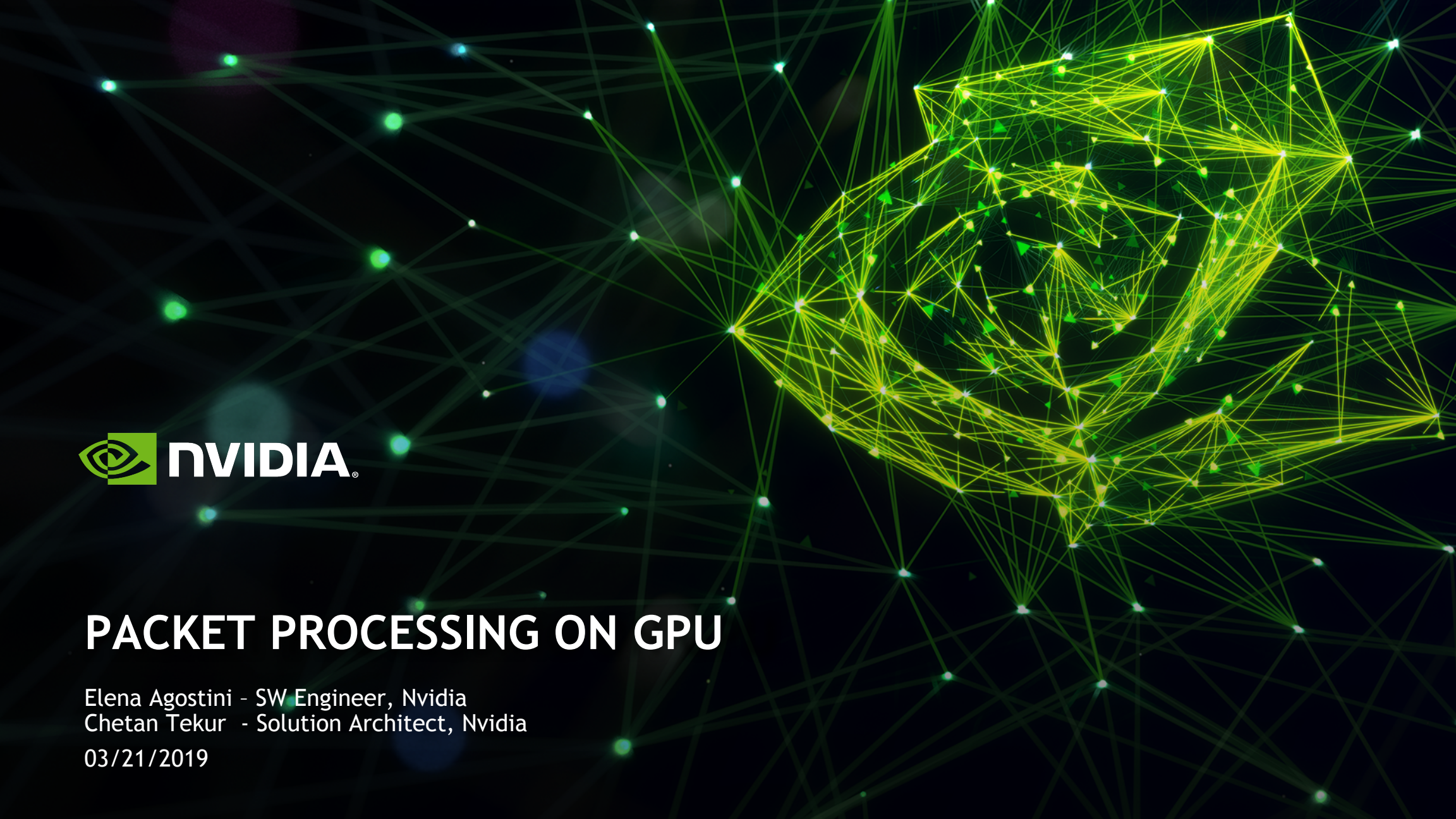
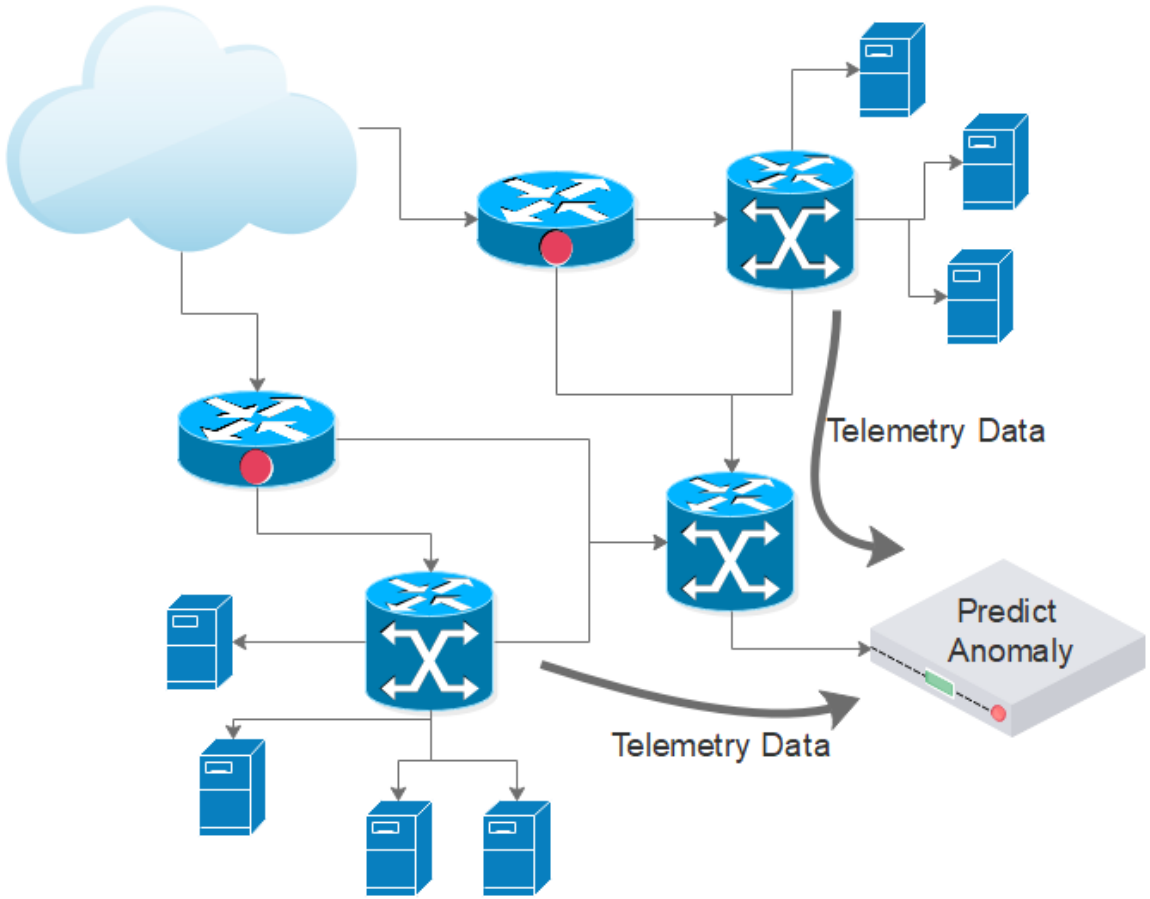


PACKET PROCESSING ON GPU

Elena Agostini – SW Engineer, Nvidia
Chetan Tekur - Solution Architect, Nvidia

03/21/2019

TELEMETRY DATA ANALYSIS



RESEARCH PAPERS

APUNet: Revitalizing GPU as Packet Processing Accelerator

- **Zero-copy packet processing** is highly desirable in APUNet for efficient utilization of the shared memory bandwidth

Exploiting integrated GPUs for network packet processing workloads

- Shared Physical Memory (SPM) and Shared Virtual Memory (SVM)

GASPP: A GPU-Accelerated Stateful Packet Processing Framework

- Combines the massively parallel architecture of GPUs with **10GbE** network interfaces

Fast and flexible: Parallel packet processing with GPUs and click

- Reaching full line rate on **four 10 Gbps NICs**

PacketShader: A GPU-accelerated Software Router

- **40 Gbps** throughput achieved

GTC - 2017

Deep Packet Inspection Using GPUs - Wenji Wu (Fermilab)

Highlights:

- GPUs accelerate network traffic analysis
- I/O architecture to capture and move network traffics from wire into GPU domain
- GPU-accelerated library for network traffic analysis

Features	NPU/ASIC	CPU	GPU
High compute power	Varies	✗	✓
High memory bandwidth	Varies	✗	✓
Easy programmability	✗	✓	✓
Data-parallel execution model	✗	✓	✓

Architecture Comparison

Features	cores	Bandwidth	DP	SP	Power	Price
NVidia K80	4992	480 GB/s	2.91 TF	8.73 TF	300W	$4,349
Intel E7-8890	18	102 GB/s	0.72 TF	1.44 TF	165W	$7,174

NVidia K80 vs. Intel E7-8890

Future Challenges:

- Optimize and evolve the GPU-based **network traffic analysis framework for 40GE/100GE Network**

GTC - 2018

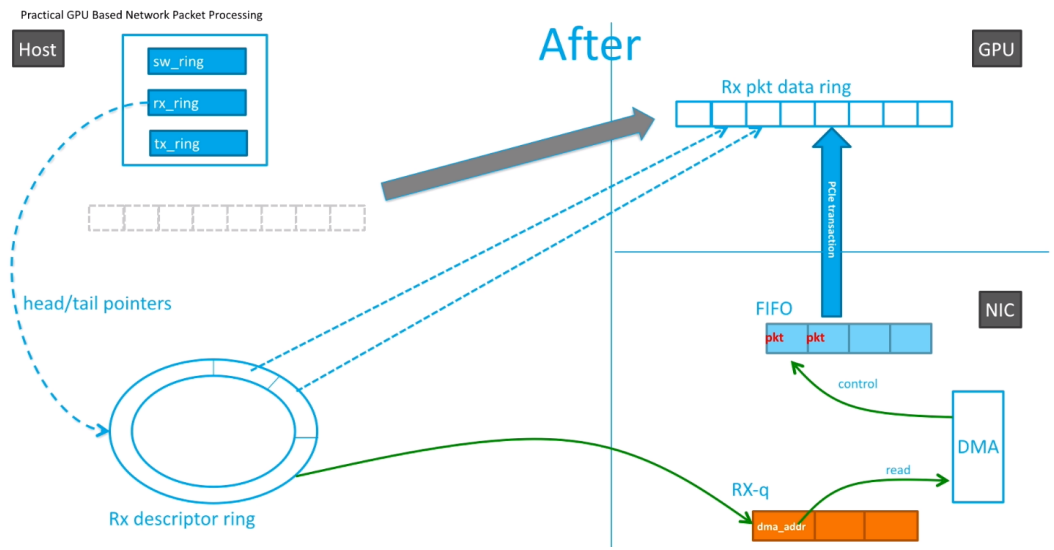
Practical GPU Based Network Packet Processing – Hal Purdy (ATT)

Best Practices and Results:

- Use DPDK
- Minimize data copying
- Stateful, compute intensive processing to GPU
- Reached 100% line rate at 10 GigE

Future Challenges:

- GPU based I/O: **Completely offload CPU**
- Reach line rate at **100 GigE**



HIGH LEVEL PROBLEM STATEMENT

- Building GPU accelerated network functions has its challenges

- Each network function has following recurring tasks:
  - NIC-CPU-GPU or NIC-GPU interaction
  - Pipelining and buffer management
  - Deploying batch or flows to compute cores
  - Low latency and high throughput requirement

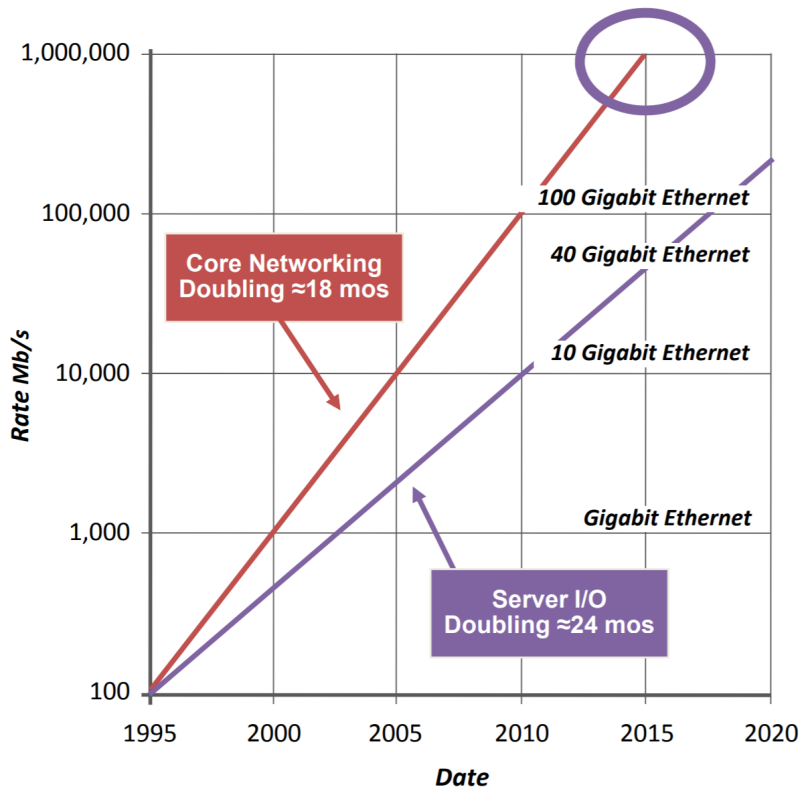
WHY GPU?

MOTIVATION

Problem statement

- BW increase
  - More IO & Memory BW
- Higher perf/cost
  - More compute @ lower cost
- Agility
  - Software Defined Network : Programmability

GPU for Network Packet Processing



Source : IEEE

MOTIVATION

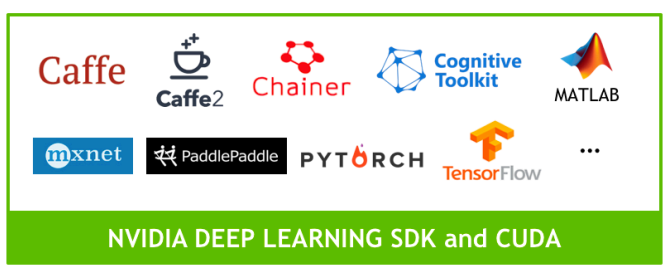
Common Workloads:

- Packet forwarding
- Encryption/Decryption
- Intrusion Detection Systems
- Stateful Traffic Classification
- Pattern matching

Solutions:

- Nvidia supports Machine Learning, Deep Learning and Custom Parallel Programming models

RAPIDS



NVIDIA CUDA


SETTING THE STAGE

GPUDIRECT TECHNOLOGIES

GPUDirect P2P → data

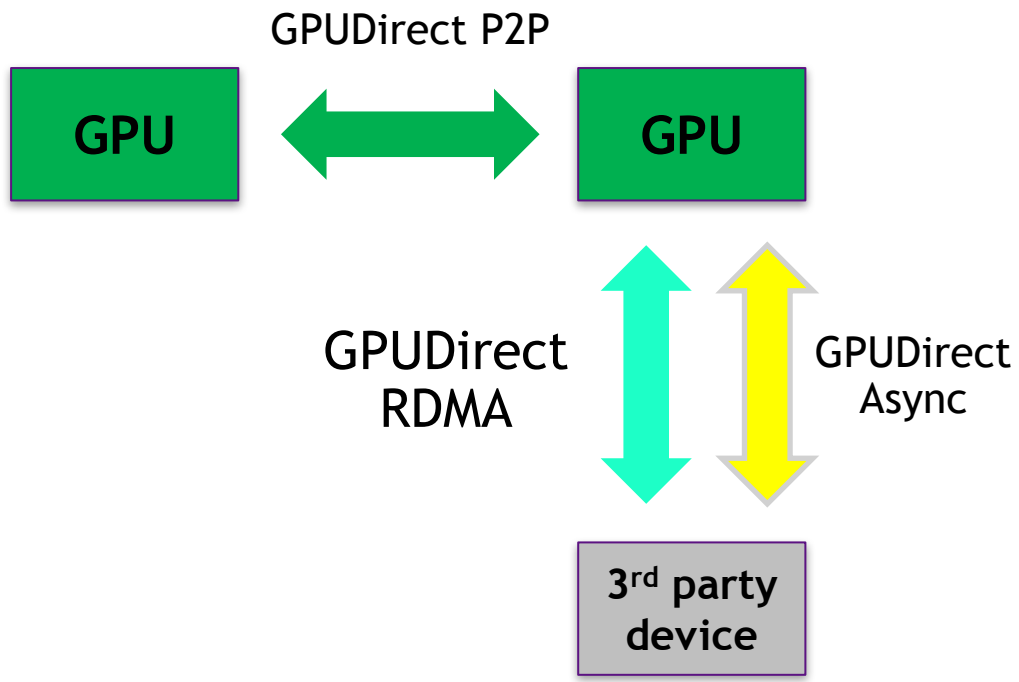
- GPUs master & slave
- Over PCIe, NVLink1, NVLink2

GPUDirect RDMA → data

- GPU slave, 3rd party device master
- Over PCIe, NVLink2

GPUDirect Async → control

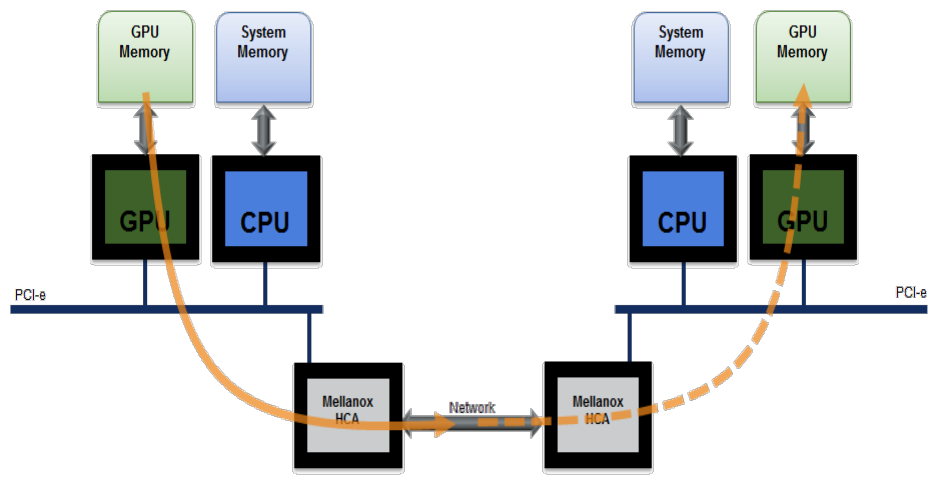
- GPU, 3rd party device, master & slave
- Over PCIe, NVLink2



GPUDIRECT RDMA

Overview

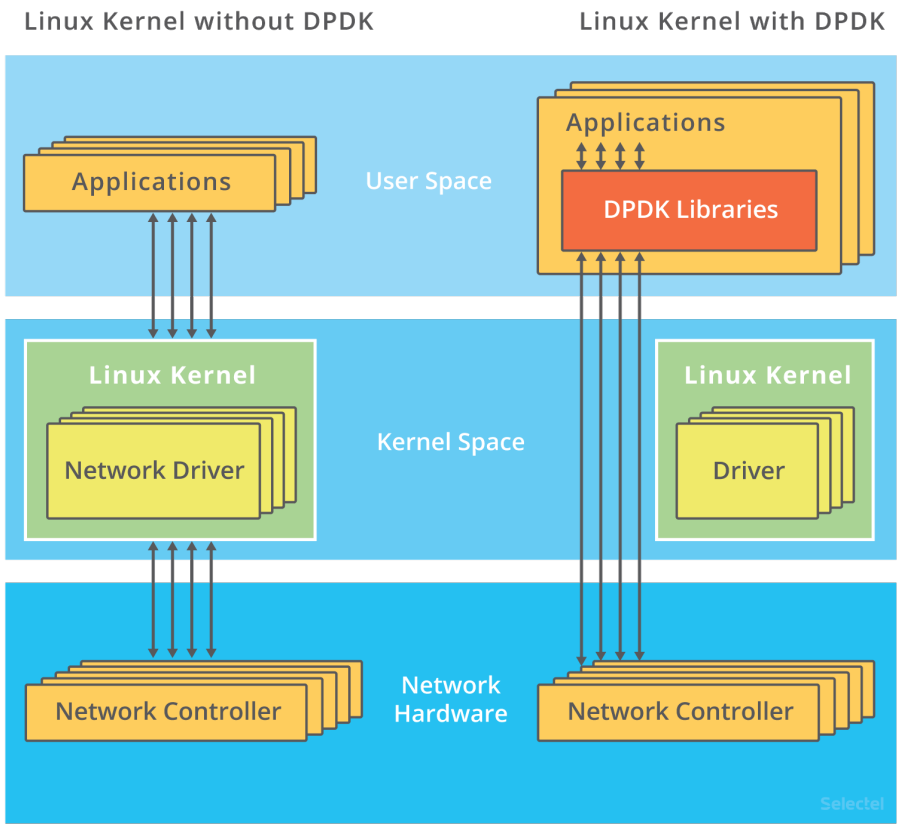
- 3rd party PCIe devices can directly read/write GPU memory
  - e.g. network card
  - GPU and external device must be under the same PCIe root complex
- No unnecessary system memory copies and CPU overhead
  - MPI_Send(gpu_buffer)
- External modules:
  - Mellanox NIC required nv_peer_mem
- https://docs.nvidia.com/cuda/gpudirect-rdma/index.html



DPDK

Data Plane Development Kit

- A set of data plane libraries and network interface controller drivers for fast packet processing
- Provides a programming framework for x86, ARM, and PowerPC processors
- From user space, an application can directly dialog with the NIC
- www.dpdk.org

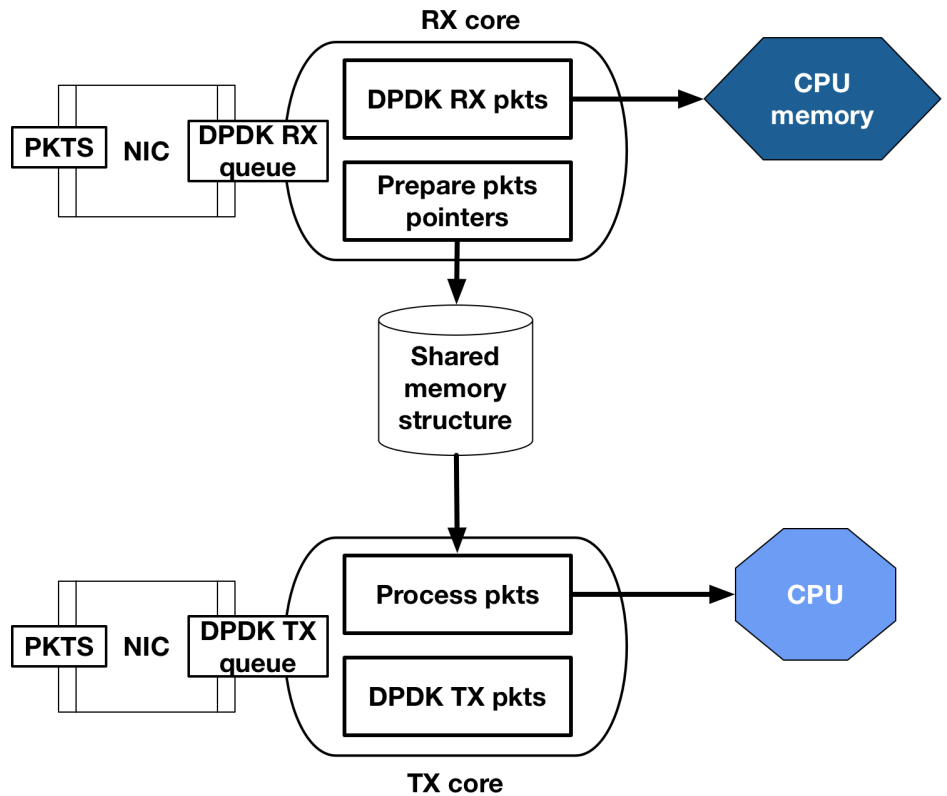


Source: https://blog.selectel.com/introduction-dpdk-architecture-principles/

DPDK

Typical application layout

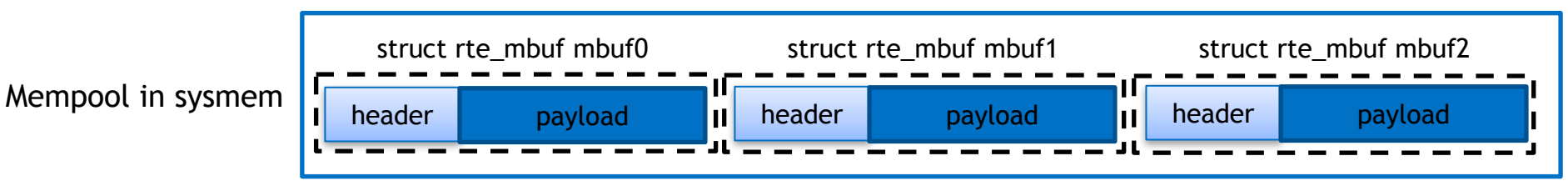
```
device_port = prepare_eth_device();
mp = prepare_mempool();

while(1) {
    //Receive a burst of packets
    packets = rx_burst_packets(device_port, mp);

    //Do some computation with the packets
    compute(packets);

    //Send the modified packets
    tx_burst_packets(packets, device_port);
}
```



DPDK MEMORY MANAGEMENT

Mbufs & Mempool

- The mbuf library provides the ability to allocate and free buffers (mbufs) useful to store network packets
- Mbuf uses the mempool library: an allocator of a fixed-sized object in system memory
  - DPDK makes the use of hugepages (to minimize TLB misses and disallow swapping)
- Each mbuf is divided in 2 parts: header and payload
  - Due to the mempool allocator, headers and payloads are contiguous in the same memory area




DPDK + GPU

DPDK + GPU

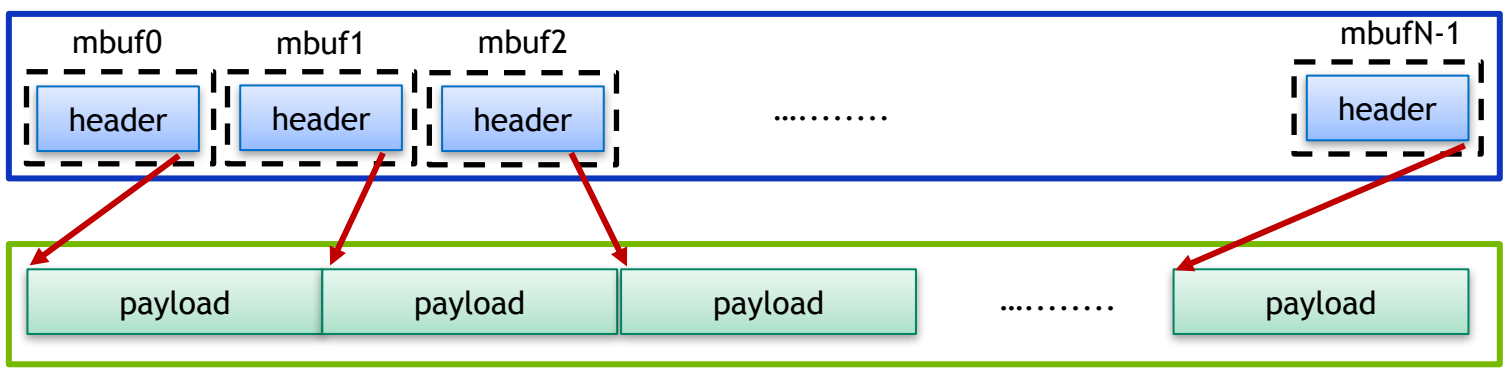
Enhancing original implementation

- Exploit GPU parallelism
  - process in parallel the bursts of received packets with CUDA kernels
- Goal
  - offload workload onto GPU working at line rate
- Need to extend default DPDK
  - Memory management: mempool/mbufs visible from GPU
  - Workload: incoming packets are processed by the GPU
- RX/TX still handled by the CPU
  - GPUDirect Async can't be used here (for the moment...)

DPDK + GPUDIRECT

Memory management: external buffers

- Default DPDK mempool is not enough: mbufs in system (host) virtual memory
- New requirements: mbufs must be reachable from the GPU
- Solution: use external buffers feature (since DPDK 18.05)
  - Mbuf payload resides in a different memory area wrt headers



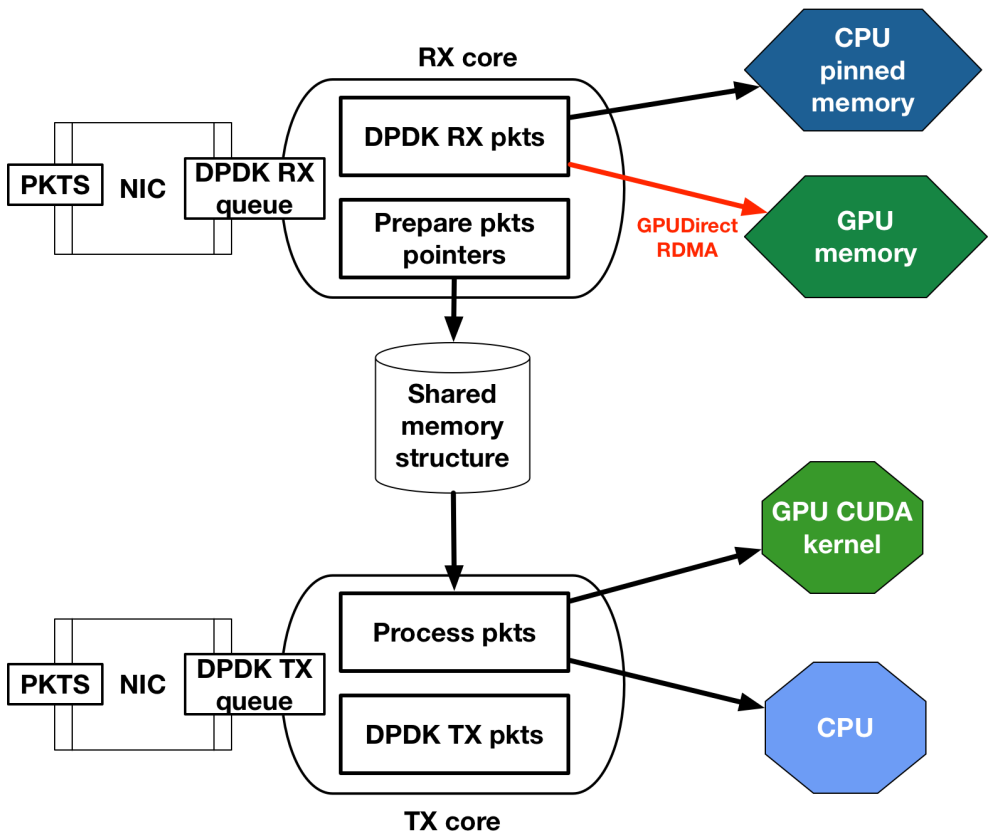
Mempool – host pinned memory only

External memory reachable from GPU:
Host pinned memory or Device memory

DPDK + GPUDIRECT

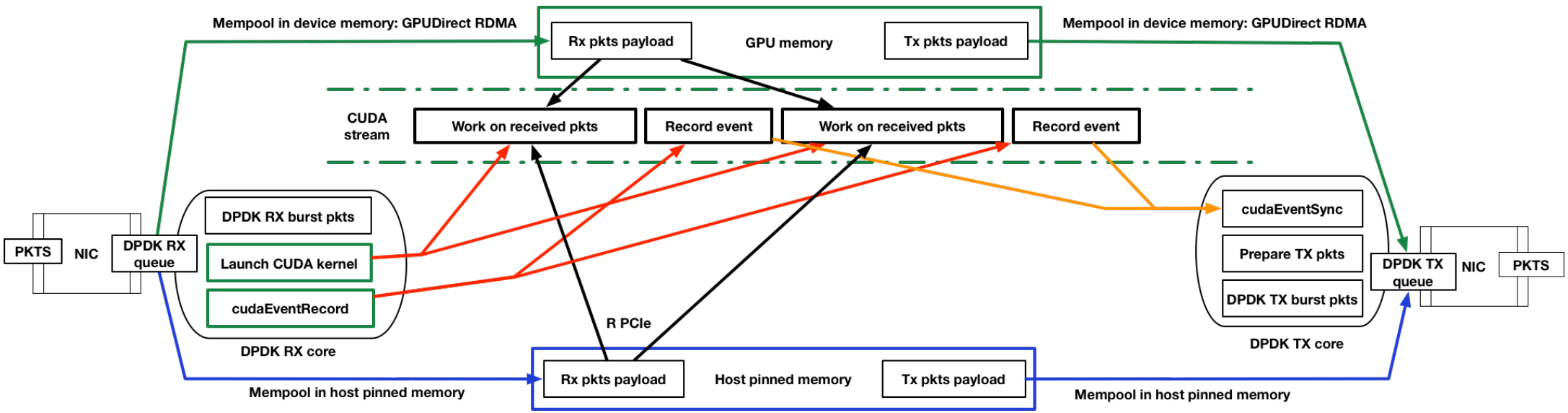
Application workflow

```
device_port = prepare_eth_device();
mp = nv_mempool_create();

while(1) {
    //Receive a burst of packets
    packets = rx_burst_packets(device_port, mp);

    //Do some computation with the packets
    kernel_compute<<<stream>>>(packets);
    wait_kernel(stream);

    //Send the modified packets
    tx_burst_packets(packets, device_port);
}
```



DPDK + GPU

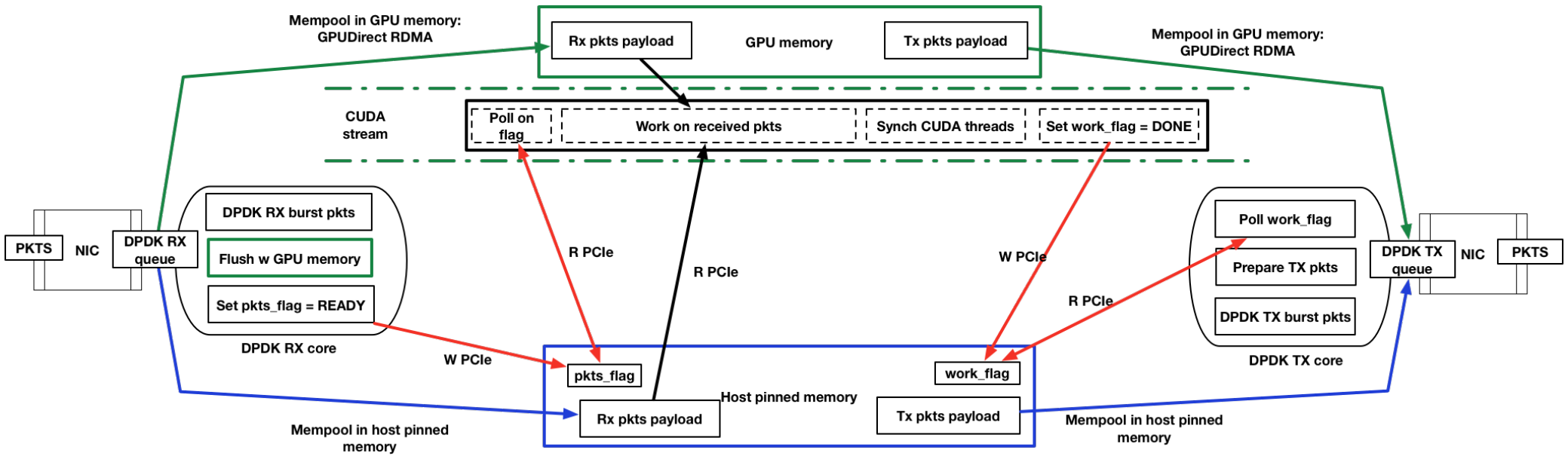
Workload: Multiple CUDA Kernels



- Launch a CUDA kernel as soon as there is a new RX burst of packets
- PCIe transactions only if mempool is in host pinned memory
- Need to hide latency of every (CUDA kernel launch + cudaEventRecord)
- When different CPU RX cores are launching different CUDA kernels there may be CUDA context lock overheads

DPDK + GPU

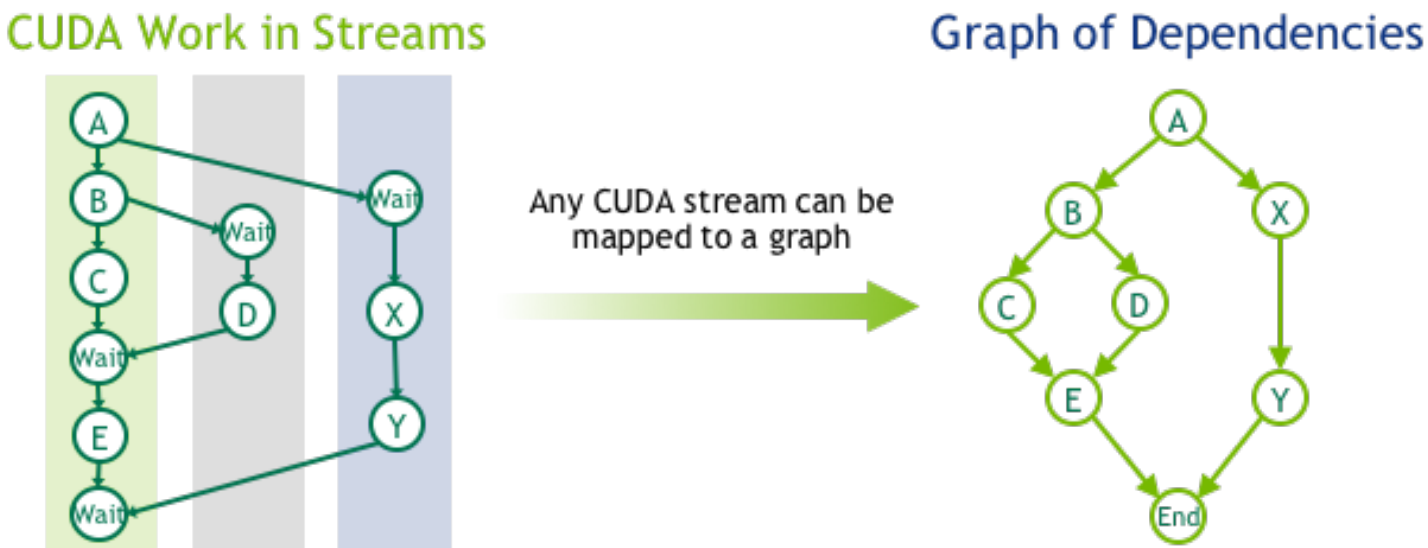
Workload: CUDA Persistent Kernel



- Avoids kernel launch latencies and jitter
- Still incurs latencies for CPU-GPU synchronization over PCIe
- Fixed grid and shared memory configuration for lifetime of the kernel, may not be efficient for all stages of the pipeline
- Harder to leverage CUDA libraries
- With GPUDirect RDMA (GPU memory mempool) you need to "flush" NIC writes into device memory for consistency
  - *S9653 - HOW TO MAKE YOUR LIFE EASIER IN THE AGE OF EXASCALE COMPUTING USING NVIDIA GPUDIRECT TECHNOLOGIES*

DPDK + GPU

Workload: CUDA Graphs



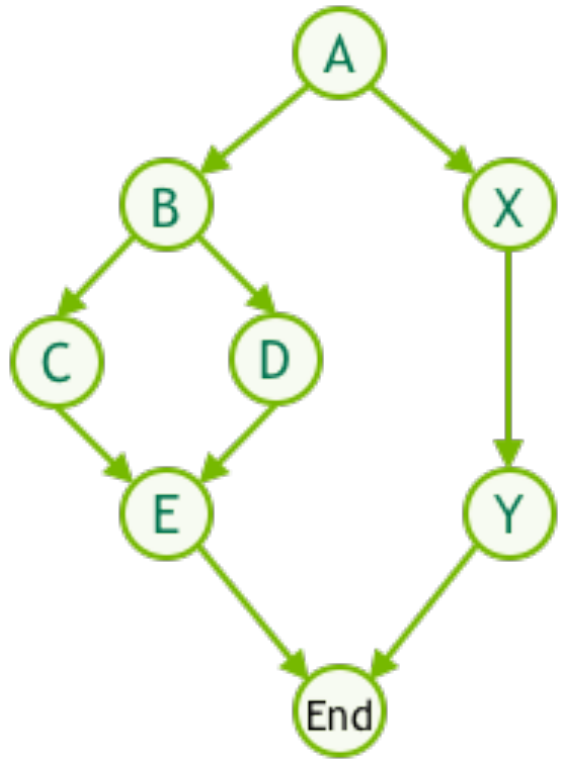
DPDK + GPU

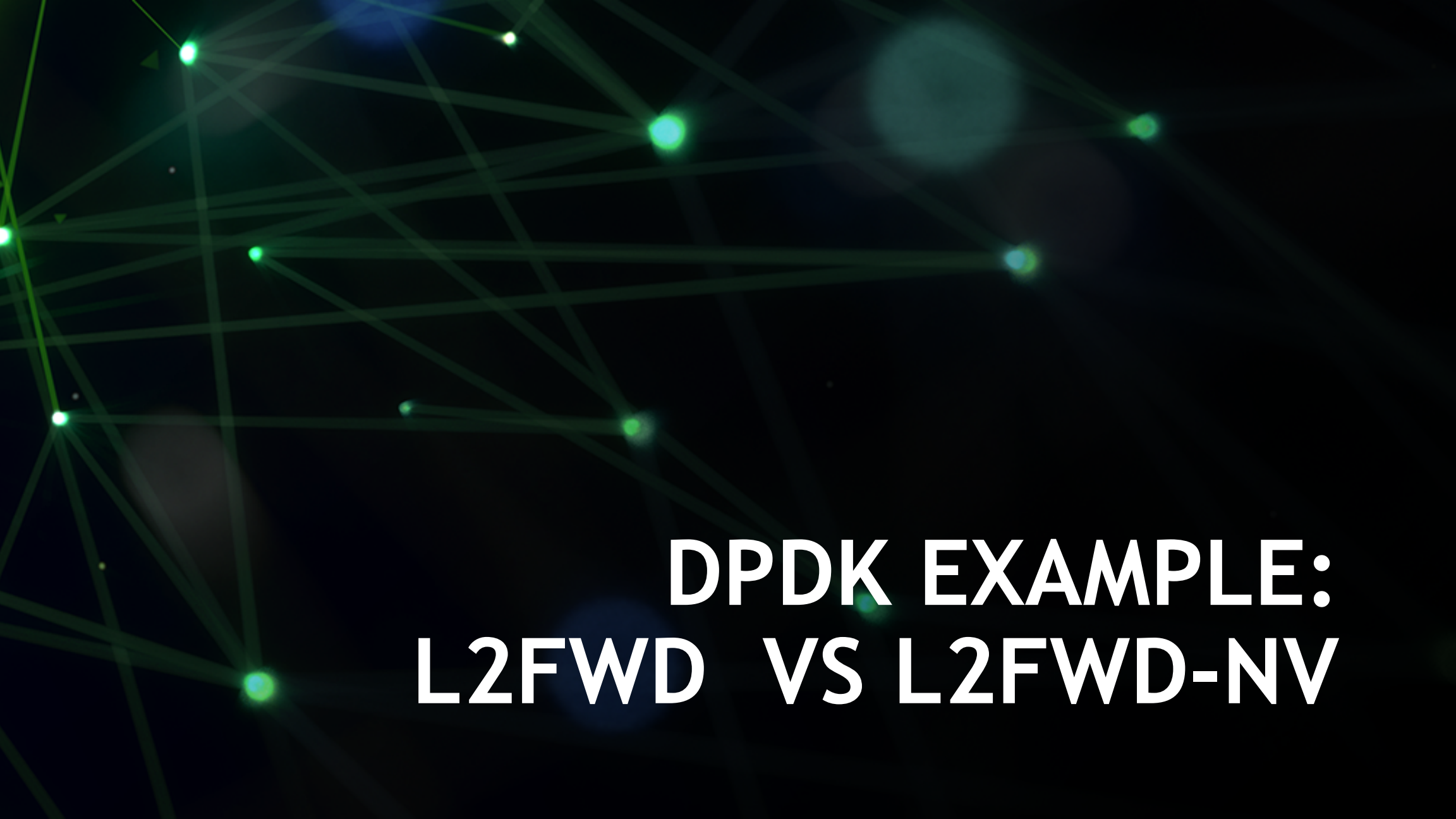
Workload: CUDA Graphs

Sequence of operations, connected by dependencies.

Operations are one of:

Kernel Launch	CUDA kernel running on GPU
CPU Function Call	Callback function on CPU
Memcopy/Memset	GPU data management
Sub-Graph	Graphs are hierarchical



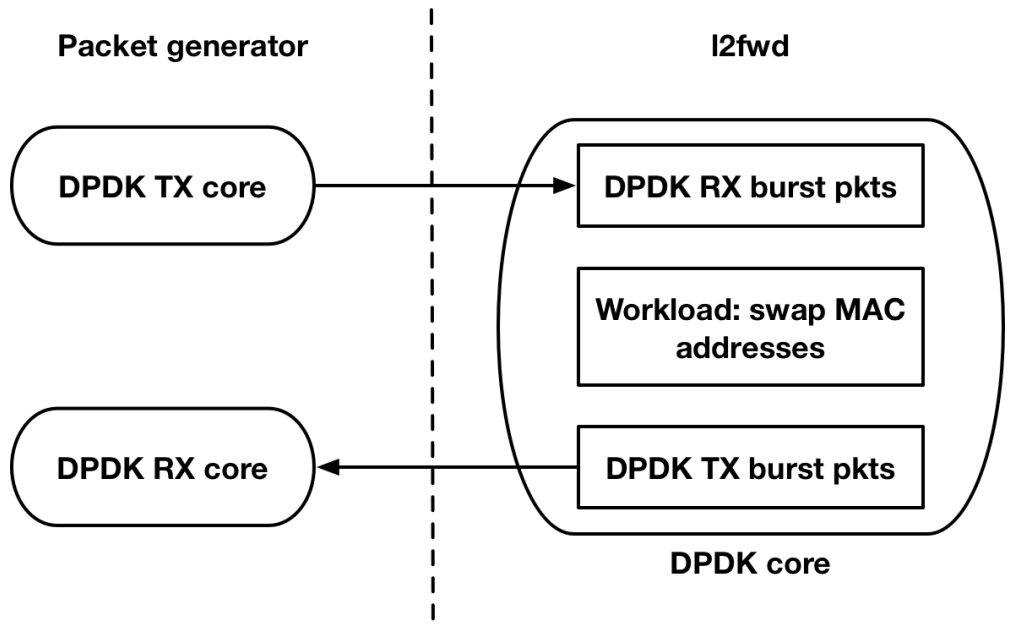


DPDK EXAMPLE: L2FWD VS L2FWD-NV

L2FWD

Workload on CPU

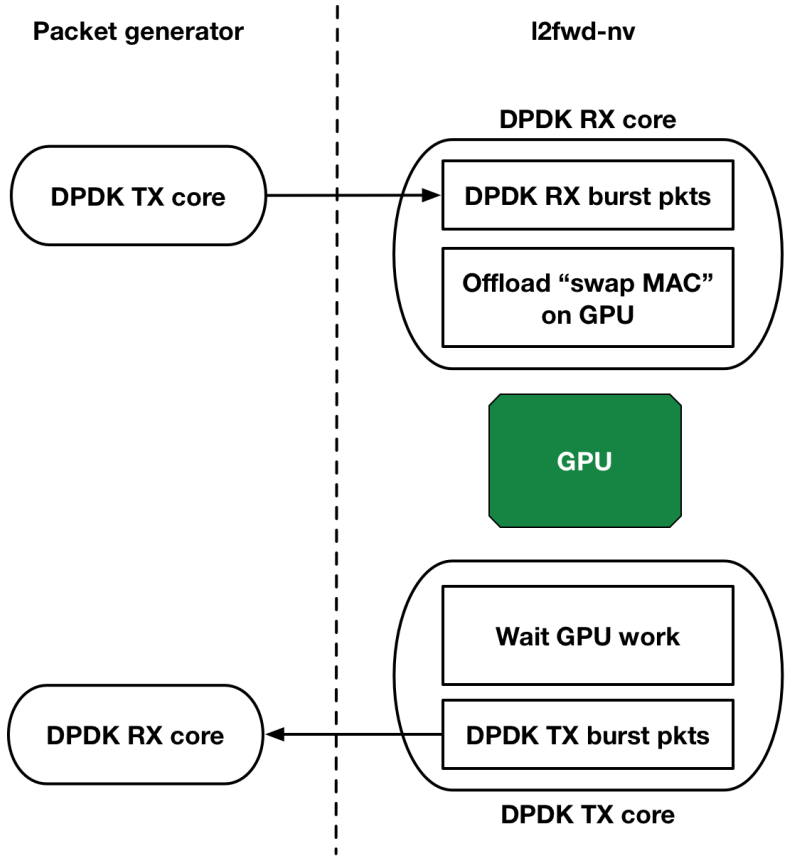
- Vanilla DPDK simple example
- L2fwd workflow:
  - RX a burst of packets
  - Swap MAC addresses (src/dst) in each packets
    - Initial bytes of packet payload
  - TX modified packets back to the source
- No overlap between computation and communication
- Packet generator: testpmd



L2FWD-NV

Workload on GPU

- Enhance vanilla DPDK l2fwd with NV API and GPU workflow
- Goals:
  - Work at line rate (hiding GPU latencies)
  - Show a practical example of DPDK + GPU
- Mempool allocated with *nv_mempool_create()*
- 2 DPDK cores:
  - RX and offload workload on GPU
  - Wait for the GPU and TX back packets
- Packet generator: testpmd
- Not the best example:
  - Swap MAC workload is trivial
  - Hard to overlap with communications



L2FWD-NV PERFORMANCE

HW configuration

- Testpmd as packet generator
- Two Supermicro 4029GP-TRT2
  - Connected back-to-back
  - Ubuntu 16.04
  - CPU: Intel(R) Xeon(R) Platinum 8168 CPU @ 2.70GHz
  - GPU: Tesla V100, CUDA 10, NVIDIA driver 410
  - NIC: Mellanox ConnectX-5 (100 Gbps) with MOFED 4.4
  - PCIe: MaxPayload 256 bytes, MaxReadReq 1024 bytes

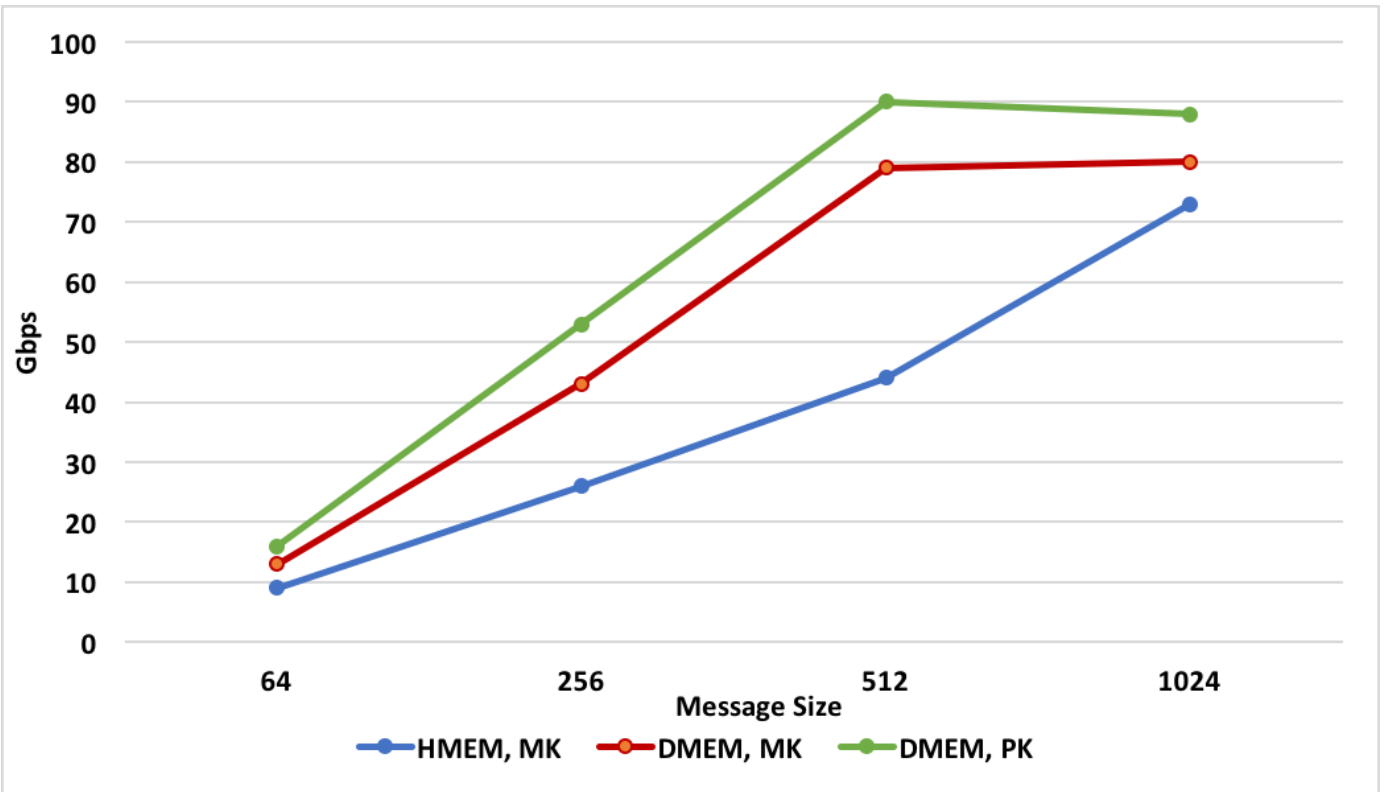
- l2fwd-nv parameters:
  - 8 cores (4 RX , 4 TX)
  - 64 and 128 pkts x burst

- One mempool for all the DPDK RX/TX queues

L2FWD-NV PERFORMANCE

Data rate

- Receiving data in GPU memory always the better solution
  - GPUDirect RDMA required
- With small messages < 512
  - does not inline data in GPU memory
  - exploring design options
- Persistent kernel shows 10% better performance
- But significantly more complex to use
  - L2FWD has trivial compute
  - Latencies get overlapped with larger workloads
  - Regular kernels are flexible and can give similar performance



L2FWD-NV PERFORMANCE

Additional considerations

- With Intel NICs:
  - Ethernet Controller 10 Gigabit X540-AT2
  - Ethernet Controller XL710 for 40GbE QSFP+
  - Line rate reached, no packet loss

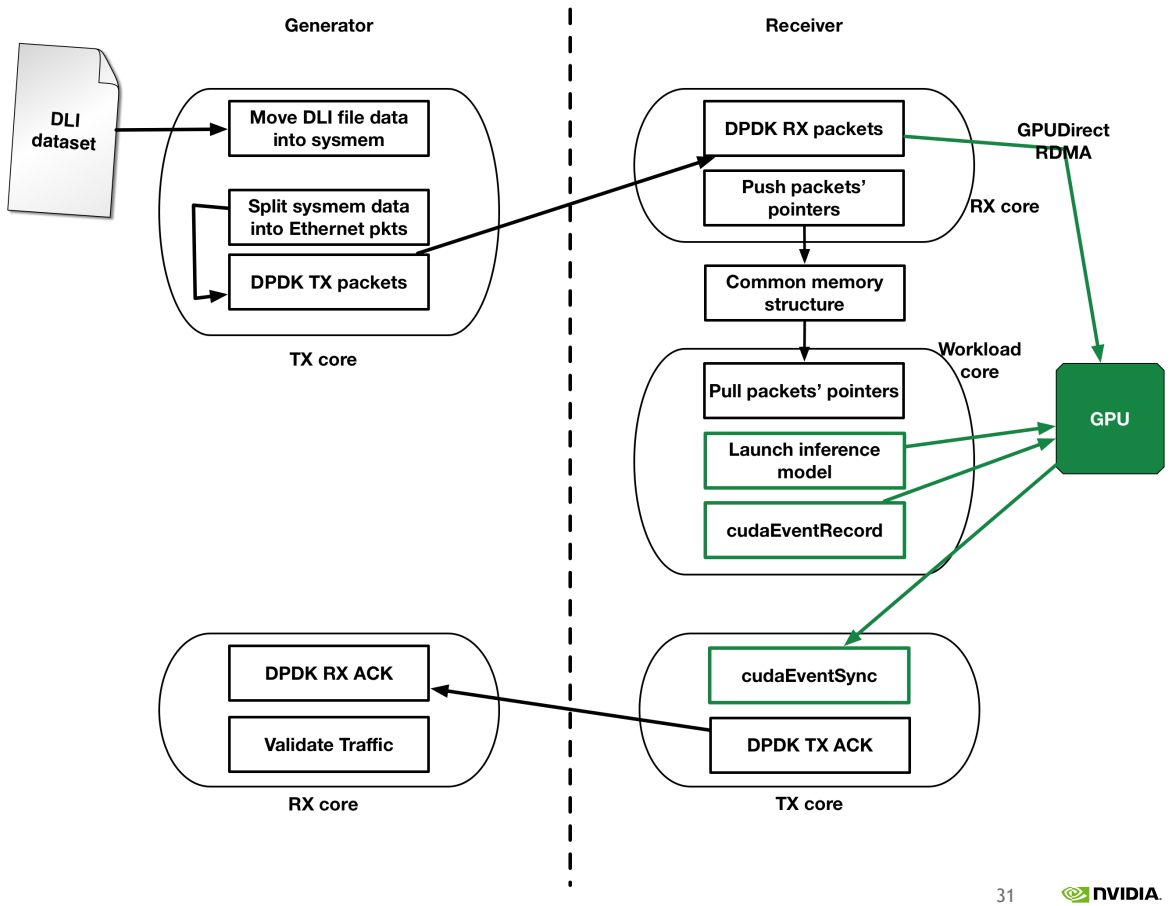
- With large messages (> 1024):
  - Jumbo frames?


DPDK GPU + TELECOM
ANOMALY DETECTION

DESIGN OVERVIEW

Generator - Receiver

- The generator keeps sending packets simulating continuous network flow
- The receiver has 3 DPDK cores:
  - RX and prepare packets
  - Trigger the inference model
    - Can't use persistent kernel
  - TX ACK back: is this anomalous traffic?
- Overlap between computation and communications




CONCLUSIONS

CONCLUSIONS

Next steps

- Continue optimizations for throughput – CUDA graphs, inlining
- Implement Anomaly detection based on the work done for DLI course
- Looking to collaborate with Industry partners to accelerate more workloads. Please reach out to us or Manish Harsh, mharsh@nvidia.com Global Developer Relations, Telecoms


NVIDIA