DPDK
DATA PLANE DEVELOPMENT KIT

Accelerating Packet Processing with GRO and GSO in DPDK

DPDK Summit - San Jose – 2017

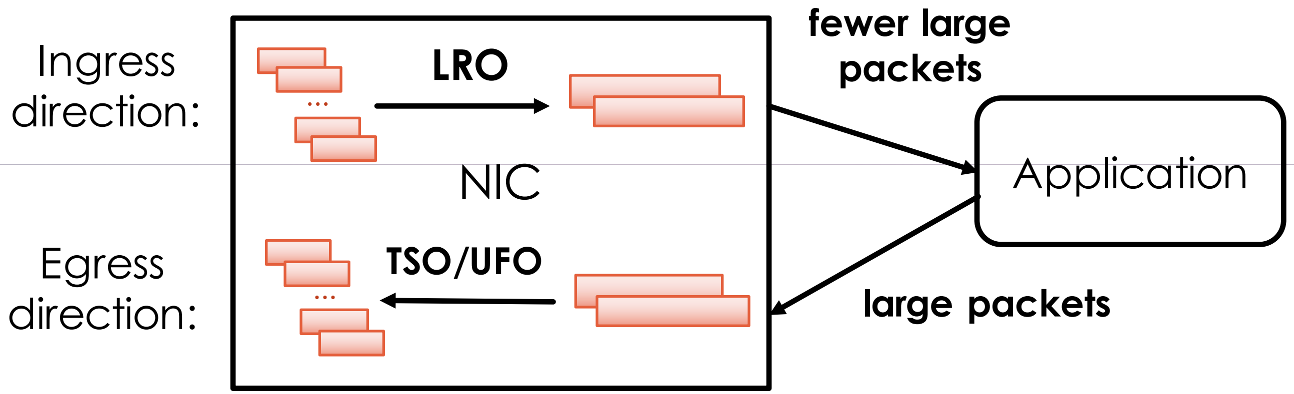
#DPDKSummit

Packet Processing Overheads

- A major part of packet processing “applications” are performed on a per-packet basis (e.g. Firewall, TCP/IPv4 stack)
- Per-packet routines (e.g. header processing) dominate packet processing overheads
- ***Reduce the packet number*** to be processed can mitigate the per-packet overhead

Methods to Reduce the Packet Number

- Use large Maximum Transmission Unit (MTU)
  - **MTU size depends on physical links** ☹
- Use network interface card (NIC) features: Large Receive Offload (LRO), TCP Segmentation Offload (TSO) and UDP Fragmentation Offload (UFO)



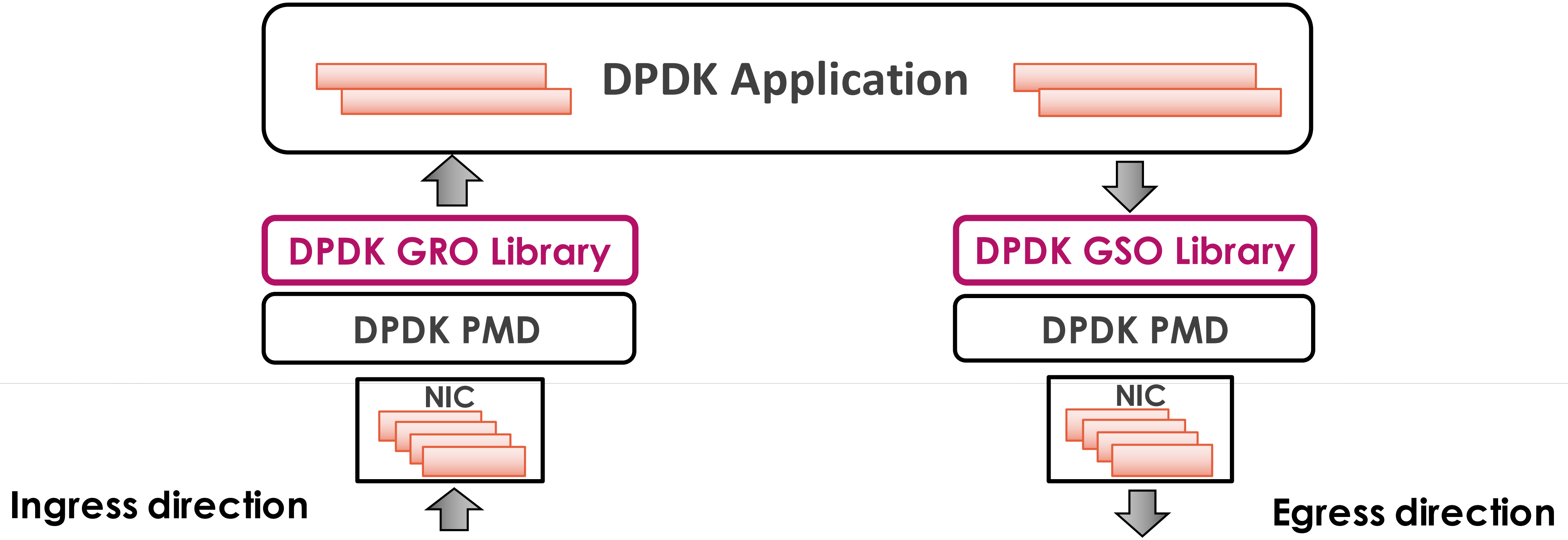
Drawbacks:
- **Only support TCP and UDP**
- **Not all NICs support LRO/TSO/UFO**

GRO and GSO

- Software Method:
  - Generic Receive Offload (GRO) merges small packets into larger ones
  - Generic Segmentation Offload (GSO) splits large packets into smaller ones
- Advantages:
  - **Don’t reply on physical link support**
  - **Don’t reply on NIC hardware support**
  - **Able to support various kinds of protocols, like TCP, VxLAN and GRE**

GRO and GSO in DPDK

- In DPDK, GRO and GSO are two standalone libraries



Library Framework

DPDK

rte_gro_ctx_create()
rte_gro_ctx_destroy()
rte_gro_reassemble…()
rte_gro_timeout_flush()
rte_gro_get_pkt_count()

GRO API

Packet type: IPv4-GRE-IPv4-TCP

Packet type: IPv4-TCP

IPv4-GRE-TCP/IPv4 GRO

…

TCP/IPv4 GRO

- One GRO type, one kind of packets
- Indicated by MBUF->packet_type

---

ol_flags:
PKT_TX_TCP_SEG |
PKT_TX_IPV4 |
PKT_TX_OUTER_IPV4 |
PKT_TX_TUNNEL_GRE

GSO API

rte_gso_segment(pkt)

ol_flags:
PKT_TX_TCP_SEG | PKT_TX_IPV4

IPv4-GRE-TCP/IPv4 GSO

…

TCP/IPv4 GSO

- One GSO type, one kind of packets
- Indicated by MBUF->ol_flags

API: How Applications Use it?

GRO Sample

```
nb_pkts = rte_eth_rx_burst(…, pkts, …);
nb_segs = rte_gro_reassemble_burst(pkts, nb_pkts, …);
rte_eth_tx_burst(…, pkts, nb_segs);
```

GSO Sample

```
struct rte_mbuf *gso_segs[N];
nb_segs = rte_gso_segment(…, pkt, gso_segs, …);
rte_eth_tx_burst(…, gso_segs, nb_segs);
```

API: Two Sets in GRO

Lightweight Mode API

rte_gro_reassemble_burst()

Heavyweight Mode API

ctx = rte_gro_ctx_create()

↓

rte_gro_reassemble(pkts, ctx)

↓

rte_gro_timeout_flush(ctx)

- Support to merge a **small number** of packets **rapidly**

- Supports to merge a **large number** of packets with **fine-grained control**

How to Merge and Split Packets?

- GRO Reassembly Algorithm merges packets
- GSO Segmentation Scheme splits packets

GRO Algorithm: Challenges

- A high cost algorithm/implementation would cause packet dropping in a high speed network

Algorithm is lightweight to scale fast networking speed

- Packet reordering makes it hard to merge packets
  - Linux GRO fails to merge packets when encounters packet reordering

Algorithm is capable of handling packet reordering

GRO Algorithm: Key-based Approach

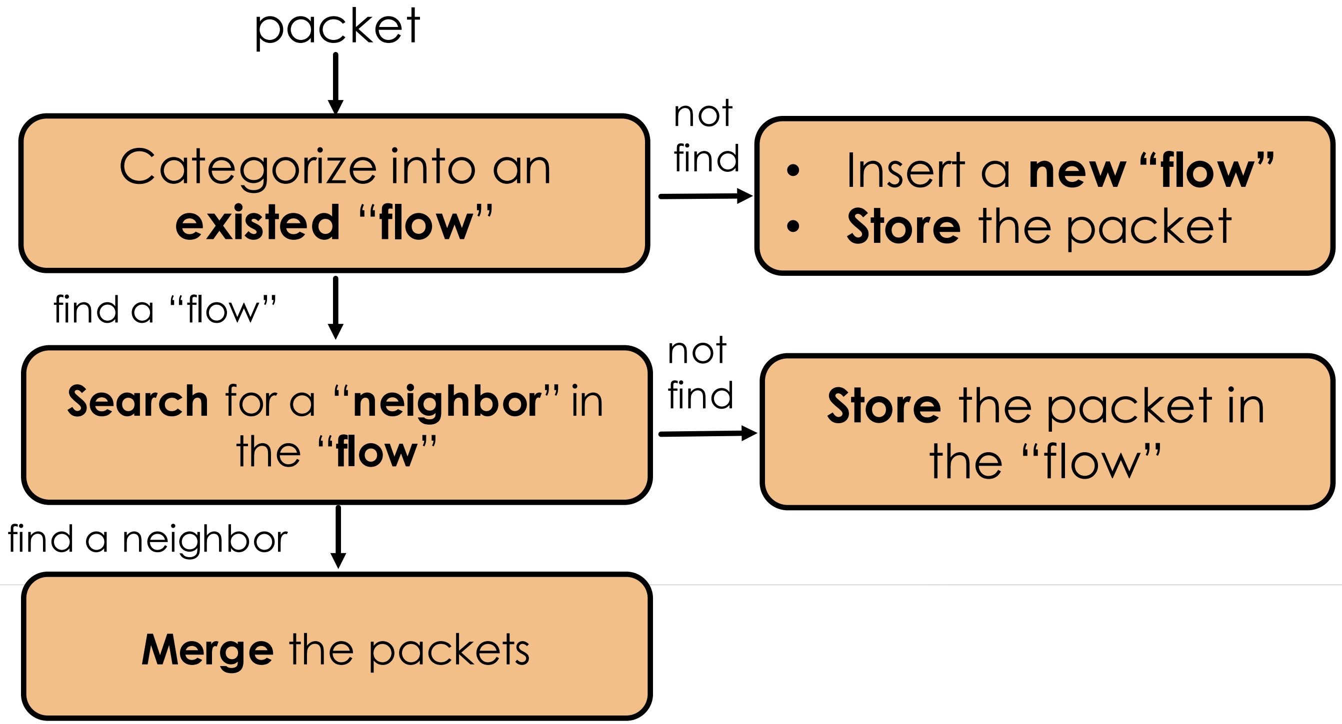
TCP/IPv4 Packets

Header fields representing a "flow":

- src/dst: mac, IP, port **same**
- ACK number **value**

Header fields deciding "neighbor":

- IP id **incremental**
- Sequence number



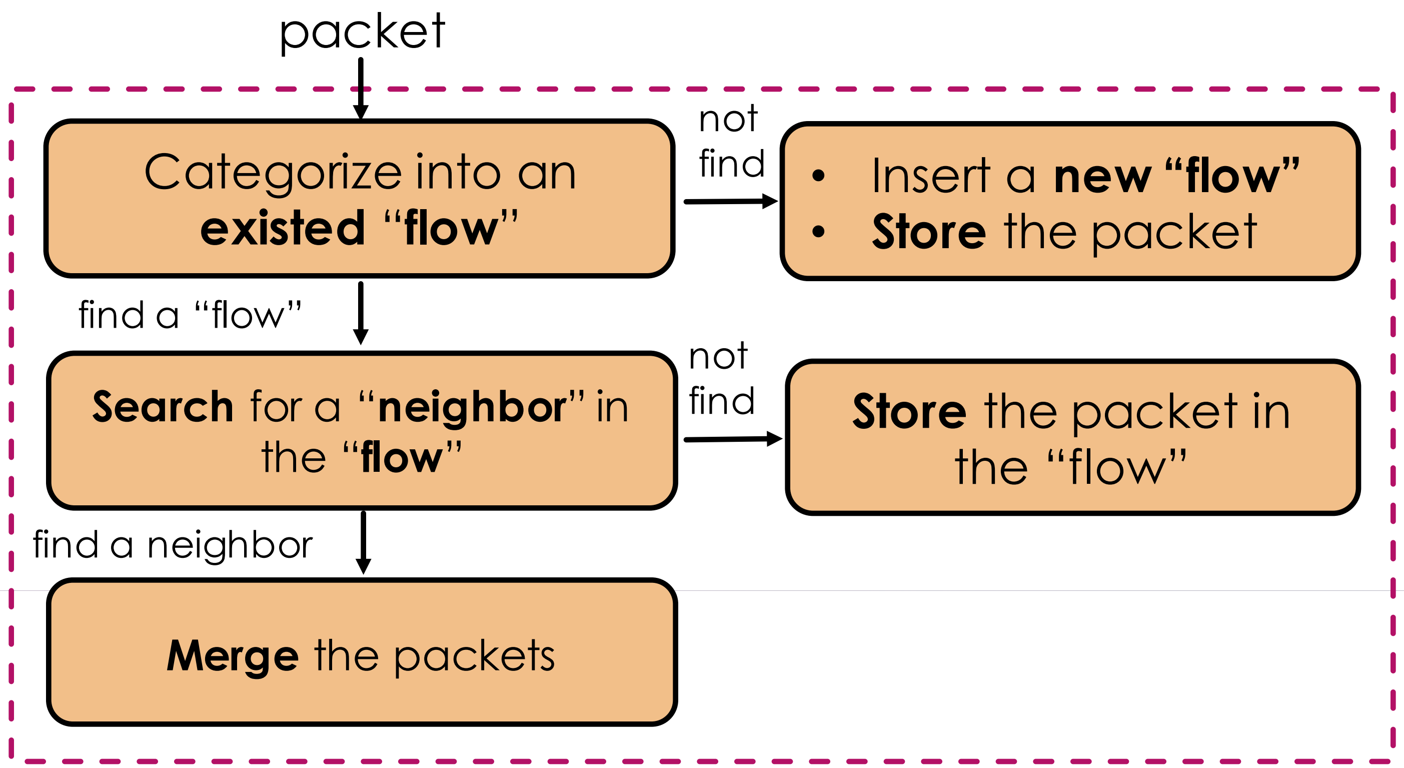
GRO Algorithm: Key-based Approach

DPDK

Two Characters

- **Lightweight:** classify packets into "flows" to accelerate packet aggregation is simple
- **More**: storing out-of-order packets makes it possible to merge later

Address challenge 1 and 2



GRO Algorithm: TCP/IPv4 Example

①

Flow	MAC	IP	Port	ACK Number
F1	1:2:3:4:5:6 11:22:33:44:55:66	1.1.1.1 1.1.1.2	5041 5043	1
F2	1:2:3:4:5:6 11:22:33:44:55:66	1.1.1.1 1.1.1.2	5001 5002	1

Packet	IP ID	Sequence number	Payload length
P1	7	701	1500

②

Packet	IP ID	Sequence number	Payload length
P0	1	1	100
P2	3	301	100

P3

Flow	IP ID	Sequence number	Payload length
F2	4	401	100

GRO Algorithm: TCP/IPv4 Example

DPDK

Flow	MAC	IP	Port	ACK Number
F1	1:2:3:4:5:6 11:22:33:44:55:66	1.1.1.1 1.1.1.2	5041 5043	1
F2	1:2:3:4:5:6 11:22:33:44:55:66	1.1.1.1 1.1.1.2	5001 5002	1

Packets	IP ID	Sequence number	Payload length
P1	7	701	1500

Packets	IP ID	Sequence number	Payload length
P0	1	1	100
P2	**3**	**301**	**100**

P3

Flow	IP ID	Sequence number	Payload length
F2	**4**	**401**	100

IP ID and sequence number are incremental → Neighbors

GRO Algorithm: TCP/IPv4 Example

Flow	MAC	IP	Port	ACK Number
f0	1:2:3:4:5:6 11:22:33:44:55:66	1.1.1.1 1.1.1.2	5041 5043	1
f1	1:2:3:4:5:6 11:22:33:44:55:66	1.1.1.1 1.1.1.2	5001 5002	1

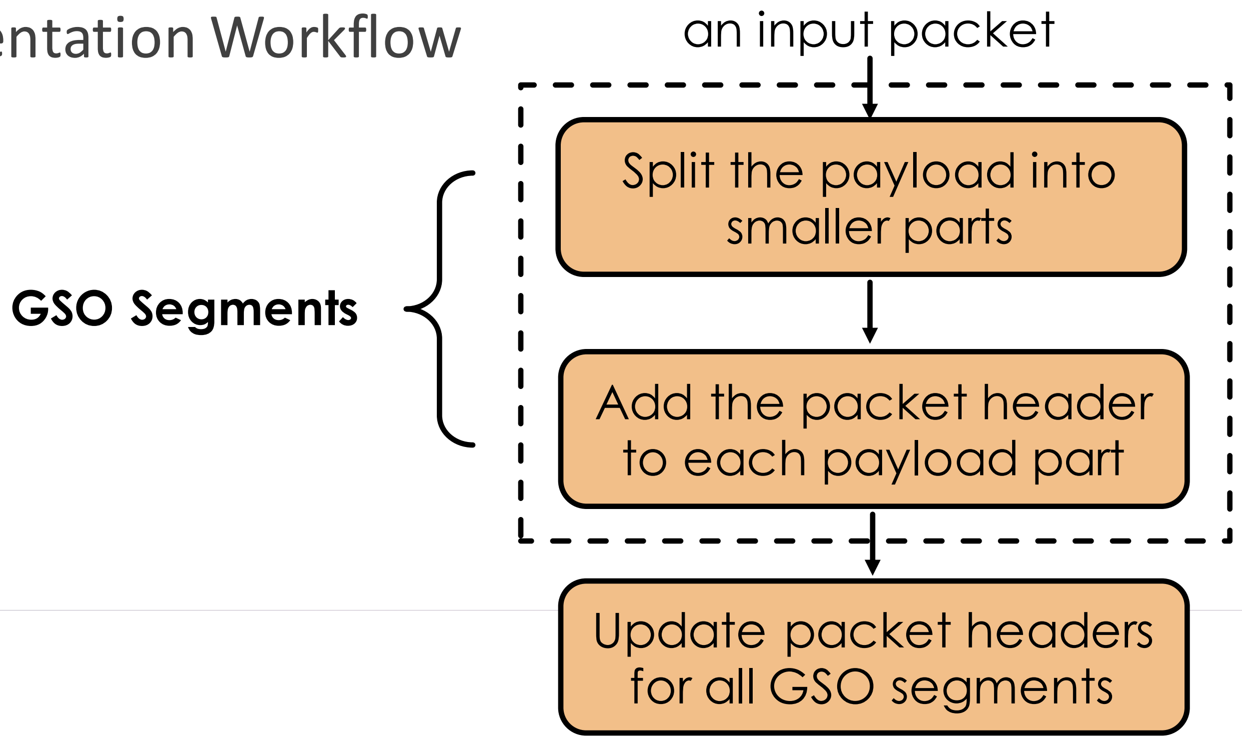
Packets	IP ID	Sequence number	Payload length
P1	7	701	1500

Packets	IP ID	Sequence number	Payload length
P0	1	1	100
P2	**4**	**301**	**200**

③

GSO Scheme: Overview

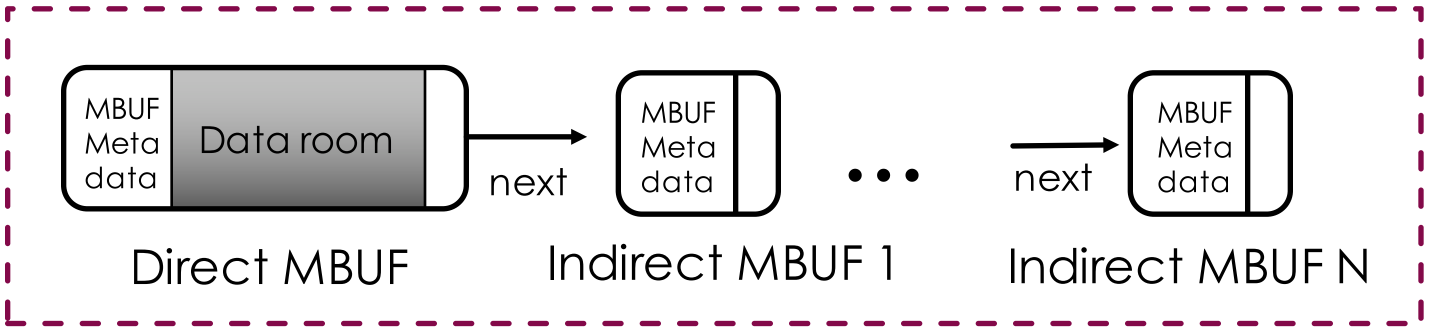
- Segmentation Workflow



How to organize a GSO segment?

GSO Scheme: Zero-copy Based Approach

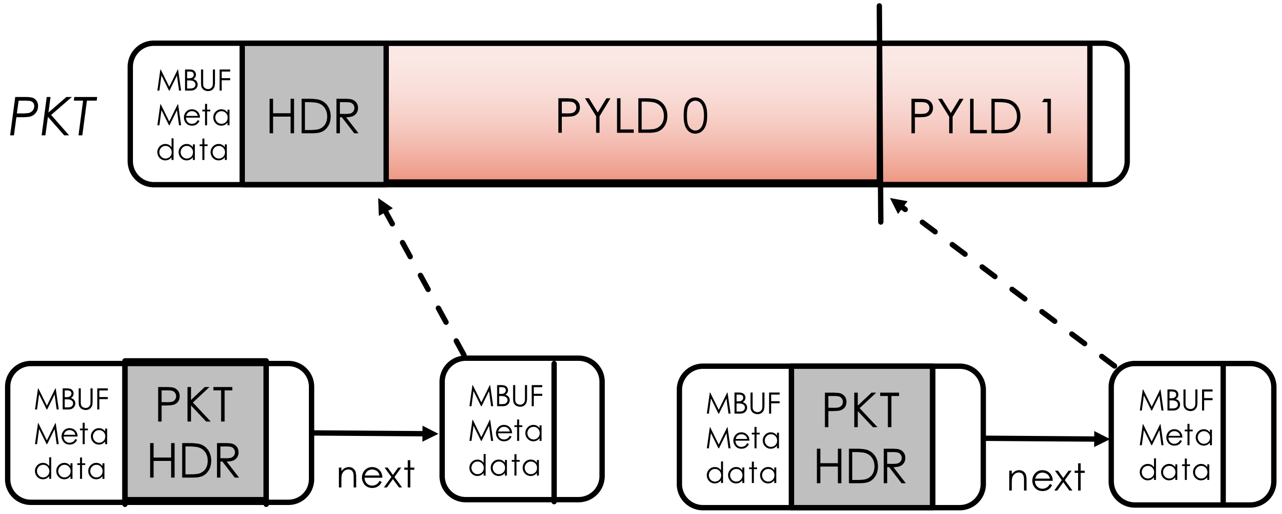
- "Two-part" MBUF is used to organize a GSO segment



  - Direct MBUF **holds** the packet header
  - Indirect MBUF: "**pointer**", point to a payload part of the packet to segment

Forming a GSO segment just needs to copy the header!

GSO Scheme: Example



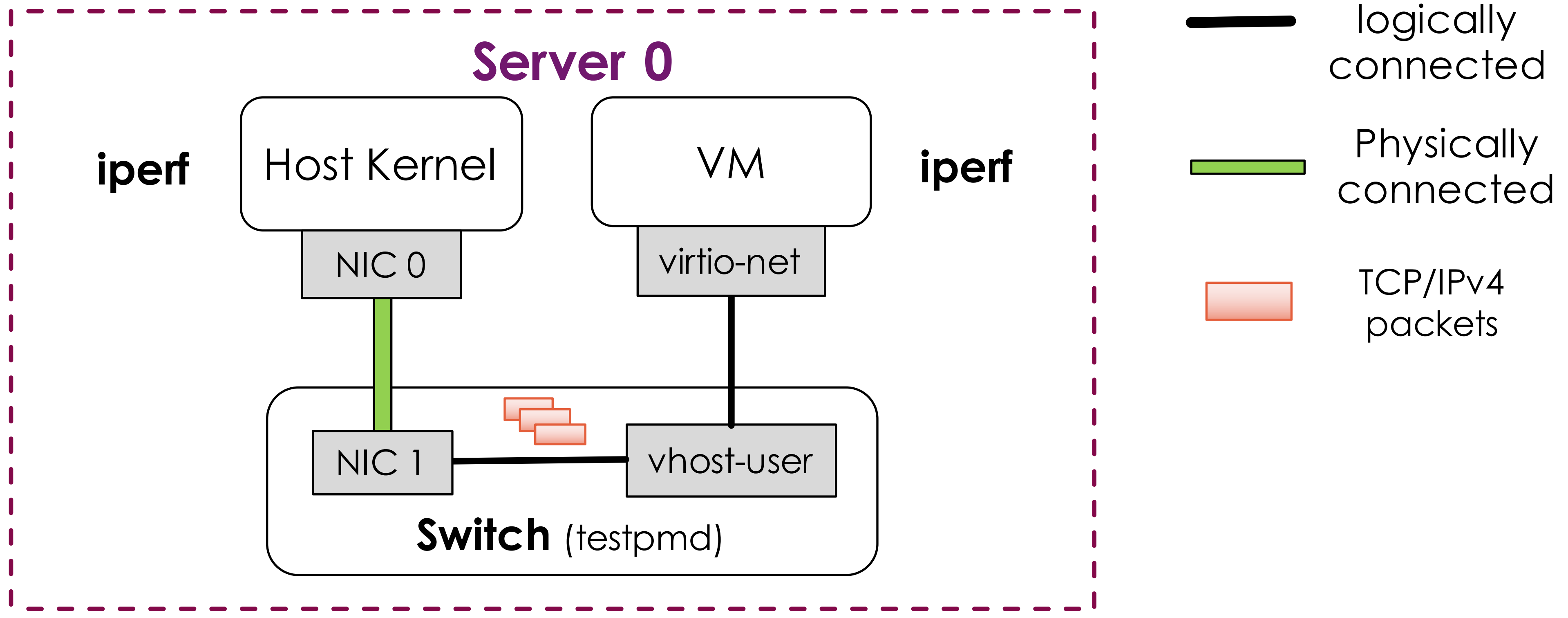
③ Reduce reference counter of *PKT*'s MBUF by 1

- *PKT* will be freed automatically, When all indirect MBUFs are freed

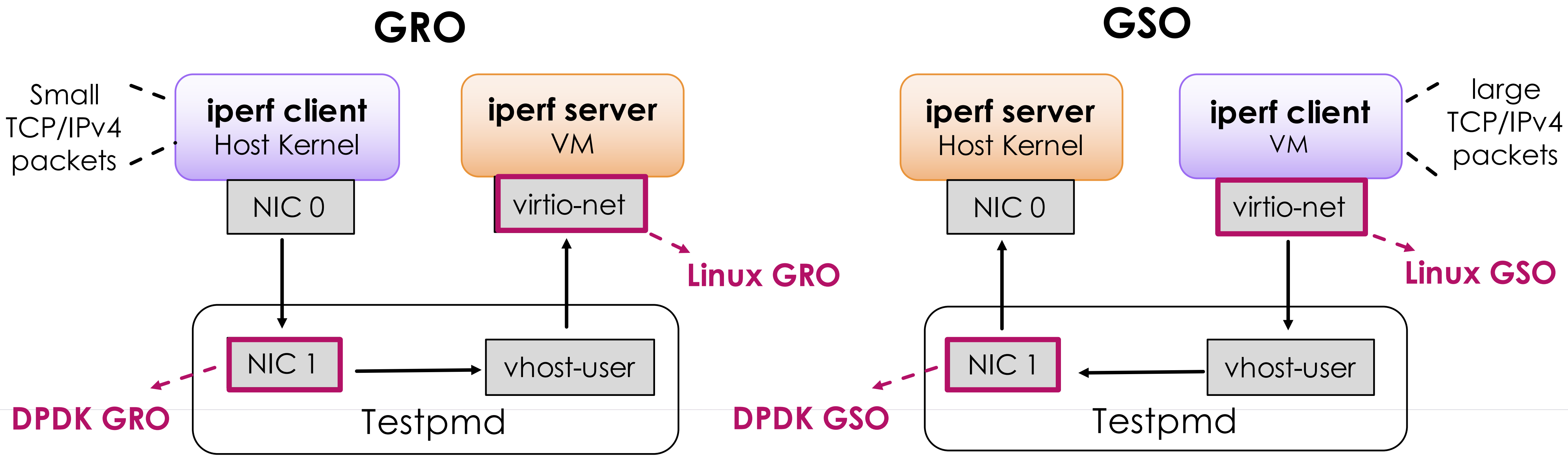
① Copy HDR to direct MBUF

② "Attach" indirect MBUF to PKT and make it point to the correct payload part

Experiment: Physical Topology



Experiment: Setup



Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz
Ethernet Controller XL710 for 40GbE QSFP+

Experiment: GRO Performance

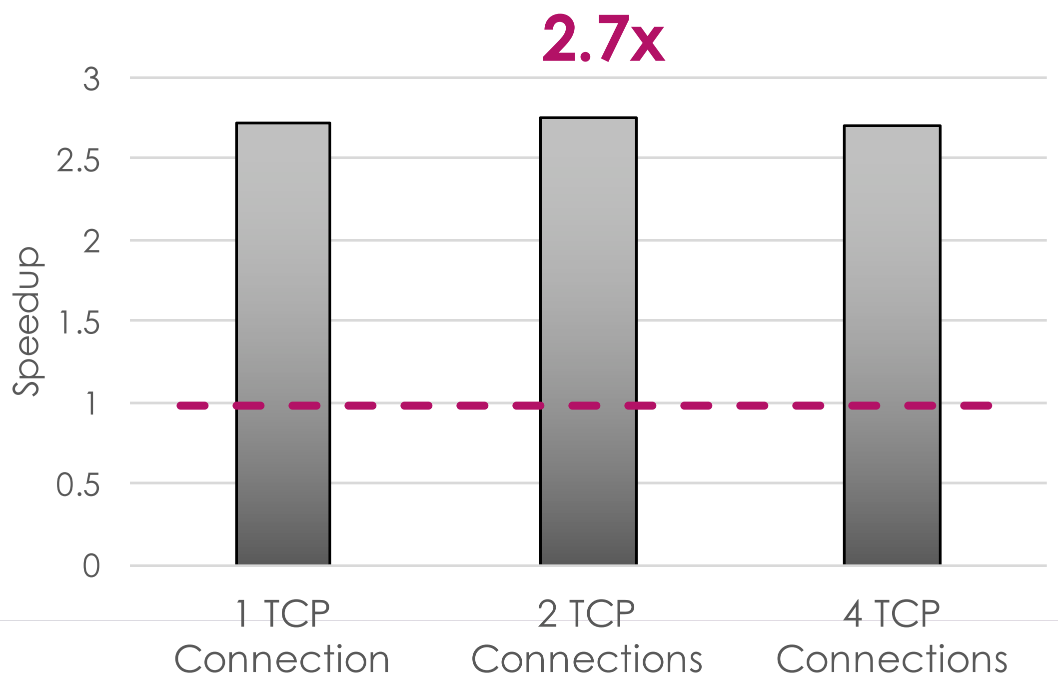


Figure 1. Iperf throughput Improvement of **DPDK-GRO** over **No-GRO**

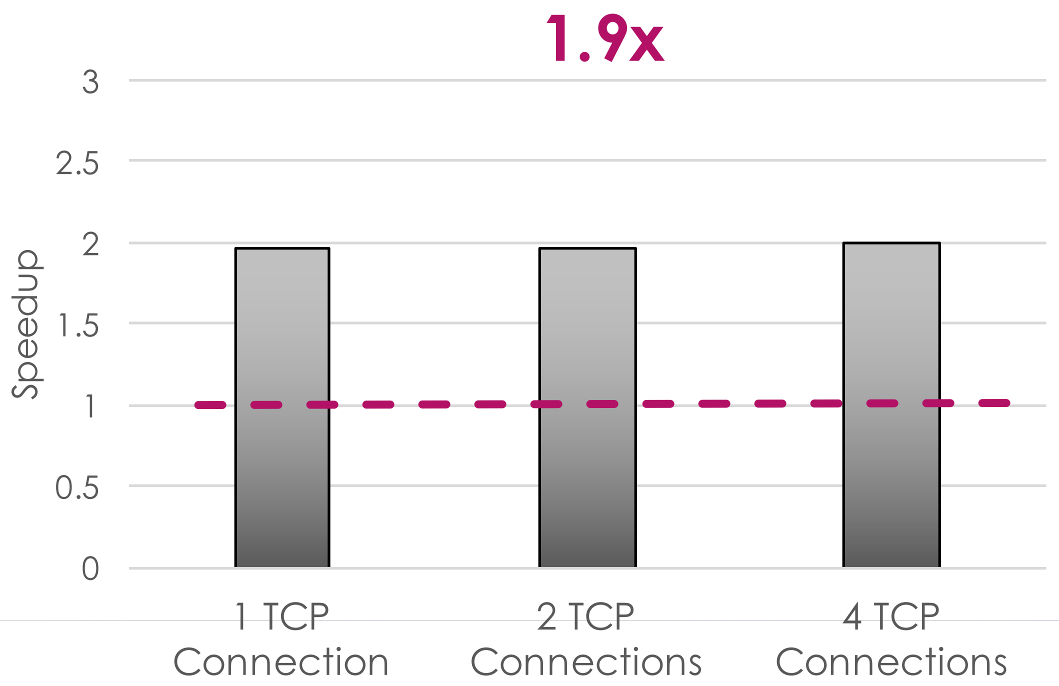


Figure 2. Iperf throughput Improvement of **DPDK-GRO** over **Linux-GRO**

Experiment: GSO Performance

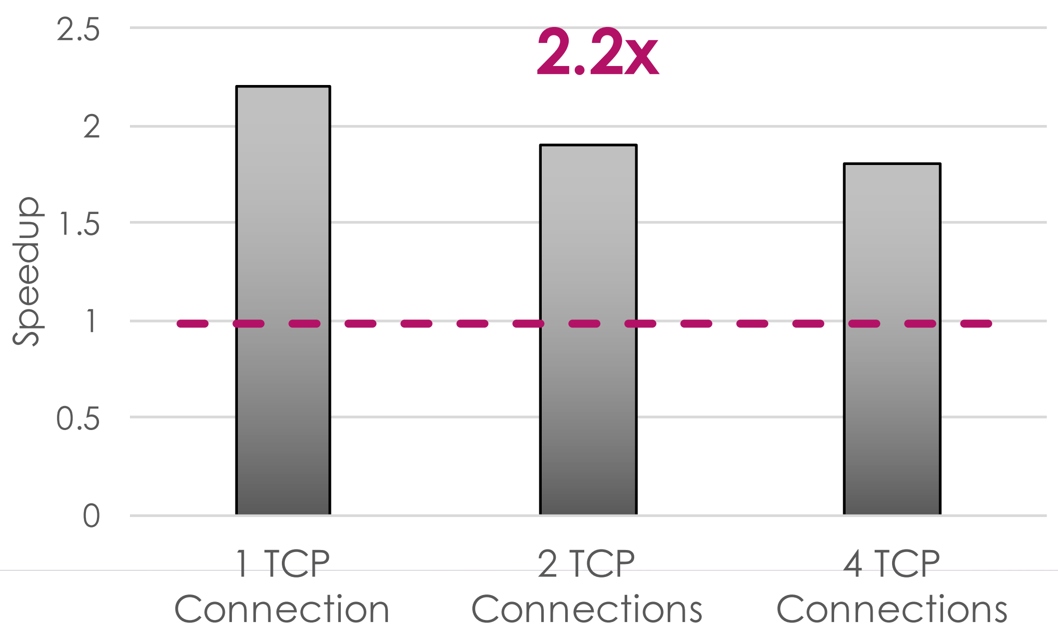


Figure 1. Iperf throughput Improvement of **DPDK-GSO** over **No-GSO**

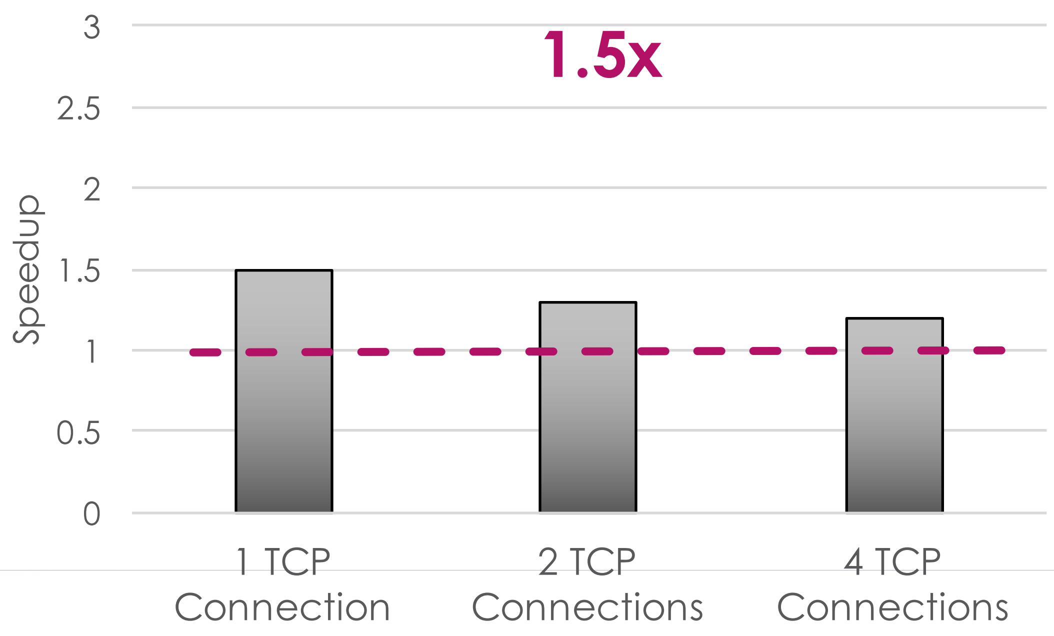


Figure 2. Iperf throughput Improvement of **DPDK-GSO** over **Linux-GSO**

Thanks!

Jiayu Hu

jiayu.hu@intel.com