
DPDK
SUMMIT
Powering the Future of Networking Software
12–13 May 2026
Applying Header–Data Split to Zero-Copy Data Transfer
ByteDance STE - Kernel Network
@Dapeng Sang

Agenda：

- 1. Background
- 2. Technical Design
- 3. Achievements

Background — Disaggregated TCP Network Protocol Stack

DPDK TCP Network Protocol Stack

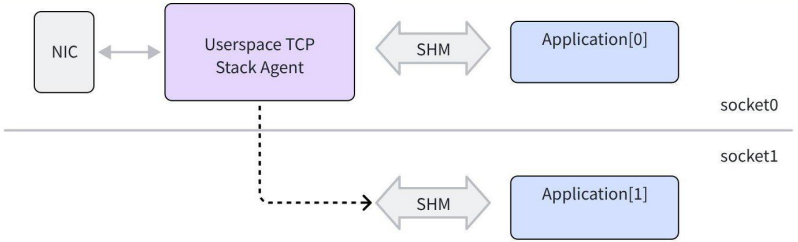
- A DPDK-based TCP network protocol stack, a high-performance network processing architecture that bypass the kernel.

Disaggregated Protocol Processing

- The application and the protocol stack run separately, achieving separation from business logic and independent maintenance.

Multi-Container Network Services

- The protocol stack capabilities are exposed as a service, simultaneously providing network services to multiple containers.



Challenge —— Zero-Copy & Memory-Isolation

- **1. High Performance: Zero-Copy**
  - Avoid data copying. The application and protocol stack communicate via SHM. The NIC DMA directly interacts with the application's payload memory, without CPU involvement in memory copying.
- **2. Security & Stability: Memory Isolation**
  - Memory: Multiple apps share the protocol stack, ensuring that the memory spaces for receiving and transmitting among multiple apps are strictly isolated to prevent data cross-contamination.
  - Permission: The protocol stack has no write permission to data, and headers are invisible to apps.

Agenda：

- 1. Background
- 2. Technical Design
- 3. Achievements

Technical Design —— HDS & Zero-Copy

- Core Principle: Upon packet RX, the NIC DMA writes the protocol header and payload into two physically separate memory.

Header Mempool

- Headers are uniformly managed by the protocol stack and are invisible to applications.
- Centralized and secure protocol parsing.

Payload Mempool

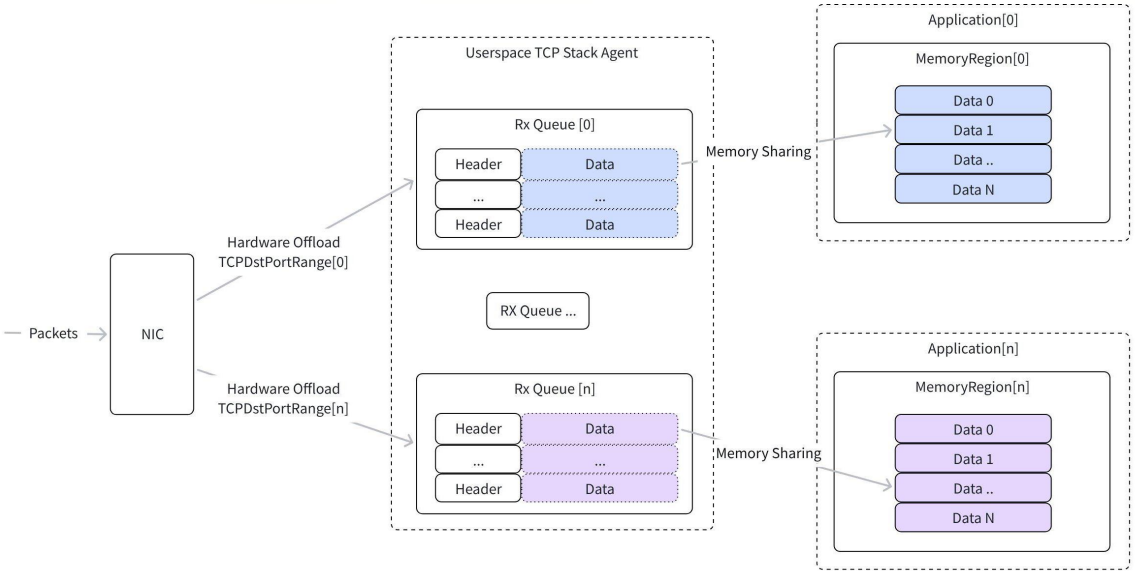
- Applications can flexibly create their own mempools and share them with the protocol stack.
- After the protocol stack completes NIC registration, it retains only READ permission.

Zero-Copy And Isolation

- Zero-Copy: Payload goes directly from NIC to App memory.
- Strong Isolation: Physical memory isolation among multi-Apps, with separation of Header and Payload.

RX Data Path – From NIC to Application

1. **Memory Registration**
2. **NIC Queues and HDS Configuration**
3. **Packet Reception & HDS**
4. **Protocol Stack Parsing**
5. **App Zero-Copy RX**



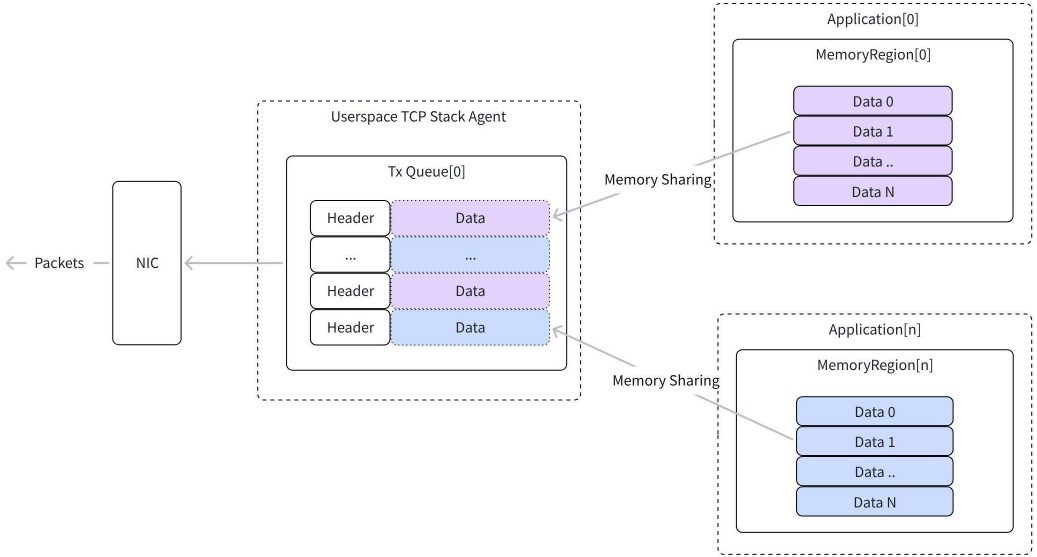
TX Data Path – From Application to NIC

1 **Memory Registration**

2 **NIC Queues Configuration**

3 **App Zero-Copy TX**

4 **Protocol Stack Packet Encapsulation**



Agenda：

- 1. Background
- 2. Technical Design
- 3. Achievements

Benefit one —— Performance

1 Zero-Copy

- Zero-copy between App and NIC queues significantly reduces end-to-end latency, and saves CPU resources otherwise consumed by memcpy.
- Overhead decreased by up to nearly 90%.

2 NUMA Affinity

- Header and Payload can reside on different sockets. DMA handles cross-socket data movement, avoiding expensive remote memory access.
- Throughput Improvement of 2 ~ 10%.

3 Cache Efficiency

- The protocol stack only processes small tcp headers. Payload never enters its cache lines, resulting in significantly improved cache hit rates.
- Cache miss decreased by nearly 90%.

Benefit two —— Safety & Stability

1 Independent Mempool

- Memory mappings are independent of each other, ensuring data isolation.

2 Ownership Boundary

- App owns the payload, and the protocol stack owns the header.

3 Permission Control

- After the protocol stack completes device memory registration, it retains only read permission on the payload.

Summary：

- HDS is applied for zero-copy in the disaggregated TCP network protocol stack
- Performance Optimization: Zero-Copy + Cache-Friendly + NUMA Affinity
- Multi-App Security: Independent Memory Pools + Read Permission Isolation + Fault Isolation

Thank you – Questions are welcome

ByteDance STE - Kernel Network