in SKB indicating that another packet is to follow.
- Driver does not issue a doorbell for the packet.

■ Idea: Transparent Xmit_more –like mechanism, in driver level.

- Save doorbells if a TX NAPI completion is expected soon.
- Issue a single doorbell in NAPI context for the accumulated packets.

■ Initial performance numbers:

- Packet-rate of single stream, tested with pktgen.
- mlx5, Connect-X 5, Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz.

	TX PKT RATE (PPS)	SPEEDUP
BASE	1,758,312	
N=2	1,905,652	108%
N=4	2,116,269	120%
N=8	2,243,828	128%
N=16	2,379,595	135%
N=32	2,526,059	144%
N=64	2,567,724	146%
N=128	2,591,796	147%

■ Challenges:

- Synchronization: Who shall ring the doorbell? `ndo_start_xmit` vs NAPI
 - HW level TX polling?
- Waiting variable delays – adaptive interrupt moderation for TX? TX Busy polling?

HW level TX bulking

Multi Packet TX descriptor (TX MPWQE)

Today – WQE (TX desc) per packet



- How can you use 3*64B cache lines in a TX queue ?

Regular WQE format			
0-3	4-7	8-11	12-15
Control Segment[0]	Control Segment[1]	Control Segment[2]	Control Segment[3]
Ethernet Segment[0]	Ethernet Segment[1]	Ethernet Segment[2]	Ethernet Segment[3]
<Memory pointer header segment>	<pointer addr>	<pointer addr>	<pointer addr>
<padding> 0000	<padding> 0000	<padding> 0000	<padding> 0000
Control Segment[0]	Control Segment[1]	Control Segment[2]	Control Segment[3]
Ethernet Segment[0]	Ethernet Segment[1]	Ethernet Segment[2]	Ethernet Segment[3]
<Memory pointer header segment>	<pointer addr>	<pointer addr>	<pointer addr>
<padding> 0000	<padding> 0000	<padding> 0000	<padding> 0000
Control Segment[0]	Control Segment[1]	Control Segment[2]	Control Segment[3]
Ethernet Segment[0]	Ethernet Segment[1]	Ethernet Segment[2]	Ethernet Segment[3]
<Memory pointer header segment>	<pointer addr>	<pointer addr>	<pointer addr>
<padding> 0000	<padding> 0000	<padding> 0000	<padding> 0000

- Answer: 3 Packets – 64B per packet



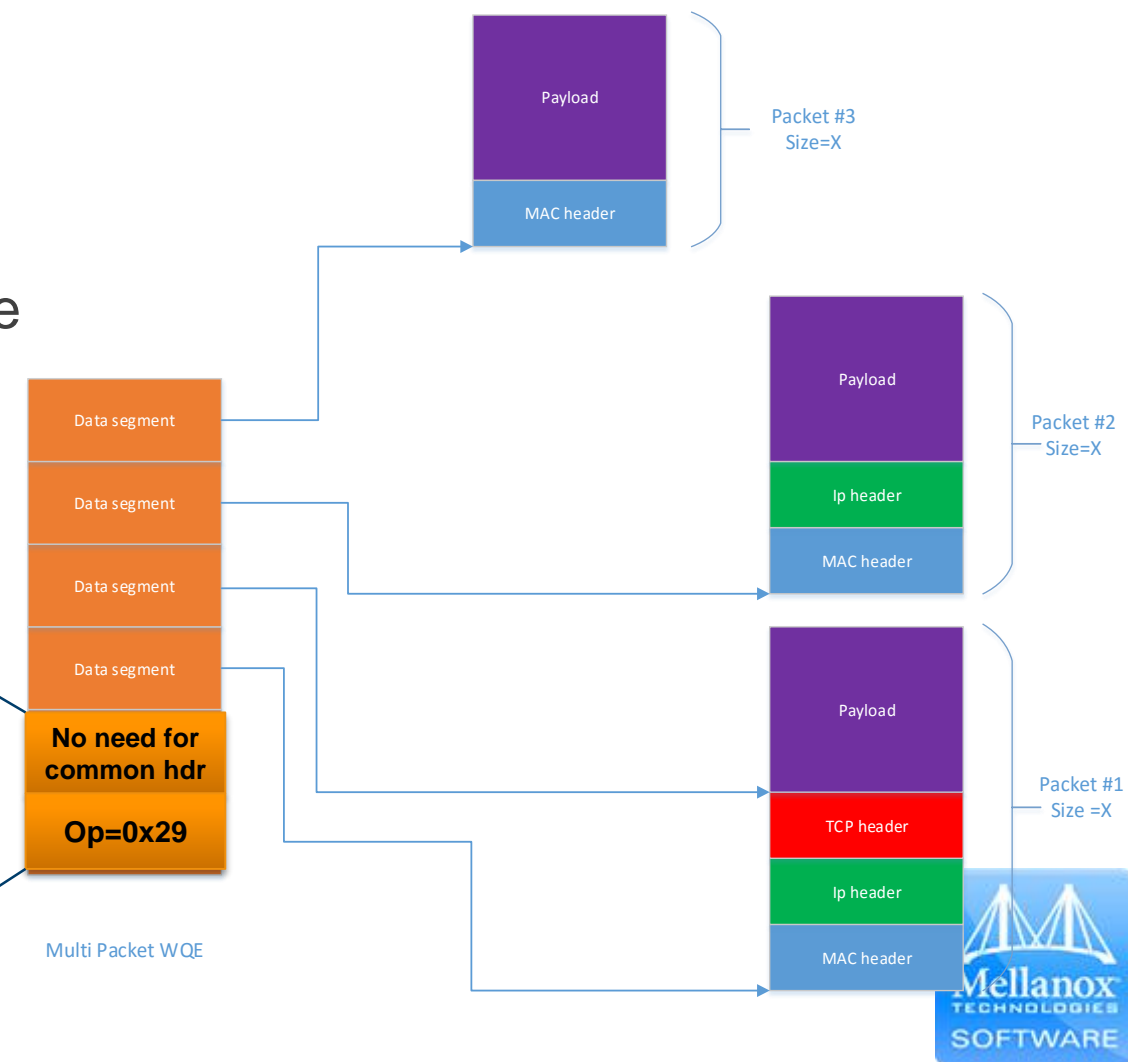
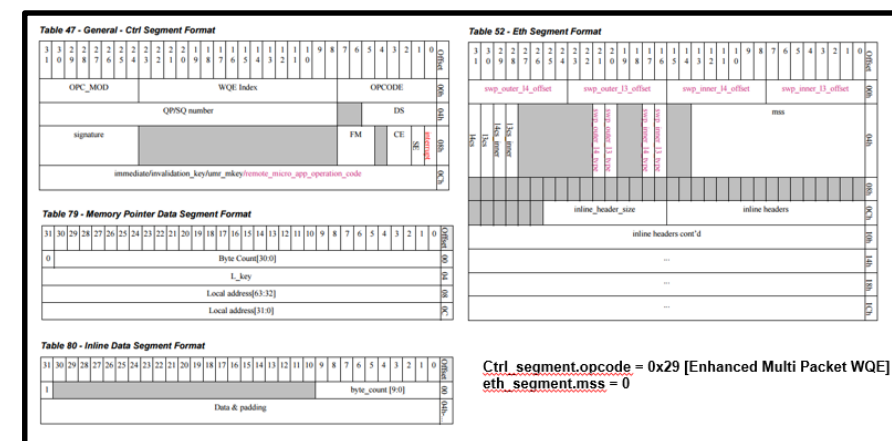
Enhanced Multi Packet WQE (TX) – ConnectX-5 and beyond

■ How does it work?

- Each TX descriptor is constructed from one control segment and multiple data segments
- The HW treat each data segment as a separate packet
- Packets can differ in headers and size
- Up to 60 packets in 1K WQE

■ Benefits

- Reduce CPU% by better use of cache lines → Higher TX packet rate
- Reduce PCI control packets overhead over data packets
- TX ring size can be reduced holding same amount of packets



Enhanced Multi Packet WQE format - segments



Table 47 - General - Ctrl Segment Format

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0	Offset
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0							00h		
OPC_MOD								WQE Index												OPCODE						00h				
QP/SQ number																						DS				04h				
signature																				FM				CE		SE	interrupt	08h		
immediate/invalidation_key/umr_mkey/remote_micro_app_operation_code																												0Ch		

Table 79 - Memory Pointer Data Segment Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Offset
0	Byte Count[30:0]																															00
	L_key																															04
	Local address[63:32]																															08
	Local address[31:0]																															0C

Table 80 - Inline Data Segment Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Offset
1																						byte_count [9:0]							00			
Data & padding																																04h-...

Table 52 - Eth Segment Format

3	3	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0	Offset	
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0										00h	
swp_outer_14_offset								swp_outer_13_offset								swp_inner_14_offset								swp_inner_13_offset								00h
14cs		13cs		14cs inner		13cs inner										swp_outer_14 type		swp_outer_13 type		swp_inner_14 type		swp_inner_13 type		mss								04h
																																08h
								inline_header_size								inline headers																0Ch
inline headers cont'd																																10h
...																																14h
...																																18h
...																																1Ch

Ctrl_segment.opcode = 0x29 [Enhanced Multi Packet WQE]
eth_segment.mss = 0



Enhanced Multi Packet WQE format



- How can you use 3*64B cache lines in a TX queue ?

EMPW - WQE format – All pointers			
0-3	4-7	8-11	12-15
Control Segment[0]	Control Segment[1]	Control Segment[2]	Control Segment[3]
Ethernet Segment[0]	Ethernet Segment[1]	Ethernet Segment[2]	Ethernet Segment[3]
<Memory pointer header segment>	<pointer addr>	<pointer addr>	<pointer addr>
<Memory pointer header segment>	<pointer addr>	<pointer addr>	<pointer addr>
<Memory pointer header segment>	<pointer addr>	<pointer addr>	<pointer addr>
<Memory pointer header segment>	<pointer addr>	<pointer addr>	<pointer addr>
<Memory pointer header segment>	<pointer addr>	<pointer addr>	<pointer addr>
<Memory pointer header segment>	<pointer addr>	<pointer addr>	<pointer addr>
<Memory pointer header segment>	<pointer addr>	<pointer addr>	<pointer addr>
<Memory pointer header segment>	<pointer addr>	<pointer addr>	<pointer addr>
<Memory pointer header segment>	<pointer addr>	<pointer addr>	<pointer addr>
<Memory pointer header segment>	<pointer addr>	<pointer addr>	<pointer addr>

- 10 Packets – 19B per packet, 60 packets in 1K – 17B per packet



TX Ring Size optimization



- TX Ring Default - 1024
 - Outstanding packets: 1024
 - Bytes for the TX queue: $64 \times 1024 = 64\text{KB}$
- Smaller TX Ring with EMPW - 256
 - Outstanding packets: 256-1024, depend on burst size
 - Bytes for the TX queue: $64 \times 256 = 16\text{KB}$

ethtool -G p4p1 tx 256

ethtool -g p4p1 tx 256

Ring parameters for p4p1:

Current hardware settings:

RX: 1024

RX Mini: 0

RX Jumbo: 0

TX: 256





Thank You