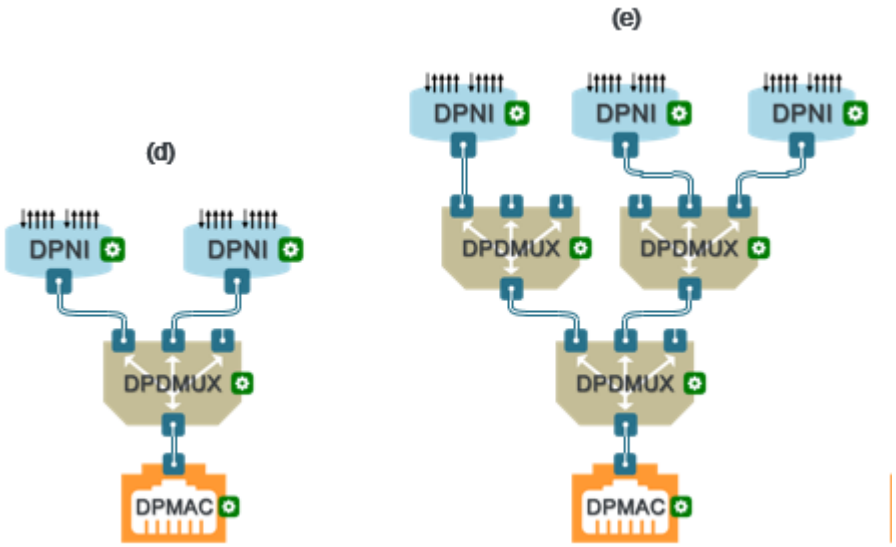
Traffic Bifurcation Between DPDK and Linux Kernel using DPDMUX

DPDMUX a device like DPSW (Switch) which allows switching of packets within the DPAA2 blocks. This application add DPDMUX support in DPDK, uses LS2088ARDB as an example platform for demonstrating the use case that traffic bifurcation between DPDK and Linux Kernel using DPDMUX on DPAA2 platform.

Introduction to DPDMUX

DPDMUX or Data Path network DeMUX is a device like DPSW (Switch) which allows switching of packets within the DPAA2 blocks. DPDMUX has a single “uplink” port and multiple internal interfaces. The uplink interface can be external or internal interface.
DPDMUX has an internal database which classifies the received frames (from any interface) and sends to either uplink or internal interfaces. But, unlike DPSW, DPDMUX doesn’t support automatic route learning – but, it can learn MAC address and VLAN IDs from connected DPNI objects. Also, unlike DPSW, there is no aging of database entries. Thus, if pre-added rules or MAC address/VLAN ID of connected DPNIs doesn’t match, packets are dropped.
DPDMUX can be programmed to direct packets based on header fields beyond Layer 2. This DPAA2 block is available in LS2080, LS2088, LS1088 and LX2160.
From the RM, following diagram snippet describes the usage of DPDMUX.

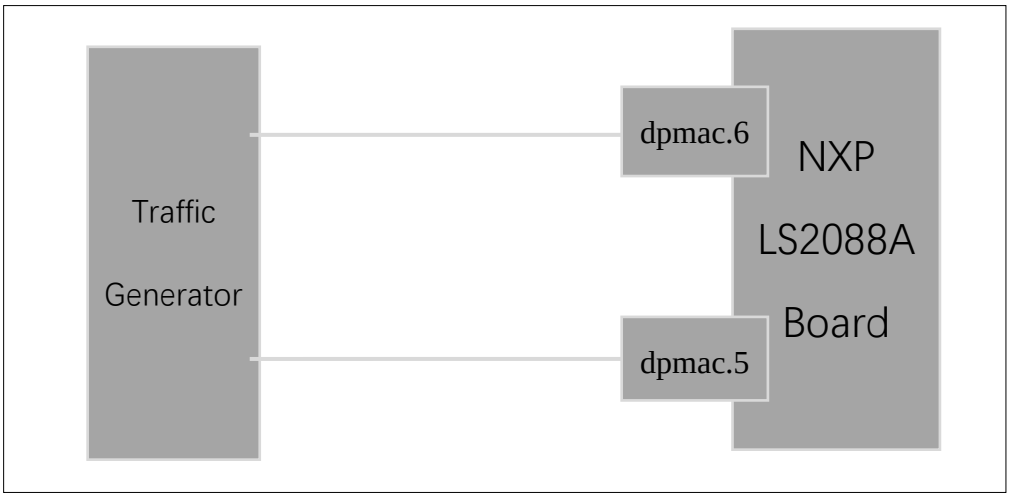


2. Environment Setting up of this application

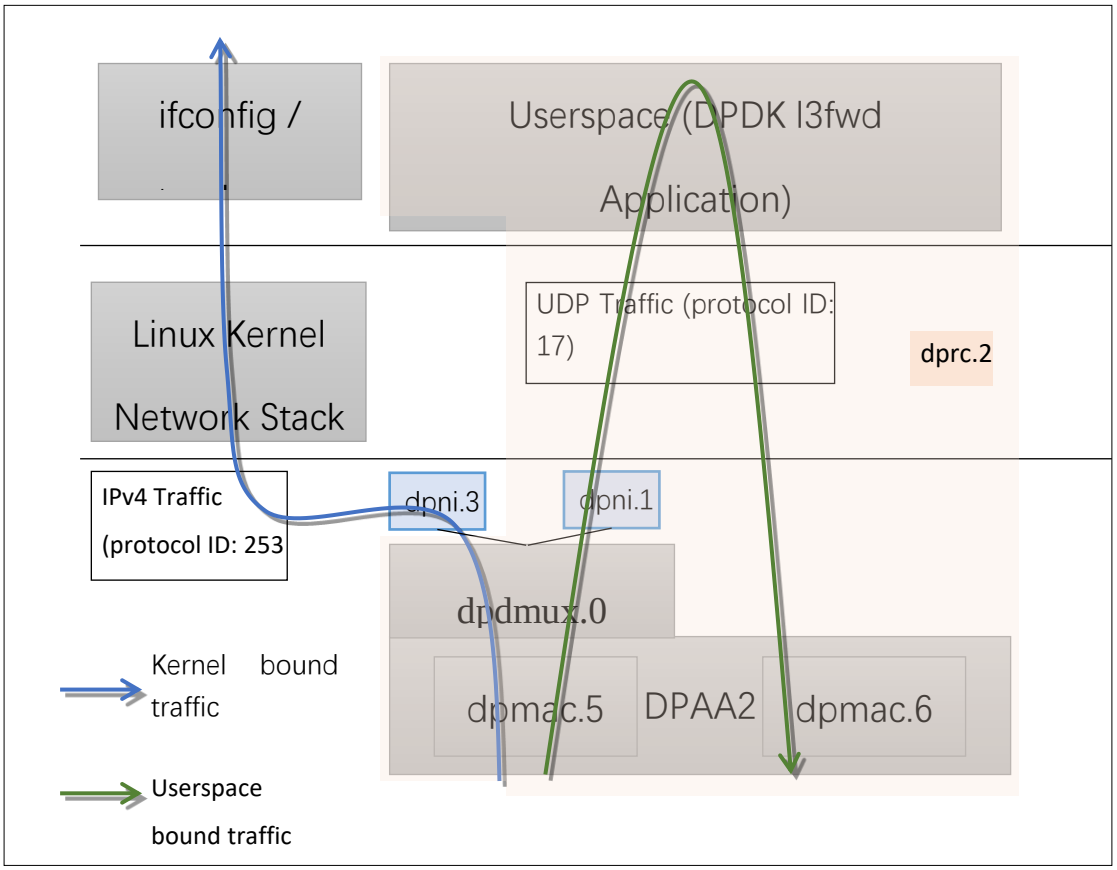
A NXP LS2088A board has been connected to a Packet generator (Spirent) – back-to-back – through two DPAA2 interfaces.

1. Board runs LSDK 18.12 Images
2. DPDK Binary (for Userspace application) is can be built with the attached dpdk source code. This code is different from LSDK 1812 as this contains DPDMUX support.
3. A compiled L3fwd binary has been place in the attachment named “l3fwd.2501”.
4. Spirent TCC, demonstrating the flows created, please refer to the attachment the file is named ls2088ardb-14-idc.tcc.

Flows defined over port connected to DPDMUX are:
UDP traffic with incremental port numbers from 20->30
IPV4 Traffic with incremental protocol numbers from 200->210



NXP LS2088A internal block for traffic bifurcation setup is as the following.



In the above environment setup, a DPRC container (dprc.2) is created containing DPAA2 *dpmac.5* and *dpmac.6* interfaces. DPDMUX *dpdmux.0* is created with *dpni.1* and *dpni.3*, while *dpni.2* is connected with *dpmac.6*.

On a standard LSDK 1812 configuration, these ports are represented using dpmac.X naming. Corresponding to the image above describing the ports, following is the naming convention:

- dpmac.1, dpmac.2, dpmac.3 and dpmac.4 are ETH4, ETH5, ETH6 and ETH7, respectively.
- dpmac.5, dpmac.6, dpmac.7 and dpmac.8 are ETH0, ETH1, ETH2 and ETH3, respectively.

Following are the commands to create the above setup:
Though this section uses dpmac.5 and dpmac.6 as interfaces; similar setup can be created using any other ports of LS2088A
(or any other DPAA2 DPDMUX supporting board). Replace dpmac.X in commands below with equivalent port name.
1. Create DPRC with dpmac.5 and dpmac.6 attached. This would create dpni.1 and dpni.2 internally.
/usr/local/dpdk/dpaa2/dynamic_dpl.sh dpmac.5 dpmac.6

Output would be like:

###################### Container dprc.2 is created ######################

Container dprc.2 have following resources :=>

** 1 DPMCP*

** 16 DPBP*

** 8 DPCON*

** 8 DPSECI*

** 2 DPNI*

** 18 DPIO*

** 2 DPCI*

** 2 DPDMAI*

############################ Configured Interfaces ############################

Interface Name	*Endpoint*	*Mac Address*
dpni.1	*dpmac.5*	*-Dynamic-*
dpni.2	*dpmac.6*	*-Dynamic-*

1. Create a DPNI for assigning to Linux Kernel. This would be used for forwarding the UDP traffic.

   ls-addni --no-link

   Output would be like:

   Created interface: eth0 (obje*ct:dpni.3, endpoint:)*

It is important to note the dpni.X naming which is dynamically generated by the dynamic_dpl.sh script and ls-addni command. In case they are different from what is described in this document, corresponding changes should be done in the commands below.

1. *Unplug t*he DPRC from VFIO, create a DPDMUX, assign DPNIs (dpni.1 and dpni.3) to it, and then plug the DPRC back again to VFIO so that Userspace application can use it. This was already in plugged state because of the dynamic_dpl.sh script.

   # Unbinding dprc.2 from VFIO

   echo dprc.2 > /sys/bus/fsl-mc/drivers/vfio-fsl-mc/unbind

   # Remove dpni.2 from dprc.2 so that it can be assigned to dpdmux

   restool dprc disconnect dprc.2 --endpoint=dpni.1

   # Create dpdmux with CUSTOM flow creation; Flows would be created

 # from the Userspace (DPDK) application

restool dpdmux create --num-ifs=2 --method DPDMUX_METHOD_CUSTOM --manip=DPDMUX_MANIP_NONE --option=DPDMUX_OPT_CLS_MASK_SUPPORT --container=dprc.1

 # Create DPDMUX with two DPNI connections and one DPMAC connection

restool dprc connect dprc.1 --endpoint1=dpdmux.0.0 --endpoint2=dpmac.5

restool dprc connect dprc.1 --endpoint1=dpdmux.0.1 --endpoint2=dpni.3

restool dprc connect dprc.1 --endpoint1=dpdmux.0.2 --endpoint2=dpni.1

restool dprc assign dprc.1 --object=dpdmux.0 --child=dprc.2 --plugged=1

Note that the default queue has been configured as 0.1 in DPDK DPMUX driver. In the above commands, dpni.3 has been configured to --endpoint1=dpdmux.0.1. Thus, all traffic which is not filtered would be sent by dpdmux.0 to dpni.3. Further, the l2fwd application has currently configured UDP traffic (IPv4 Protocol Header field value 17) to be sent to --endpoint1=dpdmux.0.2, which corresponds to dpni.1.

 # Bind the DPRC back to VFIO

echo dprc.2 > /sys/bus/fsl-mc/drivers/vfio-fsl-mc/bind

 # Export the DPRC

export DPRC=dprc.2

If required, IP Address can be assigned to eth0, which would appear in Linux OS to represent the dpni.3. Thereafter, external packet generator or a device can send ICMP traffic to confirm the bifurcation of traffic.

root@Ubuntu:~# ifconfig eth0 1.1.1.10/24 up

root@Ubuntu:~# ifconfig eth0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 1.1.1.10 netmask 255.255.255.0 broadcast 1.0.0.255

inet6 fe80::dce6:feff:fe3a:e105 prefixlen 64 scopeid 0x20<link>

ether de:e6:fe:3a:e1:05 txqueuelen 1000 (Ethernet)

RX packets 0 bytes 0 (0.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 6 bytes 516 (516.0 B)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@Ubuntu:~# restool dpni info dpni.3

dpni version: 7.8

dpni id: 3

plugged state: plugged

endpoint state: 1

endpoint: dpdmux.0.1, link is up

link status: 1 - up

mac address: de:e6:fe:3a:e1:05

dpni_attr.options value is: 0

2. Run the l3fwd application

./l3fwd -c 0xF0 -n 1 -- -p 0x3 --config="(0,0,4),(1,0,5)" –P --traffic-split-proto 17:2

In the command above, “-c 0xF0” corresponds to the cores being used by the DPDK Application. In case they are different, the mask should be changed.
Further, “--config="(0,0,4),(1,0,5)" represents “(Port, Queue, Core)” – which should align with the core masks provided. The Port value is ‘0’ and ‘1’ assuming only dpmac.5 and dpmac.6 have been assigned to the DPRC dprc.2. Only single queue per device has been considered. Numbering for all elements of this tuple starts from 0.

Output would be like:

EAL: Detected 8 lcore(s)

EAL: Probing VFIO support...

EAL: VFIO support initialized

EAL: PCI device 0000:01:00.0 on NUMA socket -1

EAL: Invalid NUMA socket, default to 0

EAL: probe driver: 8086:10d3 net_e1000_em

PMD: dpni.1: netdev created

PMD: dpni.2: netdev created

PMD: dpsec-0 cryptodev created

PMD: dpsec-1 cryptodev created

PMD: dpsec-2 cryptodev created

PMD: dpsec-3 cryptodev created

PMD: dpsec-4 cryptodev created

PMD: dpsec-5 cryptodev created

PMD: dpsec-6 cryptodev created

PMD: dpsec-7 cryptodev created

~L3FWD: Promiscuous mode selected

Splitting traffic on Protocol:17, DPDMUX.0.2

L3FWD: LPM or EM none selected, default LPM on

Initializing port 0 ... Creating queues: nb_rxq=1 nb_txq=4...
Address:00:00:00:00:00:01, Destination:02:00:00:00:00:00, Allocated mbuf pool on socket 0

LPM: Adding route 0x01010100 / 24 (0)

LPM: Adding route 0x02010100 / 24 (1)

LPM: Adding route IPV6 / 48 (0)

LPM: Adding route IPV6 / 48 (1)

txq=4,0,0 txq=5,1,0 txq=6,2,0 txq=7,3,0

Initializing port 1 ... Creating queues: nb_rxq=1 nb_txq=4...
Address:DA:CA:B2:78:68:19, Destination:02:00:00:00:00:01, Allocated mbuf pool on socket 0

txq=4,0,0 txq=5,1,0 txq=6,2,0 txq=7,3,0

Initializing rx queues on lcore 4 ... rxq=0,0,0

Initializing rx queues on lcore 5 ... rxq=1,0,0

Initializing rx queues on lcore 6 ...

Initializing rx queues on lcore 7 ...

Checking link statusdone

Port0 Link Up. Speed 1000 Mbps -full-duplex

Port1 Link Up. Speed 10000 Mbps -full-duplex

L3FWD: entering main loop on lcore 5

L3FWD: -- lcoreid=5 portid=1 rxqueueid=0

L3FWD: lcore 7 has nothing to do

L3FWD: lcore 6 has nothing to do

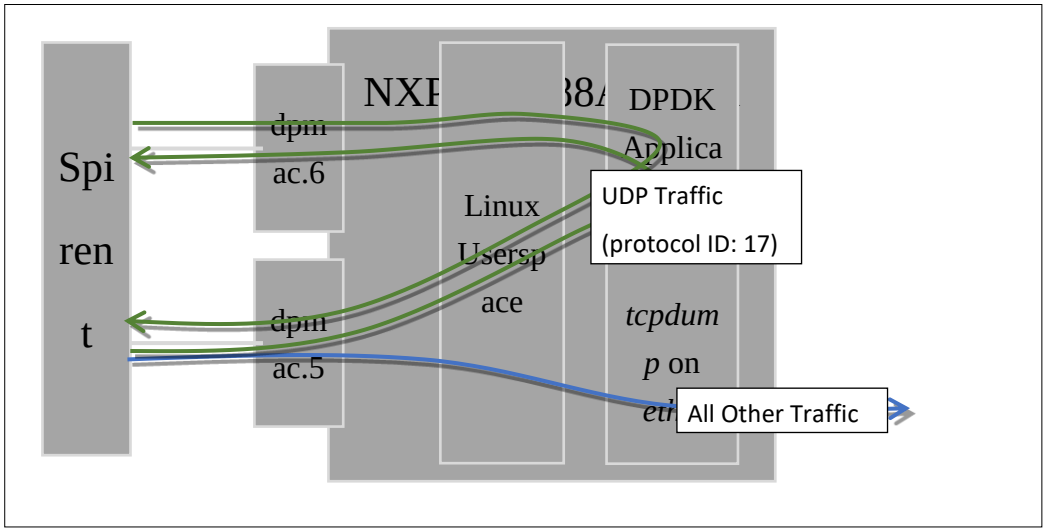
L3FWD: entering main loop on lcore 4

L3FWD: -- lcoreid=4 portid=0 rxqueueid=0

3. Send following packet streams from the Packet generator (in this case, Spirent)
    a. Packets sent to dpmac.5
        i. UDP Traffic: IPv4 Packet with Protocol ID field (next protocol) = 0x11 (hex) or 17 (decimal); Size greater than 82 bytes.
        ii. IPv4 Only Traffic: IPv4 Traffic with any random Protocol ID (next protocol) = 253 (Experimental); Size greater than equal to 64 bytes. Src IP: 1.1.1.1; Dst IP: 2.1.1.1 (so that packets can be forwarded by l3fwd application from dpmac.5 to dpmac.6.
    b. Packets sent to dpmac.6
        i. IPv4 Only Traffic: IPv4 Traffic with any random Protocol ID (next protocol) = 253 (Experimental); Size greater than equal to 64 bytes. Src IP: 2.1.1.1; Dst IP: 1.1.1.1 (so that packets can be forwarded by l3fwd application from dpmac.6 to dpmac.5.

Expected Results

Expected output on Linux Userspace and Packet Generator



1. All traffic with UDP Protocol set in IPv4 header would be sent to Linux Kernel network stack and would be eventually available on the ethernet interface (backed by *dpni.3*). Application like tcpdump would be able to demonstrate the packets coming in:

   root@localhost:~# ifconfig

   …

   eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

   inet 10.0.0.10 netmask 255.255.255.0 broadcast 10.0.0.255

   inet6 fe80::5885:a5ff:fe1c:76af prefixlen 64 scopeid 0x20<link>

   ether 5a:85:a5:1c:76:af txqueuelen 1000 (Ethernet)

   RX packets 0 bytes 0 (0.0 B)

   RX errors 0 dropped 0 overruns 0 frame 0

   TX packets 5 bytes 426 (426.0 B)

   TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

   …

   tcpdump: verbose output suppressed, use -v or -vv for full protocol decode

   listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes

   22:39:10.286502 IP 10.0.0.11 > Ubuntu.ls2088ardb: ip-proto-200 90

   22:39:11.286385 IP 10.0.0.11 > Ubuntu.ls2088ardb: ip-proto-201 90

   22:39:12.286286 IP 10.0.0.11 > Ubuntu.ls2088ardb: ip-proto-202 90

   22:39:13.286172 IP 10.0.0.11 > Ubuntu.ls2088ardb: ip-proto-203 90

22:39:14.286075 IP 10.0.0.11 > Ubuntu.ls2088ardb: ip-proto-204 90

22:39:15.285958 IP 10.0.0.11 > Ubuntu.ls2088ardb: ip-proto-205 90

22:39:16.285845 IP 10.0.0.11 > Ubuntu.ls2088ardb: ip-proto-206 90

22:39:17.285757 IP 10.0.0.11 > Ubuntu.ls2088ardb: ip-proto-207 90

22:39:18.285636 IP 10.0.0.11 > Ubuntu.ls2088ardb: ip-proto-208 90

22:39:19.285541 IP 10.0.0.11 > Ubuntu.ls2088ardb: ip-proto-209 90

^C

10 packets captured

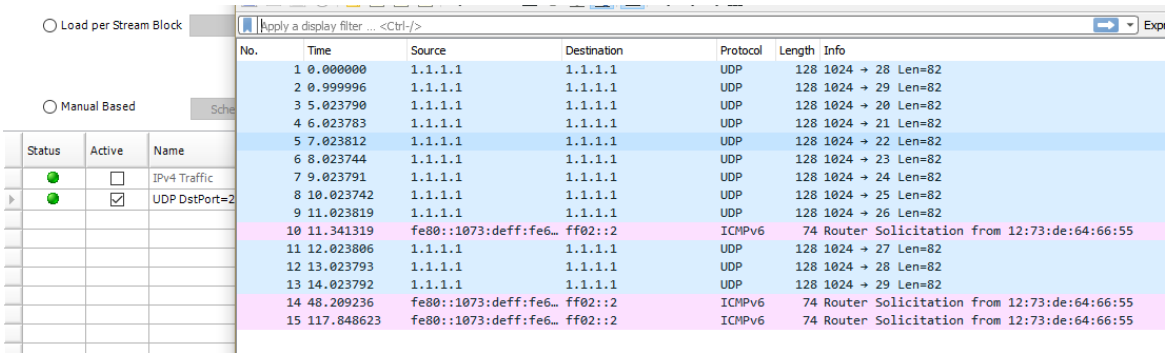
10 packets received by filter

0 packets dropped by kernel

root@Ubuntu:~#

In the above output, it can be observed that packets of different IPv4 Protocol fields are being received in Linux. (This setting can be configured in Spirent). Ubuntu.ls2088ardb refers to the local machine IP 10.0.0.10 which was configured using ifconfig.

2. All other traffic would be visible in the packet generator being reflected by 'l3fwd' application. Below is the screen-grab of Wireshark output of packet captured by Spirent which were reflected by the l3fwd application:



Known Issues

1. Static configuration of DPDK application for forwarding only traffic with Protocol ID 17 (UDP) to DPDK and all other traffic to Linux Kernel interface. For custom flows (different protocols or ports), changes in DPDK application (and verification) would be required. Changes are expected in sample application as well as DPDK driver for DPDMUX.