redhat.

Open vSwitch Performance testing

Eelco Chaudron
Senior Software Engineer
December 2018

Why another test suite?

ovs_perf

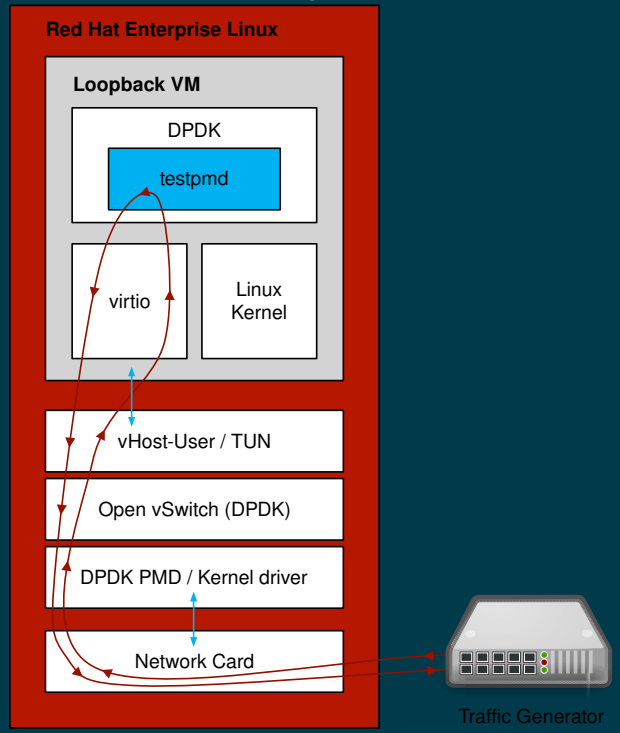
- Initially I needed something to test a single NIC independently
- I needed something that was easy to setup
- Should not mess with my setup
- Support for the kernel, DPDK, and TC Flower datapath
- Include some basic CPU graphs
- Work with the Xena tester I had (don't worry currently it works with TRex also :)

Test topologies

- Physical interface to Virtual interface back to Physical interface (PVP)
- Physical interface to Virtual interface (PV)
- Physical interface to Physical interface (PP)

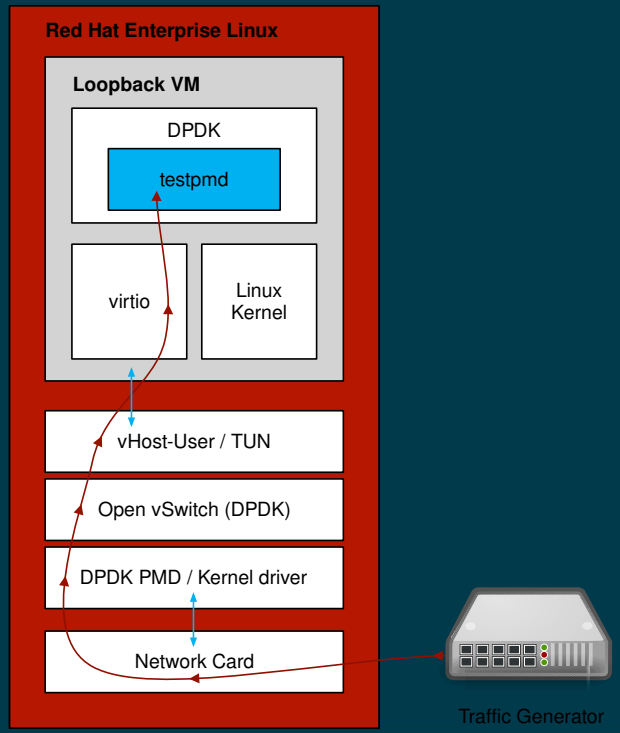
Test topologies

Physical interface to Virtual interface back to Physical interface (PVP)



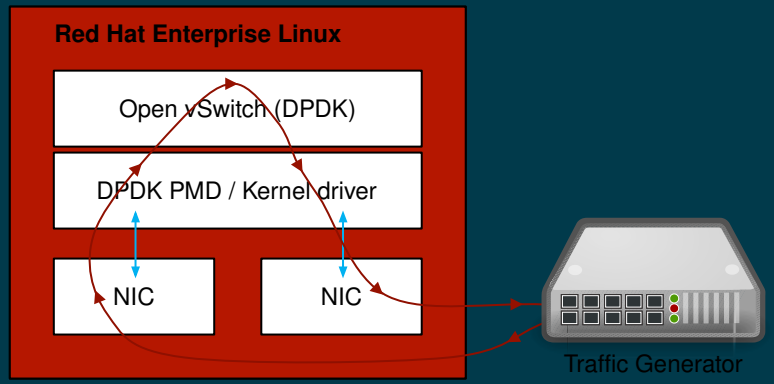
Test topologies

Physical interface to Virtual interface (PV)



Test topologies

Physical interface to Physical interface (PP)



Traffic generation

- All test by default send their traffic at the testers line rate
  - This can be adjusted by setting the *--traffic-rate option* (percentage of line speed)
- There is also a zero-loss PVP test, *--run-pvp-zero-loss-test*
  - Uses binary search to figure out at which transmit rate no packets are lost
  - Steps can be configured with *--zero-loss-step*, which is 1% by default
- You configure the number of streams, packet sizes, and the duration of each test
  - Number of streams: *--stream-list*; by default 10,1000,10000,100000,1000000
  - Packet sizes: *--packet-list*; by default 64,128,256,512,768,1024,1514
  - Runtime: *-r*, *--run-time*; by default this is 20 seconds

Traffic generation

Traffic types

- All packets send are Ethernet/IPv4/UDP
- The following traffic types can be selected, *--flow-type*:
  - L2: the source and destination MAC addresses increase for each packet
  - L3: the source and destination IPv4 addresses increase for each packet
  - L4-UDP: the source and destination UDP ports increase for each packet

NOTE: L4-UDP is NOT yet supported using the Trex

NOTE: The reason for only having two fields changing was due to a limitation in my XENA testers hardware

Open vSwitch configuration

- The script can configure the basic bridge, and add the ports
  - To skip this use the *--no-bridge-config* option, preferred
- The script will configure the required OpenFlow rules
  - This can be changed using the *--flow-rule-type* option:
    - flows (default)
    - NORMAL
    - port
    - none

NOTE: If you have a switch in the middle with learning enabled, you'll need to add the *--mac-swap* option as by default *testpmd* is started without it.

Open vSwitch configuration

--flow-rule-type options explained

--flow-rule-type options:

- **flows**(default): Each traffic flow gets its own OpenFlow rule configured based on the ingress port, in_port. Based on the configured *--flow-type* additional match criteria are programmed. For L2, dl_dst, for L3 the nw_src / nw_dst, for L4-UDP the tp_src / tp_dst. Note that for PVP test this means two rules get programmed. One towards the VM, and one from the VM.
- **NORMAL**: One OpenFlow rules gets programmed matching all traffic, with the NORMAL action, aka L2/FDB learning.
- **port**: One (PP, PV) or two (PVP) rules get programmed redirecting all ingress port traffic to the required egress port.
- **none**: Do not program any rules, and leave the ones installed in place.

How to run it?

- You do not need to run the script from the DUT itself
  - Its advised to run it on, or close to the tester
- The script will ssh to the DUT, and from there jump to the VM if needed
- Script will start/stop *testpmd* on the VM
  - Make sure the DUT has access to the VM
  - Make sure *testpmd* can be started on the VM
- For Open vSwitch configuration/inspection the ovs-..ctl commands are used, not the Python interface
- Debugging can be enabled to see all commands/responses executed by the script (*-d, --debug* in combination with *-l, --logging)*

How to run it?

Command line example

```
# ~/ovs_perf/ovs_performance.py \
  -d -l testrun_log.txt \                 # Enable script debugging, and save the output to testrun_log.txt
  --tester-type trex \                    # Set tester type to TRex
  --tester-address localhost \            # IP address of the TRex server
  --tester-interface 0 \                  # Interface number used on the TRex
  --ovs-address 10.19.17.133 \            # DUT IP address
  --ovs-user root \                       # DUT login user name
  --ovs-password root \                   # DUT login user password
  --dut-vm-address 192.168.122.5 \        # Address on which the VM is reachable
  --dut-vm-user root \                    # VM login user name
  --dut-vm-password root \                # VM login user password
  --dut-vm-nic-queues=2 \                 # Number of rx/tx queues to use on the VM (testpmd --rxq --txq)
  --physical-interface dpdk0 \            # OVS Physical interface, i.e. connected to TRex
  --physical-speed=10 \                   # Speed of the physical interface, for DPDK we can not detect it
  --virtual-interface vhost0 \            # OVS Virtual interface, i.e. connected to the VM
  --dut-vm-nic-pci=0000:00:02.0 \         # PCI address of the interface in the VM (testpmd -w)
  --packet-list=64 \                      # Comma separated list of packets to test with
  --stream-list=1000 \                    # Comma separated list of number of flows/streams to test with
  --no-bridge-config \                    # Do not configure the OVS bridge, assume it's already done
  --skip-pv-test                          # Skip the Physical to Virtual test
```

See *./ovs_performance.py --help* for additional options.

Results

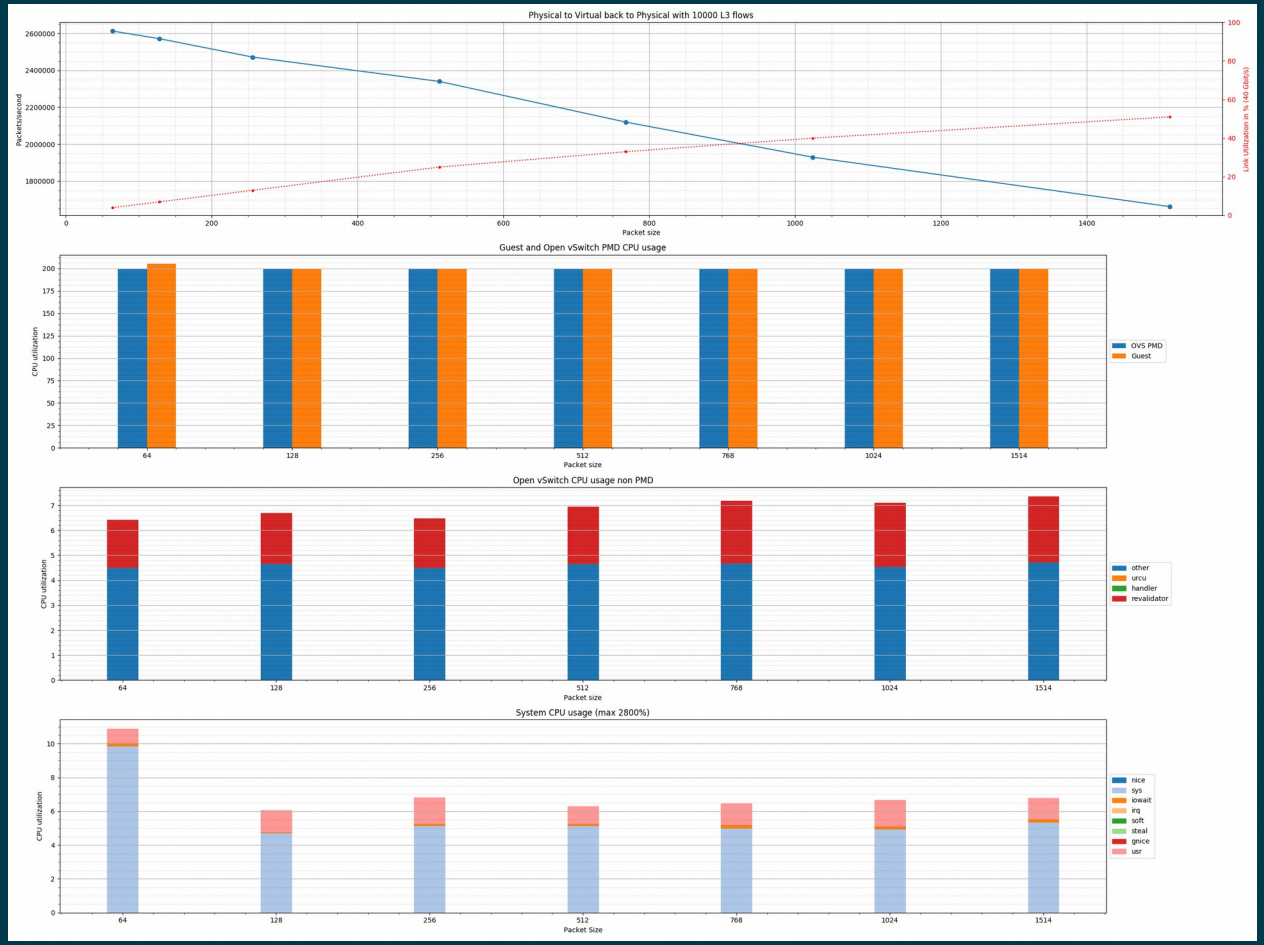
- Results are in a CSV file, including some oddly formatted CPU statistics:

```
$ grep -v cpu test_results_l3.csv
"Physical port, ""dpdk0"", speed 10 Gbit/s, traffic rate 100%"

"Physical to Virtual to Physical test, L3 flows"
,Packet size
Number of flows,64,100,128,200,256,300,400,500,512,600,700,800,900,1000,1024,1100,1200,1300,1400,1500,1514

10,4682768.6875,4258093.4875,4284439.48125,3711447.0375000006,3770709.4250000003,3086844.75625,...
1000,1525347.8750000002,1502883.7499999998,1499924.7999999998,1471797.05,1461272.05625,1418358,...
10000,1460243.8250000002,1419266.4312500001,1412304.9249999998,1398823.84375,1377321.8187499999,..
```

- In addition, several graphs are generated:
  - Two per stream size, one with and one without showing the theoretical maximum
  - Two summarizing all stream sizes, again with and without showing the theoretical maximum

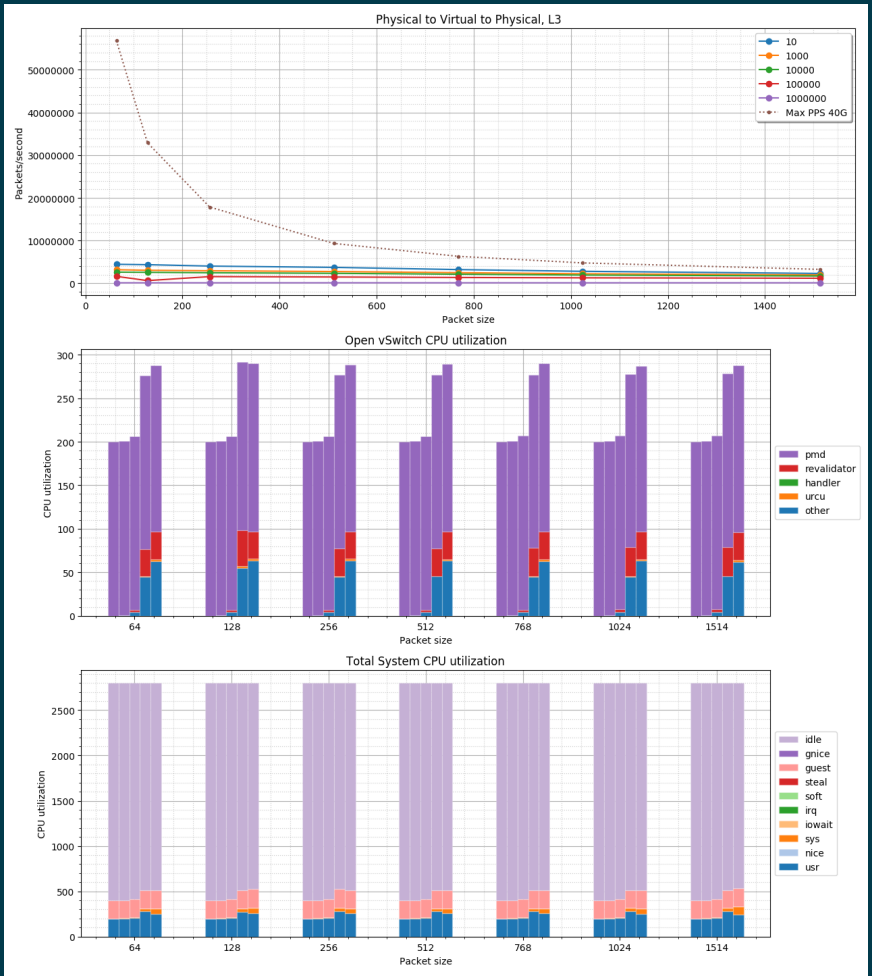
Results

Graphs



Results

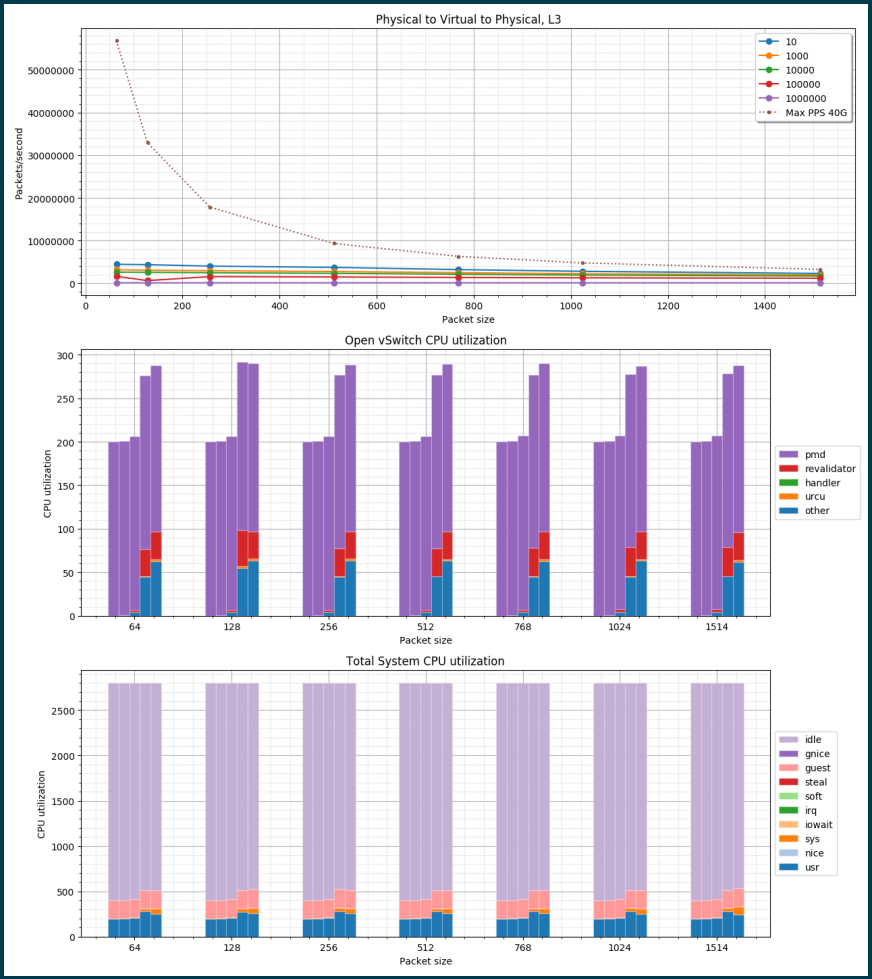
Graphs



redhat.

Results

Graphs

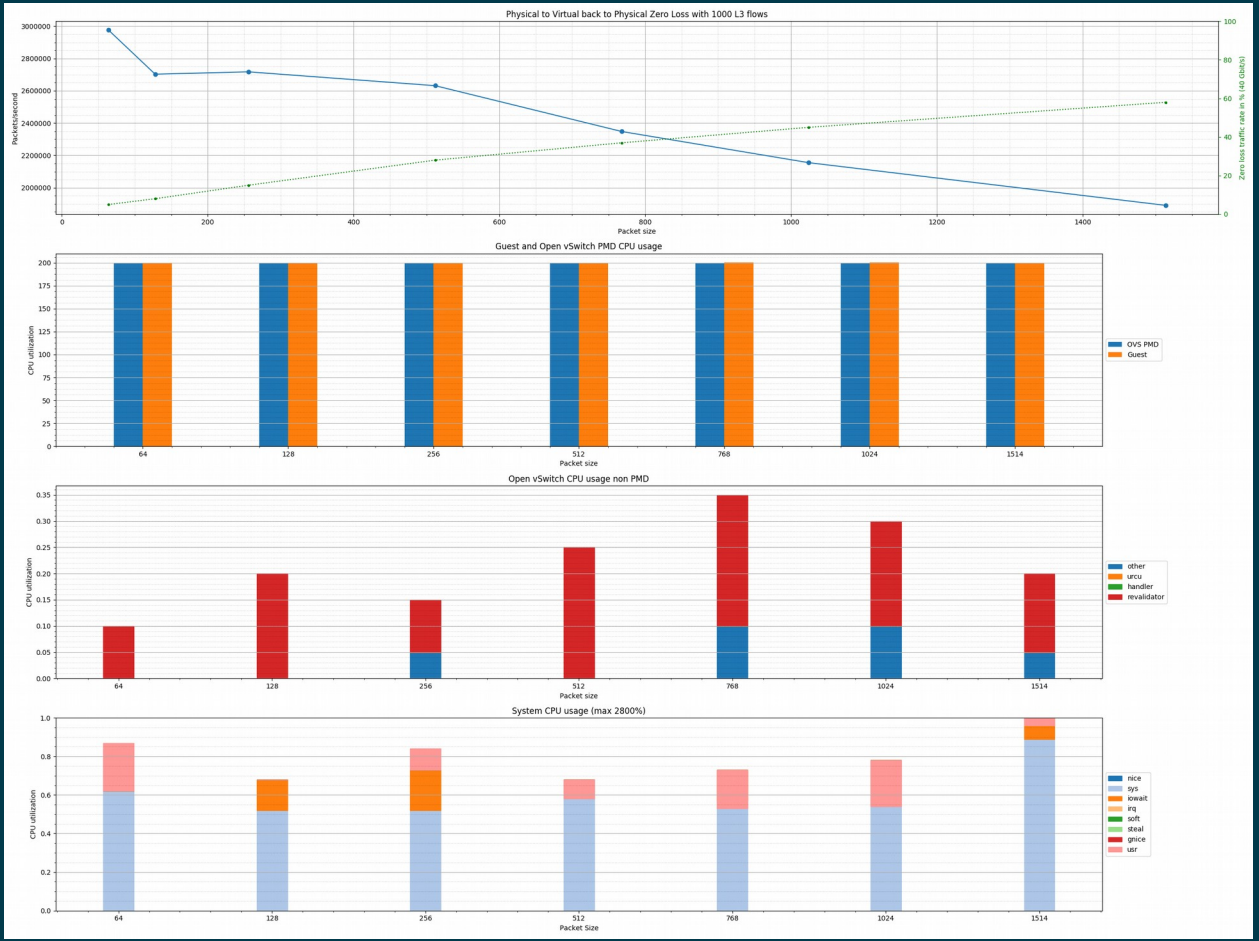


redhat.

Results

Graphs

zero packet loss

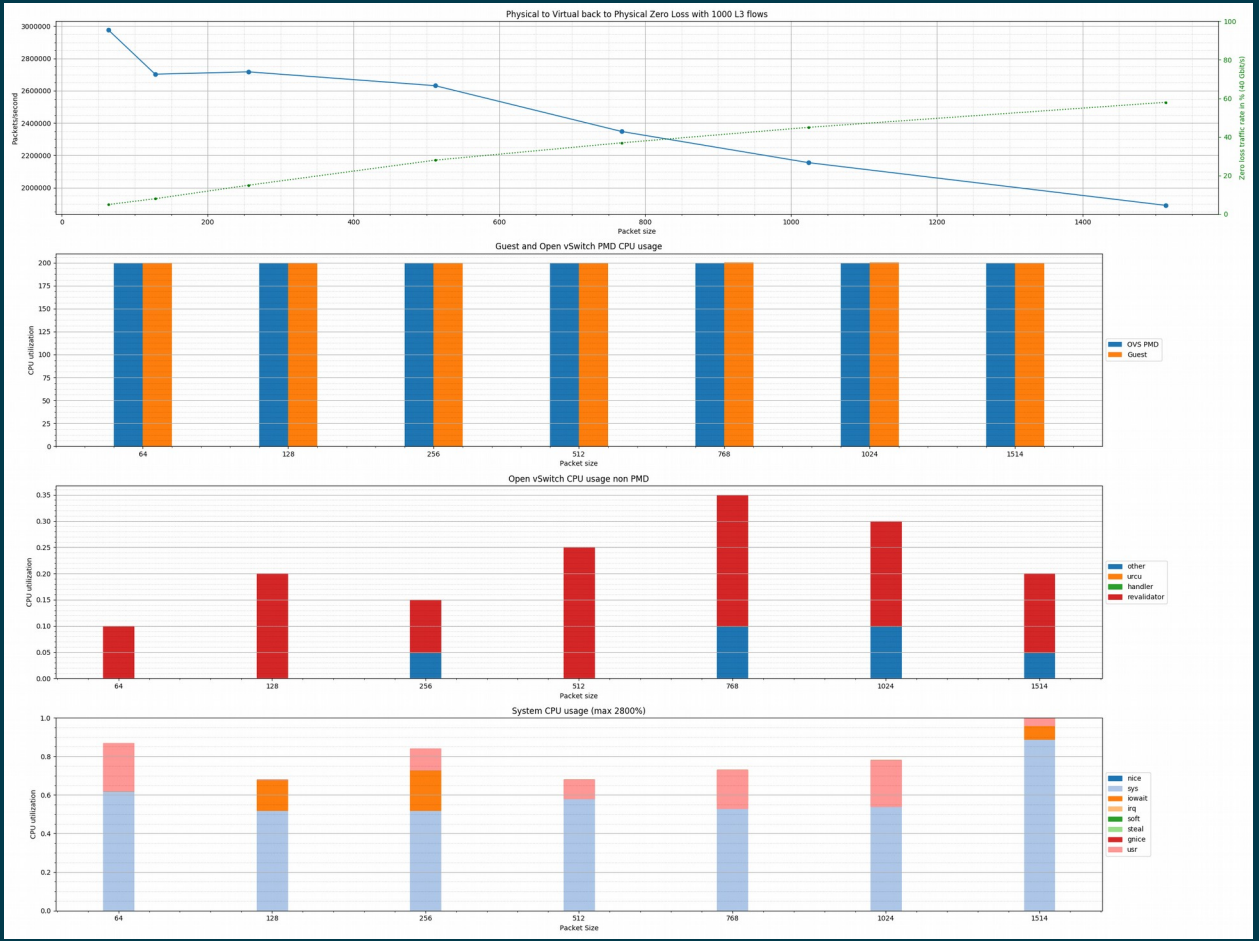


redhat.

Results

Graphs

zero packet loss



redhat.

Where to get it?

- Available on GitHub: https://github.com/chaudron/ovs_perf

redhat.

THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHat

youtube.com/user/RedHatVideos