HKG18-110

net_mdev: Fast-path userspace I/O

Ilias Apalodimas
Mykyta Iziumtsev
François-Frédéric Ozog
Why userland I/O

● **Time sensitive networking**
  ○ Developed mostly for Industrial IOT, automotive and audio/video applications requiring low latency and jitter
  ○ Some of applications need 1 microsecond latency
  ○ Adapter-adapter latency across 5 cut-through switches can be 1 microsecond
  ○ Adapter-application latency with 500MHz-1Ghz processor: 20-40µs, jitter 200-600µs!
  ○ Adopted by OpenAvnu for Intel i210 (without IOMMU support)

● **High speed networks**
  ○ Kernel can’t do line rate on 40Gbbit
  ○ VFIO-PCI already does that but unmaps the netdevice from the kernel and needs to be reprogrammed from the userspace
Why net_mdev

- Based on MDEV VFIO framework, already a part of the linux kernel (mdev since 4.10)
- Uses IOMMU/SMMU on supported platforms to isolate devices on respective domains
- Greatly simplifies drivers in userspace.
  - VFIO-PCI/UIO needs 30k-50k LOC (userland driver is usually a copy of the kernel driver)
  - MDEV needs ~1500 LOC for a full driver
  - Don’t have to deal with complex hardware revisions, erratas, device resets etc. Kernel is still in charge
  - Kernel driver changes required to enable mdev are usually less than 500 LOC
- Can expand to much more than just ethernet (Huawei is having a similar approach for crypto, Intel is doing in on i915 GPU)
Why net_mdev

- Can support zero-copy
- Best of both worlds, you get to keep existing userspace tools
  - ifconfig/ethtool/iproute2 continue to work
  - tcpdump can be easily ported
- Freedom to use “vendor specific” memory schemes
  - Intel uses “slot mode”
  - Chelsio is using “tape mode” and coalesces Tx
- Bus agnostic (PCIe, DPAA etc)
Goals and Limits

- Any IO model usable by DPDK, ODP, VPP, any other app (AF_XDP is limited to one model at the moment, check https://www.spinics.net/lists/netdev/msg481758.html)
- Line rate even on 100Gbps (148Mpps)
- IOMMU/SMMU is a must when possible
- Expand to more than NICs (upstreamed mdev is used on i915 gpu)
- Provide a user-space API
- Limit to cache-coherent hardware (not all architectures support unprivileged cache flushing/invalidating and doing syscalls for syncing will kill performance)
Code statistics

- Current implementation does not support all hardware features

<table>
<thead>
<tr>
<th>NIC</th>
<th>Original</th>
<th>Kernel</th>
<th>Userspace</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chelsio T4/T5(10/40Gbit)</td>
<td>48KLOC</td>
<td>550LOC</td>
<td>950LOC</td>
</tr>
<tr>
<td>XL710(40Gbit)</td>
<td>52KLOC</td>
<td>400LOC</td>
<td>650LOC</td>
</tr>
<tr>
<td>e1000e(1Gbit)</td>
<td>30KLOC</td>
<td>250LOC</td>
<td>600LOC</td>
</tr>
</tbody>
</table>
Performance

- Rx not optimized yet (same numbers on 1-8 cpus)
- All results were achieved using 4 cpus (3 for Intel)
- XL710 and Chelsio T4 tested on core-i5. Chelsio T5 on Xeon

<table>
<thead>
<tr>
<th>NIC</th>
<th>Speed</th>
<th>Rx</th>
<th>Tx</th>
<th>Line rate (64b packets)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chelsio T4</td>
<td>10Gbit</td>
<td>8Mpps</td>
<td>13Mpps</td>
<td>14.88Mpps</td>
</tr>
<tr>
<td>Chelsio T5</td>
<td>40Gbit</td>
<td>10.3Mpps</td>
<td>48Mpps</td>
<td>59.52Mpps</td>
</tr>
<tr>
<td>Intel XL 710</td>
<td>40Gbit</td>
<td>19Mpps</td>
<td>41.55Mpps</td>
<td>59.52Mpps</td>
</tr>
</tbody>
</table>
Architecture

- ODP/VPP/DPDK
- MDEV lib
- User-space
- Kernel driver
- Kernel
Key differences

NOW

User-space

Kernel

VFIO-PCI

User-space

Kernel

SKBs

Userspace buffers

Circular buffer

Head (extract)

Tail (insert)

Circular buffer

Head (extract)

Tail (insert)
Key differences

MDEV

User-space Buffers (will change)

Kernel

Circular buffer

Head (extract)

Tail (insert)
IOMMU/SMMU/VFIO basics

- Intel(IOMMU)/Arm(SMMU)
- IOMMU group = VFIO group  (R/O, topology dependant, programmed at boot time)
- IOMMU domains = VFIO “container” (multiple domains can exist in a container)
- Groups are added to containers, ending up on the same domain
Operational overview (kernel)

1. Allocate Rx/Tx descriptors
2. Add MDEV regions
3. Wait for transition
4. Set NIC to “transition” status
5. Block Kernel Tx path/Rx irq
6. Receive fresh userspace buffers
7. Packet flow starts

Traffic flowing through kernel until we receive “transition”
Operational walkthrough (kernel)

- Load driver with enable parameter net_mdev=1
- mdev_add_essential(): Added on each NIC driver
  - Inventory memory regions to be mapped in user-space (Rx/Tx descriptors arrays, MMIO for doorbells). Each region is exported using struct vfio_region_info_cap_type from the VFIO-API
  - Currently supported regions are VFIO_NET_MDEV_MMIO, VFIO_NET_MDEV_RX_RING, VFIO_NET_MDEV_TX_RING, VFIO_NET_MDEV_RX_BUFFER_POOL
- VFIO-MDEV creates control files in sysfs
- I/O is handled from the kernel at this point
Operational walkthrough (kernel)

On 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 called “transition state” in the diagram (hardware dependent, from clear producer/consumer indexes to full reset at hw level)
Operational walkthrough (kernel)

- 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
- Allocate new Rx/Tx buffers and memory map them to the user-space
- This is planned to change, user-space will do the allocation, VFIO framework will map it to the hardware
- Kernel can’t do I/O but is still in charge of the device
Operational overview (user-space)

MDEV create → Add groups to VFIO containers → Discover NIC regions & lengths → Iterate over regions and map them → Inform NIC of fresh buffers → Transition complete → Packet flow starts
Operational walkthrough (user-space)

- `echo $dev_uuid > /sys/class/net/$intf/device/mdev_supported_types/$sys_drv_name/create`
- `VFIO_GROUP_GET_STATUS`, Test the group is viable and available
- `VFIO_SET_CONTAINER`, Add the group to the container
  - This adds the device on the proper IOMMU domain
- `VFIO_SET_IOMMU`, Enable the IOMMU model we want
- `VFIO_DEVICE_GET_INFO` discover device type (PCI…) and regions (Rx/Tx/MMIO)
- `VFIODEVICE_GET_REGION_INFO` get type, size and mmap each device region
Operational walkthrough (user-space)

- Packet memory preparation
- Packet arrays or unstructured memory areas allocation
- `ioctl VFIO_IOMMU_MAP_DMA` with mapping parameters (BIDIRECTIONAL…).
- Hardware update: hardware specific (populate memory area, ring doorbells)
- Signal transition finished (`ioctl`), kernel does whatever it needs to resume operation
Future development

- Allocate memory from userspace and map it to IOMMU
  - Will require small changes to VFIO API
  - Need to be able to assign devices to domains without detaching them from the kernel
- Expand net_mdev to more than NICs
- DCA (Direct Cache Access) / DDIO (Direct Data I/O) usage
- Check Arm SMMUv3/SVM RFC (Mellanox has PASID NICs instead of SR-IOV)
- Collaborate with Huawei. Wrapdrive has a similar approach for crypto devices
Resources

● POC driver for affordable e1000e NIC
● MDEV linux modifications
  https://github.com/apalos/odp-linux-mdev
● MDEV in ODP
  https://github.com/Linaro/odp/tree/caterpillar
Thank You

#HKG18
HKG18 keynotes and videos on: connect.linaro.org
For further information: www.linaro.org