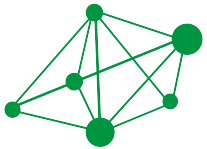
net_mdev: userland network IO

Fosdem 2018
Ilias Apalodimas
Mykyta Iziumtsev
François-Frédéric Ozog



LNG
Networking

Why userland network IO?

Time sensitive networking

- Minority of applications need 1µs latency 1µs delay
- adapter-adapter latency across 5 cut-through switches can be 1µs
- adapter-application latency with 500MHz-1Ghz processor:
  20-40µs, jitter 200-600µs!

Dual stack and drastically reduce driver building/maintenance for ODP, DPDK, VPP

- Best of both worlds

Goals

Generic

- *Any* IO model usable by DPDK, ODP, VPP, any other app

zero copy

- 100Gbps: 148Mpps, 15GB/s ~ 1 DDR4 channel
- Ring desc + packet + virtual desc (+ packet) -> 3(4) DDR4 channels

secure

- IOMMU is a minimum

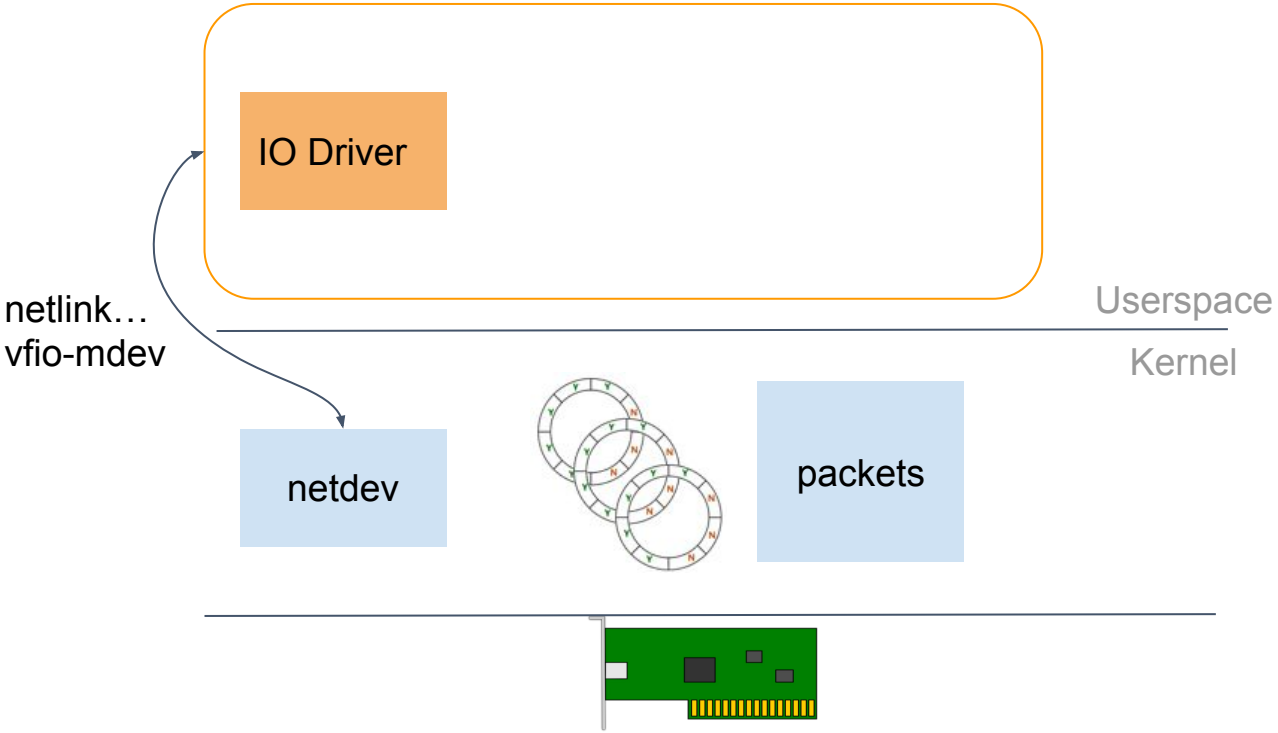
userland network IO

- No userland device drivers, hw revision/flavours insensitive and keep netdevs

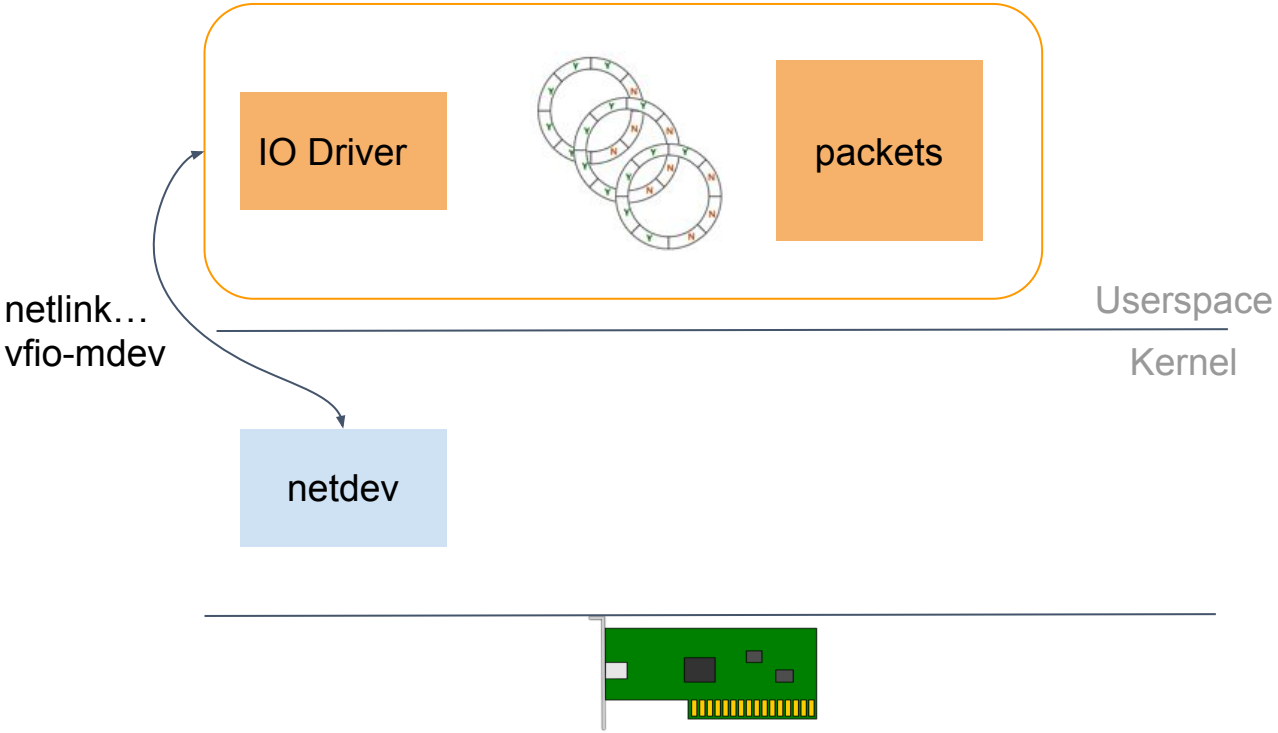
with dual stack capability

- Kernel and userland collaborate in different schemes

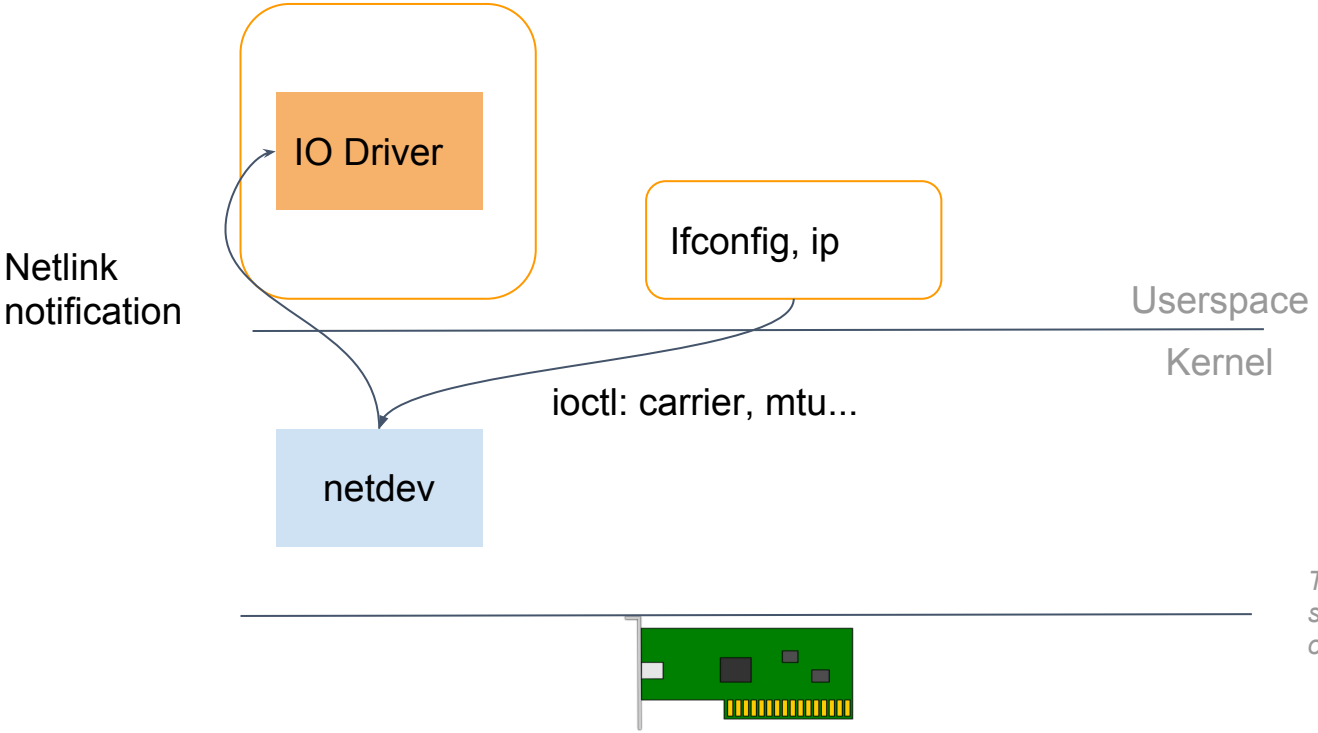
net_mdev


IO Driver
netlink…
vfio-mdev
Userspace
Kernel
netdev
packets

net_mdev


IO Driver
packets
netlink…
vfio-mdev
Userspace
Kernel
netdev

Operations: traditional command line

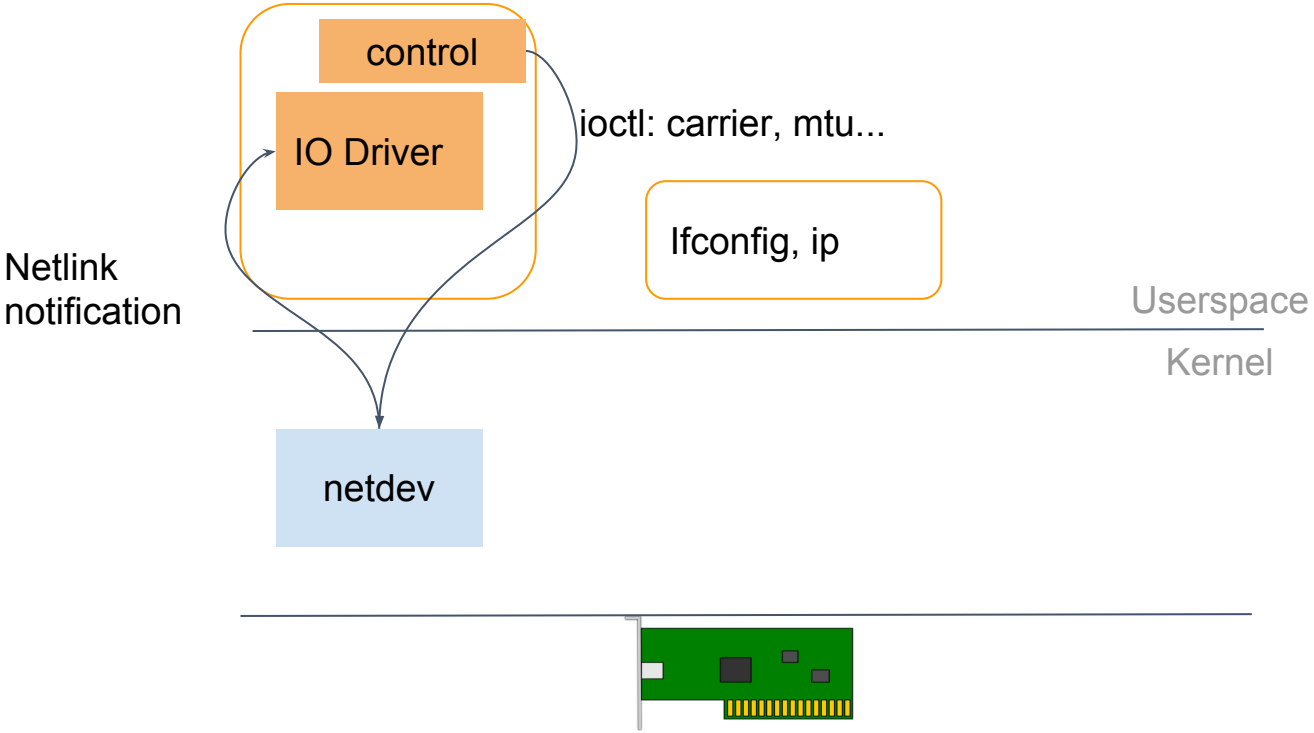


Tcpdump: will require more complex support such as injection channel and other sensing/filtering stuff

Linaro

LNG Networking

Operations: from userland network io



Design options (1/2)

AF_XDP (formerly AF_PACKET v4)

- Accelerators support
- IO models (https://www.spinics.net/lists/netdev/msg481494.html)

DMA Buf

- DMA sync too costly (OK for >=4KB buffers < 1M ops/s)

VFIO

- Loses netdev

VFIO-mdev

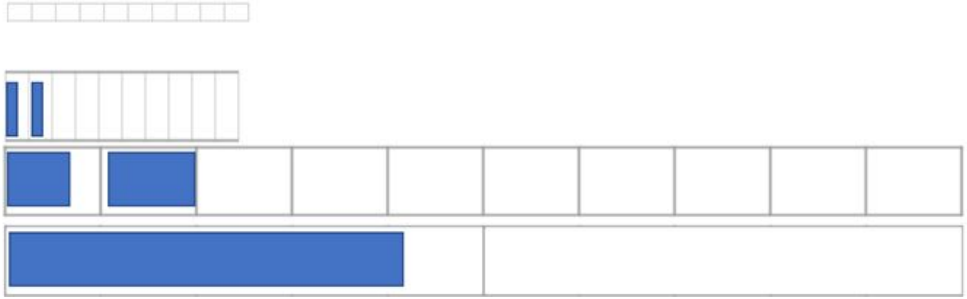
- Technology
  - Introduced in kernel 4.10.
  - Currently supported by Intel i915/QEMU to support virtual GPUs.
  - No real device IO with IOMMU support, just mapping of kernel allocated areas
- Assign queues to VMs through Qemu: Intel/RedHat
- Accelerator access (crypto…): Huawei

Receive packet IO

- Packet Array IO model (majority of PCI NICs), with inline option

Preload descriptors with slot addresses;2MB: 1024 packets

- Multi Packet Array IO model (common in Arm SoCs)



- Tape IO model (Chelsio, Netcope)

Why?
Fat pipe acceleration
Beat PCIe DMA transaction rate
../..

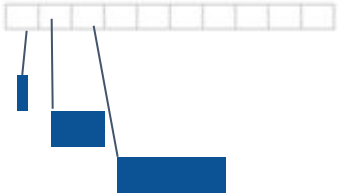
NO descriptors preload; feed HW with unstructured memory 2MB: 32768 packets

Linaro

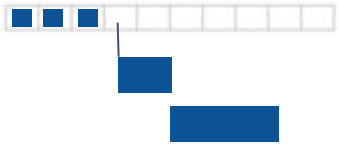
LNG Networking

Transmit packet IO

- Traditional



- Inline



Why?
Beat PCIe DMA transaction rate

Design options (2/2)

AF_PACKET v4

- Accelerators support
- NIC IO models

DMA Buf

- DMA sync too costly (OK for >=4KB buffers < 1M ops/s)

VFIO

- Loses netdev

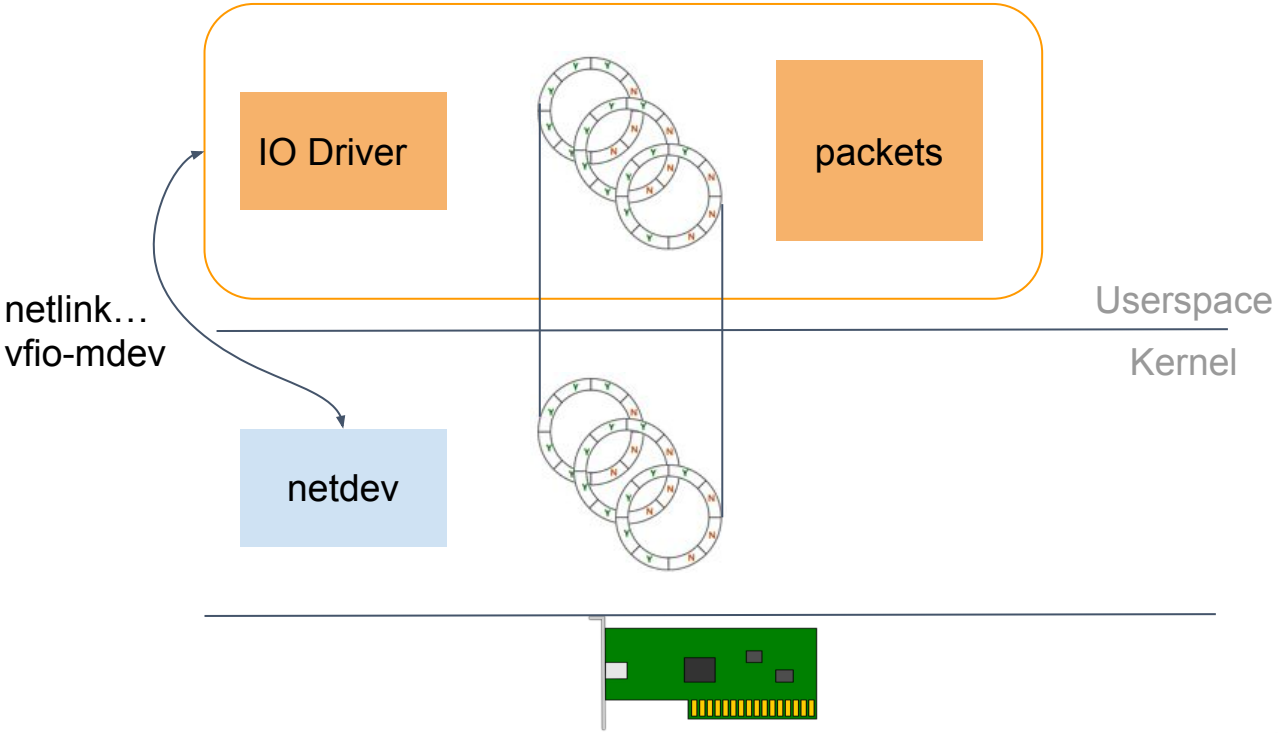
VFIO-mdev

- Technology
  - Introduced in kernel 4.10.
  - Currently supported by Intel i915/QEMU to support virtual GPUs.
  - No real device IO with IOMMU support, just mapping of kernel allocated areas
- Assign queues to VMs through Qemu: Intel/RedHat
- Accelerator access (crypto…): Huawei

From mediated devices to net_mdev

- vfio_mdev
  - Extends VFIO-mdev with IOMMU support
- Design constraints
  - net_mdev module with no impact to kernel code (net_dev_priv_flags: IFF_NET_MDEV)
  - Willing device drivers can leverage it in a “non dependent” manner
    - No module dependency
    - Severe restrict addition of ‘ifs’
- netdev “boilerplate”:
  - Registration…
  - control (mtu, carrier control, statistics are quite generic through netlink)

net_mdev


IO Driver
packets
netlink…
vfio-mdev
Userspace
Kernel
netdev

Operations walk through: kernel side

- Preparation
  - Load driver with global enable parameter net_mdev=1
  - mdev_add_essential(): Added on each NIC driver.
  - Descriptor rings are PAGE_SIZE aligned
  - VFIO-MDEV creates control files in /sys
- Capture the netdev
  - `echo $dev_uuid > /sys/class/net/$intf/device/mdev_supported_types/$sys_drv_name/create`
    - `/sys/bus/mdev/devices/$dev_uuid/netmdev/netdev`
  - Transition
    - Graceful rx/tx shutdown: netif_tx_stop_all_queues…
    - Keep carrier up if possible
    - VFIO-MDEV module sets IFF_NET_MDEV flag.
    - Set hardware in known state (hardware dependent, from clear producer/consumer indexes to full reset, rx at hw level)
    - Set RX interrupts according to polling strategy. Using the IFF_NET_MDEV flag we can intercept the kernel interrupt handler and redirect it to the userspace with eventfd or similar functionality.
  - Inventorize memory regions to be mapped in user-space (Rx/Tx descriptors arrays, doorbells MMIO, memory management MMIO…). Each region is exported using struct vfio_region_info_cap_type from the VFIO-API
  - At this stage kernel cannot do network IO (send/receive packets)

Operations walk through: userland side

- Application start
  - ioctls for VFIO_GROUP_GET_STATUS, VFIO_SET_CONTAINER, VFIO_SET_IOMMU, VFIO_DEVICE_GET_INFO to initialize IOMMU and discover device type (PCI…) and regions
  - ioctl VFIO_DEVICE_GET_REGION_INFO and mmap(net_mdev) each device region
    - Application does not specify physical memory or bus address: just region index
- Packet memory preparation
  - Packet arrays or unstructured memory areas allocation
  - ioctl VFIO_IOMMU_MAP_DMA with mapping parameters (BIDIRECTIONAL…)
  - hardware update: hardware specific
    - Update descriptor rings for packet array type
    - Load free list for tape IO model
  - Signal transition finished (ioctl), kernel does whatever it needs to re-enable packet io
- Network IO
  - RX loop (full poll mode or irqfd), DMA sync if needed
  - Zero-copy or Inline payloads, DMA sync if needed
  - Ring appropriate doorbells
  - Packet life cycle management: hardware specific

Code statistics

- Common kernel: 900
- Common userland: 650

	Original	Kernel Adds	Useland IO Driver
Realtek r8169	10000	*(obsolete)*	*(obsolete)*
Intel e1000e	29800	250	600
Intel xl710	52600	400	650
Chelsio T4/T5/T6	48000	550	950

Performance

NIC	Speed	cores	rx(Mpps)	tx(Mpps)	Max(Mpps)
Intel xl710	40Gbit	3	19	41.55	59.52
Chelsio T5	*T5-40gbit*	*4*	*10.3*	*48*	*59.52*
Chelsio T6	*T6-50Gbps*		*(74.4)*	*(74.4)*	*74.4*

- Intel xl710 was tested on a Core i5 7400 @ 3.0GHz
- Chelsio was tested on Xeon CPU E5-2620 v3 @ 2.40GHz
- Rx direction still under development
- Chelsio T6 is supported, expecting results
- Test implementation with 1Gbit e1000e is getting close to line rate results on a single core

Experience sharing

- Keep ring life cycle in the kernel
  - Complex, no real standard way of doing it, context (carrier...) of creation vary
  - Hardware revision dependent
  - Some hardware need to be turned “off” to allow decommissioning of ring: prefer not to have influence on carrier (for telecom network devices a single carrier event should happen)
- Transition can be very complex
- Single IOVA shared amongst netdev
- Multiport device
  - If PCI, one PCI Config space per port or not
  - Per port MMIO (still single PCI config space)
  - Diverse strategies to operate securely when partial port capture
    - create VFs per port
    - Implement signaling between userland and kernel

User land DMA operations

- Descriptor rings
  - dma_alloc_coherent
    - PAGE_SIZE rounding required for security
    - Either cacheable or not depending on architecture and device
  - Other: not seen
- Packet memory
  - Userland allocated then mapped by vfio_mdev API
  - dma_map_single
  - Synchronization is needed
    - Coherent dma: dma_sync_single_for_* is NOOP
    - Non coherent dma: ioctl is required, batching of operations to allow 148Mpps

What's next?

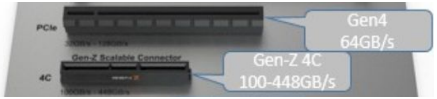
LKML

-> RFC, Intel/Redhat (mdev for Qemu), Huawei (WrapDrive), AF_XDP discussion

Kernel has to protect from devices!

-> IOMMU all the time...

Coherent interconnects (CCIX, OpenCAPI, Intel "*"), Gen-Z



-> hardware and software IO metadata have to be re-architected

Thank You

For further information: www.linaro.org