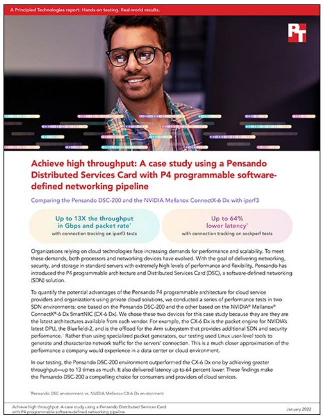

The science behind the report:

Achieve high throughput: A case study using a Pensando Distributed Services Card with P4 programmable software-defined networking pipeline

This document describes what we tested, how we tested, and what we found. To learn how these facts translate into real-world benefits, read the report Achieve high throughput: A case study using a Pensando Distributed Services Card with P4 programmable software-defined networking pipeline.

We concluded our hands-on testing on November 3, 2021. During testing, we determined the appropriate hardware and software configurations and applied updates as they became available. The results in this report reflect configurations that we finalized on November 1, 2021 or earlier. Unavoidably, these configurations may not represent the latest versions available when this report appears.

Our results

To learn more about how we have calculated the wins in this report, go to http://facts.pt/calculating-and-highlighting-wins. Unless we state otherwise, we have followed the rules and principles we outline in that document.

Please note that the data we present in this report represent the outputs from the scenarios we tested, and do not reflect the maximums either vendor has advertised.

SDN pipeline: 1 instance

Throughput

Table 2: SDN pipeline throughput in Gbps for the two test environments with 1 iPerf3 instance. Higher is better.

Packet size (bytes)	Pensando DSC-200 with connection tracking	NVIDIA® Mellanox® ConnectX®-6 Dx with connection tracking	NVIDIA Mellanox ConnectX-6 Dx without connection tracking
96	2.830	0.867	2.999
128	2.671	1.107	3.077
256	5.186	1.884	5.659
512	10.225	3.371	10.888
1,024	18.817	5.925	22.352
1,500	24.845	7.640	23.702
2,048	29.564	9.081	25.567
4,096	20.303	11.380	20.102
8,192	28.679	16.060	29.740
9,000	31.648	17.092	31.818

Packet rate

Table 3: Number of SDN pipeline packets per second for the two test environments with 1 iPerf3 instance. Higher is better.

Packet size (bytes)	Pensando DSC-200 with connection tracking	NVIDIA Mellanox ConnectX-6 Dx with connection tracking	NVIDIA Mellanox ConnectX-6 Dx without connection tracking
96	3,215,094	984,933	3,407,530
128	2,350,617	974,334	2,708,500
256	2,400,752	872,284	2,619,938
512	2,429,613	801,234	2,587,265
1,024	2,265,800	714,490	2,692,075
1,500	2,051,948	632,022	1,957,915
2,048	1,793,370	551,961	1,551,908
4,096	618,728	347,592	612,227
8,192	437,518	246,358	454,164
9,000	440,385	238,757	445,117

SDN pipeline: 4 instances

Throughput

Table 4: SDN pipeline throughput in Gbps for the two test environments with 4 iPerf3 instances. Higher is better.

Packet size (bytes)	Pensando DSC-200 with connection tracking	NVIDIA Mellanox ConnectX-6 Dx with connection tracking	NVIDIA Mellanox ConnectX-6 Dx without connection tracking
96	8.319	1.048	8.462
128	8.011	1.399	7.915
256	17.968	2.201	16.688
512	24.634	4.012	25.756
1,024	55.297	7.379	50.623
1,500	70.417	10.674	71.136
2,048	69.853	13.970	62.542
4,096	70.433	22.055	71.268
8,192	90.618	35.797	77.503
9,000	93.844	85.148	85.148

Packet rate

Table 5: Number of SDN pipeline packets per second for the two test environments with 4 iPerf3 instances. Higher is better.

Packet size (bytes)	Pensando DSC-200 with connection tracking	NVIDIA Mellanox ConnectX-6 Dx with connection tracking	NVIDIA Mellanox ConnectX-6 Dx without connection tracking
96	9,452,506	1,190,672	9,615,248
128	7,051,272	1,230,970	6,966,269
256	8,318,613	1,018,937	7,726,578
512	5,853,860	953,158	6,120,280
1,024	6,660,481	888,798	6,096,560
1,500	5,815,278	881,431	5,873,720
2,048	4,235,427	847,397	3,792,773
4,096	2,149,840	671,439	2,174,320
8,192	1,384,094	546,516	1,181,851
9,000	1,306,323	460,902	1,181,068

SDN pipeline: 16 instances

Throughput

Table 6: SDN pipeline throughput in Gbps for the two test environments with 16 iPerf3 instances. Higher is better.

Packet size (bytes)	Pensando DSC-200 with connection tracking	NVIDIA Mellanox ConnectX-6 Dx with connection tracking	NVIDIA Mellanox ConnectX-6 Dx without connection tracking
96	12.874	0.939	12.873
128	14.798	1.271	15.295
256	26.370	2.015	26.184
512	34.972	3.545	34.720
1,024	48.858	6.912	45.263
1,500	63.908	10.091	57.991
2,048	58.559	12.293	54.270
4,096	78.319	18.065	63.689
8,192	98.519	31.568	67.402
9,000	99.195	32.828	70.728

Packet rate

Table 7: Number of SDN pipeline packets per second for the two test environments with 16 iPerf3 instances. Higher is better.

Packet size (bytes)	Pensando DSC-200 with connection tracking	NVIDIA Mellanox ConnectX-6 Dx with connection tracking	NVIDIA Mellanox ConnectX-6 Dx without connection tracking
96	14,629,822	1,067,310	14,625,850
128	13,027,225	1,118,473	13,462,176
256	12,208,025	932,621	12,120,625
512	8,310,758	842,186	8,248,679
1,024	5,883,882	832,327	5,449,136
1,500	5,276,400	832,959	4,786,584
2,048	3,549,458	745,019	3,288,983
4,096	2,381,769	549,242	1,936,859
8,192	1,500,335	480,800	1,026,533
9,000	1,375,418	455,115	980,629

SDN pipeline: 32 instances

Throughput

Table 8: SDN pipeline throughput in Gbps for the two test environments with 32 iPerf3 instances. Higher is better.

Packet size (bytes)	Pensando DSC-200 with connection tracking	NVIDIA Mellanox ConnectX-6 Dx with connection tracking	NVIDIA Mellanox ConnectX-6 Dx without connection tracking
96	7.726	0.863	4.818
128	9.472	1.091	9.966
256	17.388	1.645	16.321
512	17.778	3.260	13.432
1,024	19.640	6.467	15.195
1,500	25.671	8.822	21.426
2,048	20.889	11.155	17.745
4,096	32.661	16.804	23.806
8,192	39.935	28.902	29.302
9,000	41.489	31.115	34.833

Packet rate

Table 9: Number of SDN pipeline packets per second for the two test environments with 32 iPerf3 instances. Higher is better.

Packet size (bytes)	Pensando DSC-200 with connection tracking	NVIDIA Mellanox ConnectX-6 Dx with connection tracking	NVIDIA Mellanox ConnectX-6 Dx without connection tracking
96	8,777,273	980,077	5,474,638
128	8,336,536	959,989	8,772,182
256	8,047,768	761,666	7,554,694
512	4,223,120	774,558	3,191,855
1,024	2,364,525	778,585	1,829,544
1,500	2,118,733	728,212	1,768,553
2,048	1,265,909	676,134	1,075,598
4,096	993,024	510,954	723,891
8,192	608,215	440,140	446,332
9,000	575,167	431,362	482,962

Latency

Table 10: SDN pipeline one-way latency for the two test environments with a single sockperf instance. Lower is better.

Packet size (bytes)	Pensando DSC-200 with connection tracking	NVIDIA Mellanox ConnectX-6 Dx with connection tracking	NVIDIA Mellanox ConnectX-6 Dx without connection tracking
96	9.669	21.742	10.758
128	9.731	21.525	11.352
256	10.083	21.782	11.194
512	10.724	22.028	11.305
1,024	10.829	22.533	11.372
1,500	11.018	23.794	12.220
2,048	11.392	23.745	12.759
4,096	12.403	25.286	13.720
8,192	14.786	27.225	15.557
9,000	15.472	43.206	15.906

System configuration information

Table 11: Detailed information on the system we tested.

System configuration information	
Tested by	Principled Technologies
Workload & version	• iPerf3 3.7-3 • sockperf 3.7
Workload-specific parameters	See How we tested
Iterations and result choice	See How we tested
Server platform	HPE ProLiant DL385 Gen10 Plus
BIOS name and version	A42 v2.42 (04/29/2021)
Operating system name and version/build number	Ubuntu® 18.04
Date of last OS updates/patches applied	07/02/2021
Processor	
Number of processors	2
Vendor and model	AMD EPYC™ 7302
Core count (per processor)	16
Core frequency (GHz)	3.0
Stepping	0
Hyper-threading	No
Turbo	Yes
Memory module(s)	
Total memory in system (GB)	256
NVMe™ memory present?	No
Total memory (DDR+NVMe RAM)	256
Network adapter	
Vendor and model	1x NVIDIA MCX623106AN-CDAT ConnectX-6 Dx EN Adapter Card
Number and type of ports	2x 100Gbs
Firmware version	22.31.1014
Location	PCIe® slot 5/Gen4 x16
Network adapter	
Vendor and model	1x Broadcom BCM957416N4160C
Number and type of ports	2x 1Gbps
Firmware version	218.0.166.0

How we tested

Preparing the servers

The OS and software on the two servers were identical: Ubuntu 18.04 with patches up to November 1, 2016. We installed Open vSwitch 2.9.8 from the Ubuntu repository. We installed Mellanox firmware and kernel modules from the latest Mellanox distribution (firmware 16.3.1014, and Ubuntu modules 4.15.0-159). The automatic Mellanox helper scripts initialized Mellanox devices and presented them to the kernel.

We used Open vSwitch's (OVS) capabilities in several ways to create part of the network configuration for testing of both devices. First, we created the VXLAN tunnel with OVS with the four scripts ovs_*XXX*_*SYYYY*.sh. Second, we created OVS flows on server 1 (the source of the application data) that directed packets with MAC addresses corresponding to the correct application endpoints. Otherwise, the OVS defaults to periodically flooding its ports with ARP requests, which unnecessarily reduces performance. For details on these flows, see the two scripts flw_*XXX*_source.sh.

Both devices performed the following SDN manipulations for traffic through the VXLAN tunnel:

- matching the VXLAN network ID
- matching the IP subnets of the inner packets
- matching the inner destination IP and rewriting the inner destination MAC address
- tracking TCP connections statistics

The two devices accomplish this work differently. The DSC-200G performs them on device with its SDN pipeline. For the CX-6 Dx testing, we used OVS flows on server 2 (the application sink/reflector). For details, see the script flw_cx6_con_sink.sh for the tests with connection tracking, and the script flw_cx6_non_sink.sh for the tests without connection tracking.

1. Install iPerf3 version 3.7.3 from the package at https://iperf.fr/iperf-download.php#ubuntu.
2. Compile sockperf version 3.7 from source code obtained from its GitHub repository:

```
wget https://github.com/Mellanox/sockperf/archive/refs/tags/3.7.tar.gz
tar xf 3.7.ta.gz
cd sockperf-3.7
./configure --prefix=/usr/local
make -j
make check
sudo make install
```

3. Add three kernel configuration settings to the file /etc/sysctl.d/19-sysctl.conf:

```
net.ipv4.tcp_timestamps=0
net.ipv4.tcp_sack = 0
net.ipv4.tcp_low_latency = 1
```

4. Apply these settings:

```
sudo sysctl -p
```

5. Assign tuned profile:

```
sudo apt install tuned
sudo tuned-adm profile network-latency
sudo systemctl restart tuned
sudo systemctl stop tuned
sudo systemctl disable tuned
```

6. Reset interrupts for the receive and transmit queues for the NICs used in the testing:

```
sudo systemctl stop irqbalance; sudo systemctl disable irqbalance
wget https://raw.githubusercontent.com/Mellanox/mlnx-tools/master/ofed_scripts/{common,set}_
irq_affinity.sh
chmod a+rx *.sh
sudo ./set_irq_affinity.sh ens5f1 # change en5f1 to the physical nic on the server
```

Configuring software-defined networking (SDN) for testing the Pensando DSC-200

1. On the source server, initialize and configure Open vSwitch, configure the network interfaces, and apply the SDN flow configuration:

```
sudo ./ovs_dsc_source.sh
sudo ./net_dsc_source.sh
sudo ./flw_dsc_source.sh
```

You can find the contents for these scripts below: ovs_dsc_source.sh | net_dsc_source.sh | flw_dsc_source.sh

2. On the sink server, initialize and configure Open vSwitch, and configure the network interfaces:

```
sudo ./ovs_dsc_sink.sh
sudo ./net_dsc_sink.sh
```

You can find the contents for these scripts below: ovs_dsc_sink.sh | net_dsc_sink.sh

Scripts for configuring the Pensando DSC-200 environment

ovs_dsc_sink.sh

```
#!/bin/bash
ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
ovs-vsctl --if-exists del-br sriov-ovs1
ovs-vsctl --if-exists del-br sriov-ovs2
ovs-vsctl add-br sriov-ovs1
ovs-vsctl add-port sriov-ovs1 ens2f1_0
ovs-vsctl add-port sriov-ovs1 vxlan1 -- set interface vxlan1 type=vxlan \
        options:local_ip=1.0.0.4 options:remote_ip=1.0.0.2 options:key=1 options:dst_port=4789
```

net_dsc_source.sh

```
#!/bin/bash
PHYS0="ens5f0"
REPR0="ens5f2"
PCIE0="0000:a3:00.2"
MAC0="de:ad:33:44:55:10"
IP_APP0="193.0.0.1"
IP_VTE0="1.0.0.3"

echo 1 >/sys/class/net/${PHYS0}/device/sriov_numvfs
ip l set dev ${PHYS0} vf 0 mac "$MAC0"

echo ${PCIE0}  >/sys/bus/pci/drivers/mlx5_core/unbind
echo switchdev >/sys/class/net/${PHYS0}/compat/devlink/mode

systemctl restart openvswitch-switch
ip l set dev ${PHYS0}_0 up

echo ${PCIE0} >/sys/bus/pci/drivers/mlx5_core/bind

ip a flush             dev ${PHYS0} scope global
ip a add ${IP_VTE0}/24 dev ${PHYS0}
ip a add ${IP_APP0}/24 dev ${REPR0}

ip l set dev ${PHYS0}   up
ip l set dev ${PHYS0}_0 up
ip l set dev ${REPR0}   up

ip l set dev ${PHYS0}   mtu 9216
ip l set dev ${PHYS0}_0 mtu 9216
ip l set dev ${REPR0}   mtu 9000

# hard coded for DSC
ip link set dev ${PHYS0} address 00:00:bb:bb:bb:b0
ip link set dev ${REPR0} address 00:11:11:11:00:01
# add an error for remote destination
arp -s 193.0.0.9 00:22:22:22:11:11
arp -s 1.0.0.2   00:ae:cd:01:d3:86
```

flw_dsc_source.sh

```
#!/bin/bash
ovs-ofctl add-flow sriov-ovs1 priority=100,in_port=1,dl_dst:00:22:22:22:11:11,actions=2
ovs-ofctl add-flow sriov-ovs1 priority=100,in_port=2,dl_dst:00:11:11:11:00:01,actions=1
```

ovs_dsc_source.sh

```
#!/bin/bash
ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
ovs-vsctl --if-exists del-br sriov-ovs1
ovs-vsctl --if-exists del-br sriov-ovs2
ovs-vsctl add-br sriov-ovs1
ovs-vsctl add-port sriov-ovs1 ens5f0_0
ovs-vsctl add-port sriov-ovs1 vxlan1 -- set interface vxlan1 type=vxlan \
        options:local_ip=1.0.0.3 options:remote_ip=1.0.0.2 options:key=1 options:dst_port=4789
```

net_dsc_sink.sh

```
#!/bin/bash
PHYS0="ens2f1"
REPR0="enp39s1f2"
PCIE0="0000:27:01.2"
MAC0="de:ad:33:44:55:20"
IP_APP0="193.0.0.9"
IP_VTE0="1.0.0.4"

echo 1 >/sys/class/net/${PHYS0}/device/sriov_numvfs
ip l set dev ${PHYS0} vf 0 mac "$MAC0"

echo ${PCIE0}  >/sys/bus/pci/drivers/mlx5_core/unbind
echo switchdev >/sys/class/net/${PHYS0}/compat/devlink/mode

systemctl restart openvswitch-switch
ip l set dev ${PHYS0}_0 up

echo ${PCIE0} >/sys/bus/pci/drivers/mlx5_core/bind

ip a flush             dev ${PHYS0} scope global
ip a add ${IP_VTE0}/24 dev ${PHYS0}
ip a add ${IP_APP0}/24 dev ${REPR0}

ip l set dev ${PHYS0}   up
ip l set dev ${PHYS0}_0 up
ip l set dev ${REPR0}   up

ip l set dev ${PHYS0}   mtu 9216
ip l set dev ${PHYS0}_0 mtu 9216
ip l set dev ${REPR0}   mtu 9000

# hard coded for DSC
ip link set dev ${PHYS0} address 00:00:bb:bb:bb:b1
ip link set dev ${REPR0} address 00:22:22:22:00:01
arp -s 193.0.0.1 00:11:11:11:00:01
arp -s 1.0.0.2   00:ae:cd:01:d3:86
```

Configuring SDN for testing the NVIDIA ConnectX-6 Dx

We set the SDN configuration depending on whether we were tracking connections and their statistics. For all cases, initialize the connection as follows.

1. On the source server, initialize and configure Open vSwitch, configure the network interfaces, and apply the SDN flow configuration:

```
sudo ./ovs_cx6_source.sh
sudo ./net_cx6_source.sh
sudo ./flw_cx6_source.sh
```

You can find the contents for these scripts below: ovs_cx6_source.sh | net_cx6_source.sh | flw_cx6_source.sh

2. On the sink server, initialize and configure Open vSwitch, and configure the network interfaces:

```
sudo ./ovs_cx6_sink.sh
sudo ./net_cx6_sink.sh
```

You can find the contents for these scripts below: ovs_cx6_sink.sh | net_cx6_sink.sh

Adding flows for testing with connection tracking

If the SDN configuration will track connections, add the following flow.

1. On the sink server, apply the SDN flow configuration:

```
sudo ./flw_cx6_con_sink.sh
```

You can find the contents for this script below: flw_cx6_con_sink.sh

Adding flows for testing without connection tracking

If the SDN configuration will not track connections, add the following flow.

1. On the sink server, apply the SDN flow configuration:

```
sudo ./flw_cx6_non_sink.sh
```

You can find the contents for this script below: flw_cx6_non_sink.sh

Scripts for configuring the NVIDIA CX-6 Dx environment

ovs_cx6_source.sh

```
#!/bin/bash
ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
ovs-vsctl --if-exists del-br sriov-ovs1
ovs-vsctl --if-exists del-br sriov-ovs2
ovs-vsctl add-br sriov-ovs2
ovs-vsctl add-port sriov-ovs2 ens5f0_0
ovs-vsctl add-port sriov-ovs2 vxlan2 -- set interface vxlan2 type=vxlan \
        options:local_ip=10.0.0.1 options:remote_ip=10.0.0.2 options:key=2 options:dst_port=4789
```

net_cx6_source.sh

```
#!/bin/bash
PHYS0="ens5f1"
REPR0="enp163s1f2"
PCIE0="0000:a3:01.2"
MAC0="de:ad:33:44:55:11"
IP_APP0="192.168.2.1"
IP_VTE0="10.0.0.1"

echo 1 >/sys/class/net/${PHYS0}/device/sriov_numvfs
ip l set dev ${PHYS0} vf 0 mac "$MAC0"

echo ${PCIE0}  >/sys/bus/pci/drivers/mlx5_core/unbind
echo switchdev >/sys/class/net/${PHYS0}/compat/devlink/mode

systemctl restart openvswitch-switch
ip l set dev ${PHYS0}_0 up

echo ${PCIE0} >/sys/bus/pci/drivers/mlx5_core/bind

ip a flush             dev ${PHYS0} scope global
ip a add ${IP_VTE0}/24 dev ${PHYS0}
ip a add ${IP_APP0}/24 dev ${REPR0}

ip l set dev ${PHYS0}   up
ip l set dev ${PHYS0}_0 up
ip l set dev ${REPR0}   up

ip l set dev ${PHYS0}   mtu 9216
ip l set dev ${PHYS0}_0 mtu 9216
ip l set dev ${REPR0}   mtu 9000

# modify for remote destination
arp -s 192.168.2.2 de:ad:AA:BB:CC:DD
```

flw_cx6_source.sh

```
#!/bin/bash
ovs-ofctl add-flow sriov-ovs2 priority=100,in_port=1,dl_dst:de:ad:aa:bb:cc:dd,actions=2
ovs-ofctl add-flow sriov-ovs2 priority=100,in_port=2,dl_dst:de:ad:33:44:55:11,actions=1
```

ovs_cx6_sink.sh

```
#!/bin/bash
ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
ovs-vsctl --if-exists del-br sriov-ovs1
ovs-vsctl --if-exists del-br sriov-ovs2
ovs-vsctl add-br sriov-ovs2
ovs-vsctl add-port sriov-ovs2 ens2f0_0
ovs-vsctl add-port sriov-ovs2 vxlan2 -- set interface vxlan2 type=vxlan \
        options:local_ip=10.0.0.2 options:remote_ip=10.0.0.1 options:key=2 options:dst_port=4789
```

net_cx6_sink.sh

```
#!/bin/bash
# sink server version
PHYS0="ens2f0"
REPR0="ens2f2"
PCIE0="0000:27:00.2"
MAC0="de:ad:33:44:55:21"
IP_APP0="192.168.2.2"
IP_VTE0="10.0.0.2"

echo 1 >/sys/class/net/${PHYS0}/device/sriov_numvfs
ip l set dev ${PHYS0} vf 0 mac "$MAC0"

echo ${PCIE0}  >/sys/bus/pci/drivers/mlx5_core/unbind
echo switchdev >/sys/class/net/${PHYS0}/compat/devlink/mode

systemctl restart openvswitch-switch
ip l set dev ${PHYS0}_0 up

echo ${PCIE0} >/sys/bus/pci/drivers/mlx5_core/bind

ip a flush             dev ${PHYS0} scope global
ip a add ${IP_VTE0}/24 dev ${PHYS0}
ip a add ${IP_APP0}/24 dev ${REPR0}

ip l set dev ${PHYS0}   up
ip l set dev ${PHYS0}_0 up
ip l set dev ${REPR0}   up

ip l set dev ${PHYS0}   mtu 9216
ip l set dev ${PHYS0}_0 mtu 9216
ip l set dev ${REPR0}   mtu 9000
```

flw_cx6_con_sink.sh

```
#!/bin/bash
ovs-ofctl del-flows sriov-ovs2

ovs-ofctl add-flow sriov-ovs2 table=0,priority=10,tun_id=2,ip,nw_src=192.168.2.0/24,nw_dst=192.168.2.0
/24,actions=resubmit'(,1)'
ovs-ofctl add-flow sriov-ovs2 table=0,priority=10,ip,nw_src=192.168.2.0/24,nw_dst=192.168.2.0/24,actio
ns=resubmit'(,1)'
ovs-ofctl add-flow sriov-ovs2 table=0,priority=10,arp,actions=NORMAL
ovs-ofctl add-flow sriov-ovs2 table=0,priority=10,icmp,actions=NORMAL

ovs-ofctl add-flow sriov-ovs2 table=1,priority=10,ip,nw_dst=192.168.2.2,actions=mod_dl_dst:de:ad:33:44
:55:21,resubmit'(,2)'
ovs-ofctl add-flow sriov-ovs2 table=1,priority=10,arp,actions=NORMAL
ovs-ofctl add-flow sriov-ovs2 table=1,priority=10,icmp,actions=NORMAL
ovs-ofctl add-flow sriov-ovs2 "table=1,priority=10,ct_state=-trk,ip,tcp,action=ct(table=3)"
ovs-ofctl add-flow sriov-ovs2 "table=1,priority=10,ct_state=-trk,ip,udp,action=ct(table=3)"
ovs-ofctl add-flow sriov-ovs2 table=1,priority=1,actions=drop

ovs-ofctl add-flow sriov-ovs2 table=2,priority=10,arp,actions=NORMAL
ovs-ofctl add-flow sriov-ovs2 table=2,priority=10,icmp,actions=NORMAL
ovs-ofctl add-flow sriov-ovs2 "table=2,priority=10,ct_state=-trk,ip,tcp,action=ct(table=3)"
ovs-ofctl add-flow sriov-ovs2 "table=2,priority=10,ct_state=-trk,ip,udp,action=ct(table=3)"
ovs-ofctl add-flow sriov-ovs2 table=2,priority=1,actions=drop

ovs-ofctl add-flow sriov-ovs2 "table=3,priority=1000,ct_state=+new+trk,tcp,in_port=1,actions=ct(
commit),output:2"
ovs-ofctl add-flow sriov-ovs2 "table=3,priority=1000,ct_state=+new+trk,tcp,in_port=2,actions=ct(
commit),output:1"
ovs-ofctl add-flow sriov-ovs2 "table=3,priority=1000,ct_state=+trk,udp,in_port=1,actions=ct(
commit),output:2"
ovs-ofctl add-flow sriov-ovs2 "table=3,priority=1000,ct_state=+trk,udp,in_port=2,actions=ct(
commit),output:1"
ovs-ofctl add-flow sriov-ovs2 table=3,priority=900,ct_state=+est+trk,tcp,in_port=1,actions=output:2
ovs-ofctl add-flow sriov-ovs2 table=3,priority=900,ct_state=+est+trk,tcp,in_port=2,actions=output:1
```

flw_cx6_non_sink.sh

```
#!/bin/bash
ovs-ofctl del-flows sriov-ovs2

ovs-ofctl add-flow sriov-ovs2 table=0,priority=10,tun_id=2,ip,nw_src=192.168.2.0/24,nw_dst=192.168.2.0
/24,actions=resubmit'(,1)'
ovs-ofctl add-flow sriov-ovs2 table=0,priority=10,ip,nw_src=192.168.2.0/24,nw_dst=192.168.2.0/24,actio
ns=resubmit'(,1)'
ovs-ofctl add-flow sriov-ovs2 table=0,priority=10,arp,actions=NORMAL
ovs-ofctl add-flow sriov-ovs2 table=0,priority=10,icmp,actions=NORMAL

ovs-ofctl add-flow sriov-ovs2 table=1,priority=10,ip,nw_dst=192.168.2.2,actions=mod_dl_
dst:de:ad:33:44:55:21,NORMAL
ovs-ofctl add-flow sriov-ovs2 table=1,priority=1,actions=NORMAL
```

Running the throughput tests with iPerf3

1. Configure the SDN on each server for the device under test (see Configuring software-defined networking (SDN) for testing the Pensando DSC-200 or Configuring SDN for testing the NVIDIA ConnectX-6 Dx).
2. Set the MTU on the VF interface on each server. For example, to set the MTU to 128 bytes, execute the following command using the appropriate network name as specified in the table:

```
ip set dev <DEVICE> mtu 128
```

Server	Device under test	Network device
Source	DSC 200	ens5f2
Source	ConnectX-6 Dx	ens5f2
Sink	DSC 200	enp39s1f2
Sink	ConnectX-6 Dx	ens2f2

3. Start 32 iPerf3 instances on the sink server using the script iperf-parallel-servers.sh. For example,

```
iperf-parallel-servers.sh 5000 32 run1
```

You can find the contents for this script below: iperf-parallel-servers.sh

4. Perform the throughput test by running the script run-iperf-clients.sh on the source server. This script performs the through put test four times for 1, 4, 16, and 32 iPerf3 instances. It collects throughput and packet rates (script: rxtx_counts.sh) as well as recording the SDN flows (script: flow_counts.sh). For example, for the 128 MTU tests,

```
run-iperf-clients.sh 128
```

You can find the contents for these scripts below: run-iperf-clients.sh | rxtx_counts.sh | flow_counts.sh

Scripts for throughput tests

iperf-parallel-servers.sh

```
#!/bin/bash
# Run multiple parallel instances of iperf3 servers
# Adapted from the code at https://sandilands.info/sgordon/doc/code/iperf-parallel-servers
base_port=$1;        shift
num_servers=$1;      shift
report_base=$1;      shift
# Optional command line input: other iperf options
iperf_options=”$*”
iperf3=/usr/bin/iperf3

for (( i=0; i<= num_servers; i++ )); do
  server_port=$((base_port+i));
  report_file=${report_base}-${server_port}.txt
   $iperf3 -s -p $server_port -A $i $iperf_options &> $report_file &
done
```

run-iperf-clients.sh

```
#!/bin/bash
mtu=$1
if [[ -z $mtu ]] ; then
  echo "usage: $0 MTU"
  exit 1
fi

echo "MTU = $mtu"
for i in {1..4} ; do
  for cl in 32 16 4 1; do
    fi="${mtu}_${cl}_$i"
    echo "Run $i, MTU = $mtu, Clients = $cl $fi"
    bash ~/testing/iperf/iperf-parallel-clients.sh 5000 193.0.0.9 ${cl} 30 "xfx_${fi}"
    bash ~/testing/rxtx_counts.sh ens5f2 32 | tee "rxtx_${fi}_s1.txt" &
    bash ~/testing/flow_counts.sh 32 >& "flow_${fi}_s1.txt" &
    ssh 10.30.10.104 bash ~/pt-stuff/rxtx_counts.sh enp39s1f2 32 > "rxtx_${fi}_s2.txt"
    wait
    echo quick sleep
    sleep 30
  done
done
```

iperf-parallel-clients.sh

```
#!/bin/bash
# Run multiple parallel instances of iperf3 clients
# Adapted from the code at https://sandilands.info/sgordon/doc/code/iperf-parallel-clients
base_port=$1;        shift
server_ip=$1;        shift
num_clients=$1;      shift
test_duration=$1;    shift
report_base=$1;      shift
# Optional command line input: other iperf options
iperf_options="$*"
iperf3=/usr/bin/iperf3

for (( i=0; i< num_clients; i++ )); do
  server_port=$((base_port+i));
  # Report file includes server ip, server port and test duration
  report_file=${report_base}-${server_ip}-${server_port}-${test_duration}.txt
  $iperf3 -c $server_ip -p $server_port -t $test_duration -A $i $iperf_options &> $report_file &
done
```

rxtx_counts.sh

```
#!/bin/bash
if [ -z "$1" ]; then
  echo "usage: $0 network-interface <counter>"
  exit
fi
IF=$1
sample_count=$2

#while [ $count -le $sample_count ]
for (( count=1 ; count <= sample_count; count++ )); do
  RP1=$(cat /sys/class/net/$IF/statistics/rx_packets)
  TP1=$(cat /sys/class/net/$IF/statistics/tx_packets)
  RB1=$(cat /sys/class/net/$IF/statistics/rx_bytes)
  TB1=$(cat /sys/class/net/$IF/statistics/tx_bytes)
  sleep 1
  RP2=$(cat /sys/class/net/$IF/statistics/rx_packets)
  TP2=$(cat /sys/class/net/$IF/statistics/tx_packets)
  RB2=$(cat /sys/class/net/$IF/statistics/rx_bytes)
  TB2=$(cat /sys/class/net/$IF/statistics/tx_bytes)
  TXPPS=$(( TP2 - TP1))
  RXPPS=$(( RP2 - RP1))
  TXBPS=$(( TB2 - TB1))
  RXBPS=$(( RB2 - RB1))
  TXBPS=$(( TXBPS * 8))
  RXBPS=$(( RXBPS * 8))
  TGBPS=$(echo "$TXBPS/1000000000" | bc -l )
  RGBPS=$(echo "$RXBPS/1000000000" | bc -l )
  echo "$IF: Tx $TXPPS pkts/s $TGBPS Gbits/s Rx $RXPPS pkts/s $RGBPS Gbits/s"
done
```

flow_counts.sh

```
#!/bin/bash
if [ -z "$1" ]; then
  echo "usage: $0 <counter>"
  exit
fi
sample_count=$1

for (( count=1 ; count <= sample_count; count++ )); do
  printf "%d : flows=%s\n" $count "$(ovs-appctl dpctl/dump-flows type=offloaded)"
  sleep 1

done
```

Running the latency tests with sockperf

1. Configure the SDN on each server for the device under test (see Configuring software-defined networking (SDN) for testing the Pensando DSC-200 or Configuring SDN for testing the NVIDIA ConnectX-6 Dx). Set the MTU on the VF interface on each server. For example, to set the MTU to 128 bytes, execute the following command using the appropriate network name as specified in the table from step 2 above.
2. Start the sockperf listener on the sink server using the script start_sockperf.sh.
3. Perform the latency test by running the script run_sockperf.sh on the source server.

Scripts for latency tests

start_sockperf.sh

```
#!/bin/bash
ip=$1
if [[ -z $ip ]] ; then
  echo "usage: $0 IPADDRESS-under-test"
  exit 1
fi
ulimit -n 1000000
sockperf sr --tcp --daemonize -i $ip
```

run_sockperf.sh

```
#!/bin/bash
ip=$1
if [[ "$ip" = "" ]] ; then
  echo "usage: $0 IPADRESS"
  exit 1
fi

for i in {1..4} ; do
  for pk in 96 128 256 512 1024 1500 2048 4096 8192 9000; do
    fi="${pk}_${i}_${ip}"
    echo "========================="
    echo "Run $i, Packet Size = $pk"
    sockperf pp -i $ip --tcp -m $pk | tee sockp_${fi}.txt
    echo "short wait"
    sleep 10
  done
done
```

Read the report at https://facts.pt/QtrGFqB ▶

This project was commissioned by Pensando.

Principled Technologies®

Facts matter.®