**Config details and Code changes**

As the project for primarily concerned with packet processing and performance measurement using existing tools, it did not require extensive coding, but involved quite a lot of understanding of how Intel DPDK works and involved a lot of configuration related work to get the environment and topology up and running. The VM setup was quite complicated as we had to find the correct patches and apply them during installation. We mention them in this document.

1. Following is the “pktgen-conf-1-1” file in “pktgen/examples/” folder. Here we mention the port to be used on compute19 (eth1 here), mention the destination MAC and IP address of the incoming port bounded to DPDK at compute28. The parameters which can be changed are marked in bold.

#! /bin/sh

#modprobe pktgen

function pgset() {

local result

echo $1 > $PGDEV

result=`cat $PGDEV | fgrep "Result: OK:"`

if [ "$result" = "" ]; then

cat $PGDEV | fgrep Result:

fi

}

function pg() {

echo inject > $PGDEV

cat $PGDEV

}

# Config Start Here -----------------------------------------------------------

# thread config

# Each CPU has own thread. Two CPU example. We add eth1, eth2 respectivly.

**PGDEV=/proc/net/pktgen/kpktgend\_0**

**echo "Removing all devices"**

**pgset "rem\_device\_all"**

**echo "Adding eth1"**

**pgset "add\_device eth1"**

**echo "Setting max\_before\_softirq 10000"**

**pgset "max\_before\_softirq 10000"**

# device config

# ipg is inter packet gap. 0 means maximum speed.

CLONE\_SKB="clone\_skb 1000000"

# NIC adds 4 bytes CRC

**PKT\_SIZE="pkt\_size 64"**

# COUNT 0 means forever

**#COUNT="count 0"**

**COUNT="count 10000000"**

**IPG="ipg 0"**

**PGDEV=/proc/net/pktgen/eth1**

echo "Configuring $PGDEV"

pgset "$COUNT"

pgset "$CLONE\_SKB"

pgset "$PKT\_SIZE"

pgset "$IPG"

**pgset "dst 10.10.11.2"**

**pgset "dst\_mac 00:04:23:08:91:dc"**

# Time to run

PGDEV=/proc/net/pktgen/pgctrl

echo "Running... ctrl^C to stop"

pgset "start"

echo "Done"

# Result can be vieved in /proc/net/pktgen/eth1

1. **L2Forwarder main.c**:

Can be found on compute28, in the “**dpdk/examples/l2fwd/main.c**”

Again this is something which existed as an sample application and we had to edit some parameters in order to measure performance.

**static void**

**l2fwd\_simple\_forward(struct rte\_mbuf \*m, unsigned portid)**

**{**

**struct ether\_hdr \*eth;**

**void \*tmp;**

**unsigned dst\_port;**

**dst\_port = l2fwd\_dst\_ports[portid];**

**eth = rte\_pktmbuf\_mtod(m, struct ether\_hdr \*);**

**/\* 02:00:00:00:00:xx \*/**

**tmp = &eth->d\_addr.addr\_bytes[0];**

**\*((uint64\_t \*)tmp) = 0x000000000002 + ((uint64\_t)dst\_port << 40);**

**/\*WE HAD TO CHANGE THIS MAC ADDRESS TO MAC ADDRESS OF THE NIC AT compute20 AT THE DESTINATION\*/**

**/\* src addr \*/**

**ether\_addr\_copy(&l2fwd\_ports\_eth\_addr[dst\_port], &eth->s\_addr);**

**l2fwd\_send\_packet(m, (uint8\_t) dst\_port);**

**}**

**Necessary Patches to be installed**

1. Update mk/target/generic/rte.vars.mk:

**- CFLAGS += $(TARGET\_CFLAGS)**

**+CFLAGS += $(TARGET\_CFLAGS) –fPIC**

1. Compilation Errors with /lib/librte\_eal/linuxapp/eal/eal\_ivshmem.c.

(Source: https://github.com/01org/dpdk-ovs/issues/30)

**@@ -472,7 +472,7 @@**

rte\_snprintf(path, sizeof(path), IVSHMEM\_CONFIG\_PATH,

internal\_config.hugefile\_prefix);

- fd = open(path, O\_CREAT | O\_RDWR);

+ fd = open(path, O\_CREAT | O\_RDWR, S\_IRWXU | S\_IRWXG | S\_IRWXO );

if (fd < 0) {

RTE\_LOG(ERR, EAL, "Could not open %s: %s\n", path, strerror(errno));

**@@ -486,7 +486,8 @@**

return -1;

}

- ftruncate(fd, sizeof(struct ivshmem\_shared\_config));

+ if (ftruncate(fd, sizeof(struct ivshmem\_shared\_config)))

+ return -1;

ivshmem\_config = mmap(NULL, sizeof(struct ivshmem\_shared\_config),

PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

Apart from the above patches, there are some errors for which patches are not yet available and will be released in Intel DPDK 1.10.